New Variant of Ursnif Continuously Targeting Italy | FortiGuard Labs

By Xiaopeng Zhang

Published: 2021-01-12 · Archived: 2026-04-05 18:34:41 UTC

FortiGuard Labs Threat Research Report

Affected platforms:	Microsoft Windows
Impacted parties:	Windows Users in Italy
Impact:	Collects Victims' Information
Severity level:	Critical

Ursnif (also known as Gozi) is identified as a banking Trojan, but its variants also include components (backdoors, spyware, file injectors, etc.) capable of a wide variety of behaviors.

The Ursnif Trojan has been observed targeting Italy over the past year. A few days ago, FortiGuard Labs detected a phishing campaign in the wild that was spreading a fresh variant of the Ursnif Trojan via an attached MS Word document that is continuously targeting Italy.

Although Ursnif is identified as a banking Trojan, due to its C2 server's shutdown, this latest variant has been unable download the malicious banking module it needs to steal banking information from the victim, causing it to fail to start the second stage of its attack. As a result, in this post I will share my findings around the first stage of this campaign. You will learn what the phishing email looks like, how the MS Word document attached to the email works to download Ursnif, as well as what this variant does on a victim's device.

Ursnif Phishing Email

Figure 1.1 is a screenshot of the Ursnif phishing email. As you can see, it was written in Italian and masquerades as a payment reminder.

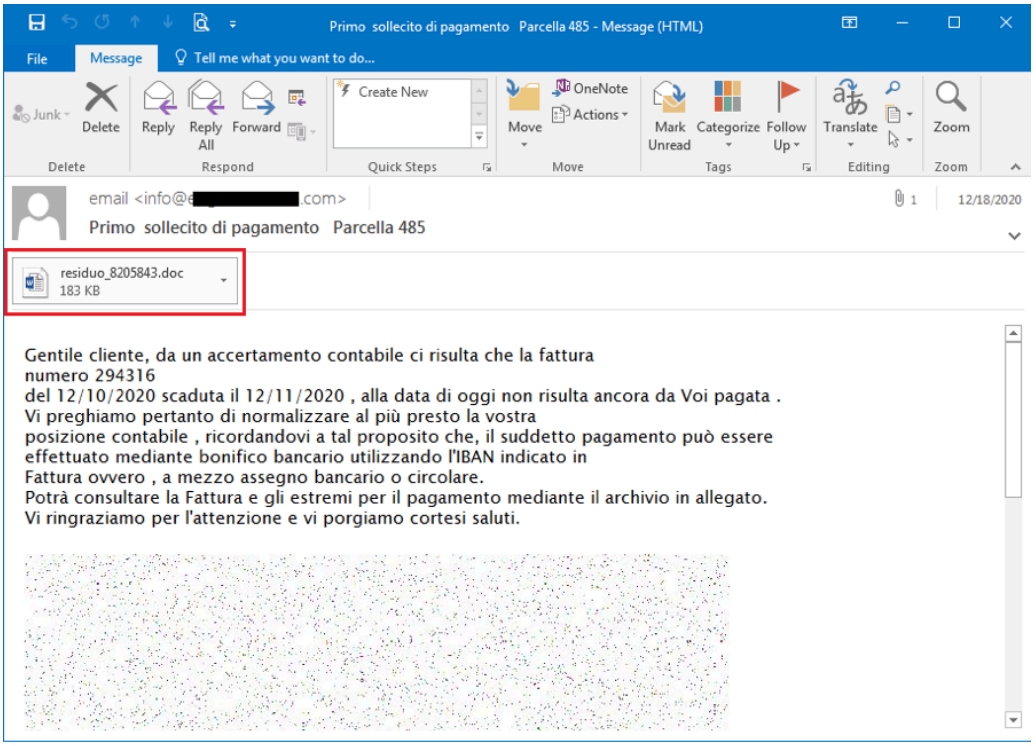

Figure 1.1. Phishing Email and attached MS Word document

I used Google Translate to translate the email content into English:

Dear customer,

A recent accounting audit shows that your invoice number 294316 of 12/10/2020 expired on 12/11/2020. As of today, it is not yet been paid by you.

Therefore, please normalize your accounting position as soon as possible. We are also reminding you that this payment can be made by bank transfer using the IBAN indicated in the invoice or, by bank check or bank draft.

You can consult the invoice and the details for the payment through the attached archive.

We thank you for your attention and we send you kind regards.

Attached to the email is an MS Word document named “residuo_8205843.doc”. The text lures the victim into opening the document to get more details of the invoice.

Word Document Analysis

As you may have guessed, the Word document contains malicious Macros. Once the victim opens the document in MS Word, it pops up a yellow warning bar to alert the user that the file contains Macros, as shown in Figure 2.1.

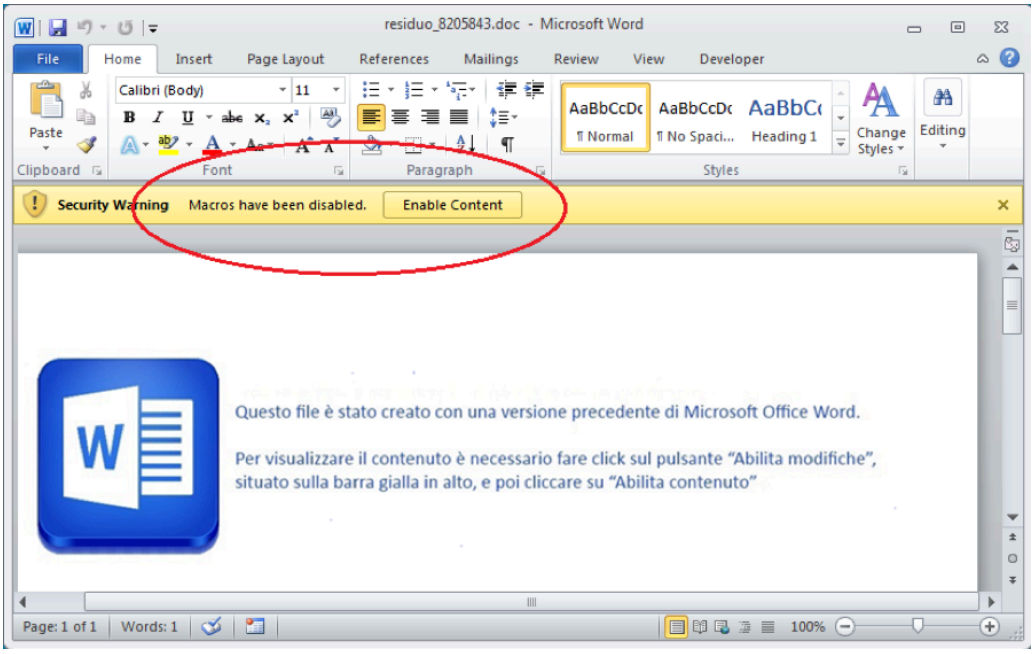

Figure 2.1. Opening the attached document in MS Word program

Once the victim clicks the button to enable the Macros, malicious Macro code will be executed in background. A built-in function, “Document_Open()”, is automatically called first when the document is opened, as shown in Figure 2.2.

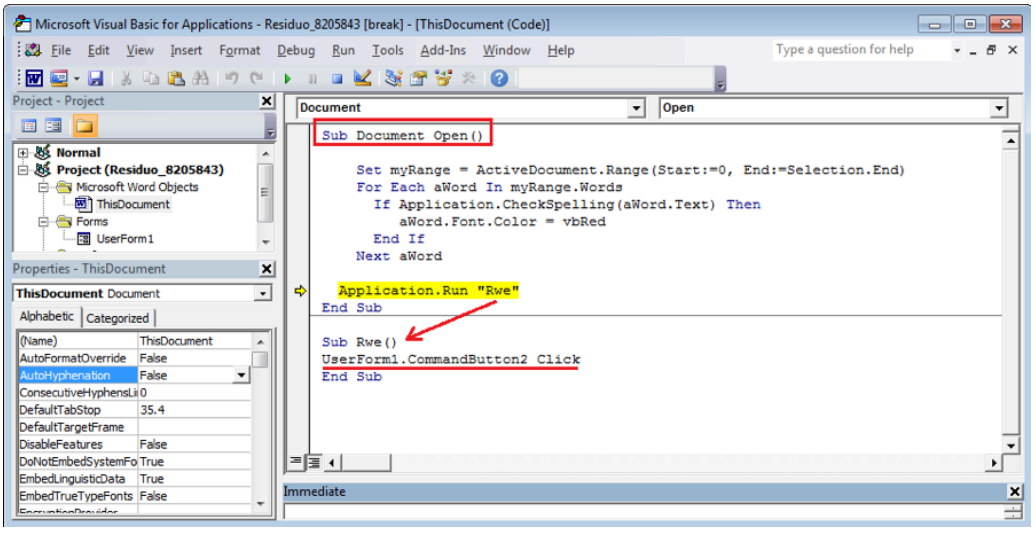

Figure 2.2. Built-in function Document_Open()

I noticed that the attacker creates a hidden UserForm (UserForm1) in the project, and a button's click event function is invoked from the Document_Open() function.

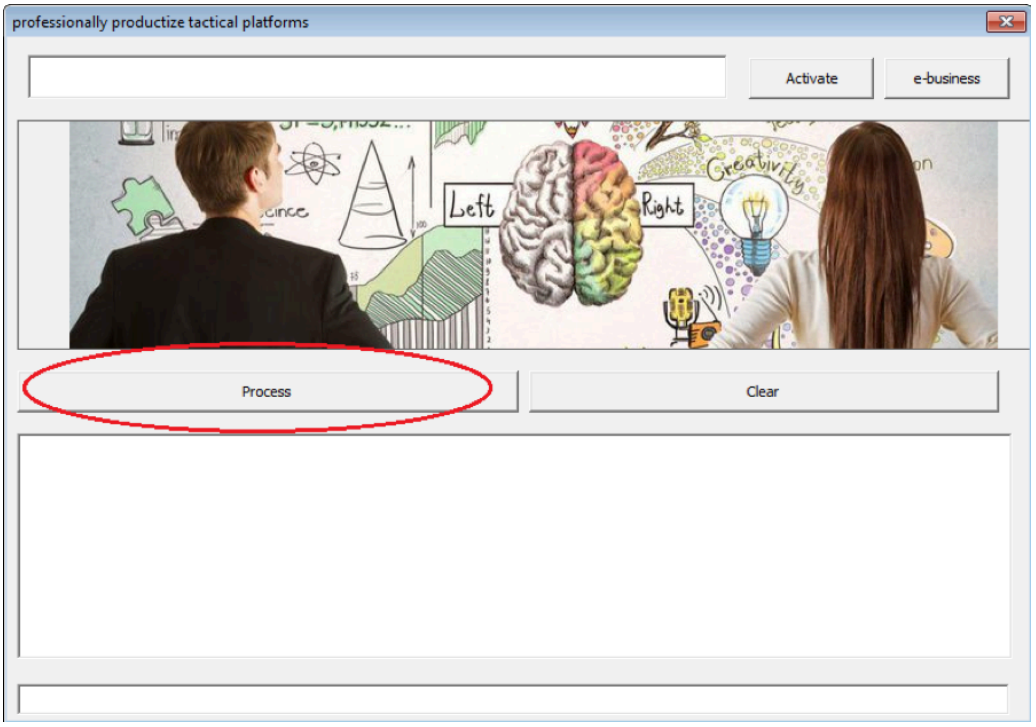

Figure 2.3. Hidden UserForm

The main work of the Macro is implemented using this form. In Figure 2.2, we can see that it calls "UserForm1.CommandButton2_Click", which belongs to the event function of the "Process" button, as shown in Figure 2.3. It functions the same as when the victim clicks the "Process" button. Other clicked buttons also directly call their event functions in code.

The Macro then downloads a DLL file from a hardcoded URL, "longline[.]cyou/p1cture3[.]jpg" (refer to Figure 2.4) and resaves it into the file "C:\users\public\px.dat". The final step the Macro performs is to run the downloaded DLL using the Windows program "RegSvr32.exe". Figure 2.4 is the screenshot of when the Macro runs this DLL.

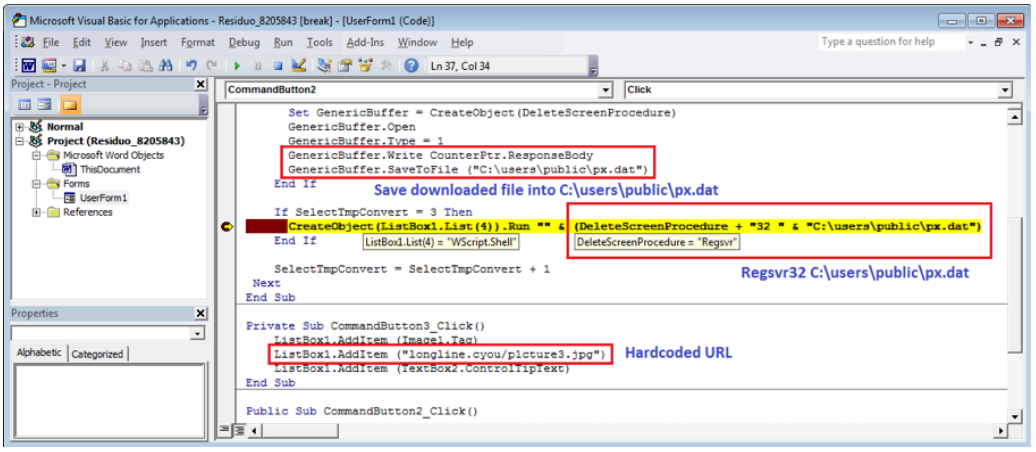

Figure 2.4. Downloading and Running a DLL file from the hardcoded URL

Analyzing the Downloaded DLL File

Like most other malware protected by a packer, this downloaded DLL file (C:\users\public\px.dat) is similarly protected by a packer as well. It is started by RegSvr32.exe, and the unpack program gets called to first extract Ursnif into the memory.

Ursnif's DllEntryPoint() function will be called at the end by the unpack program. Figure 3.1 is a screenshot of the unpacked DllEntryPoint() function.

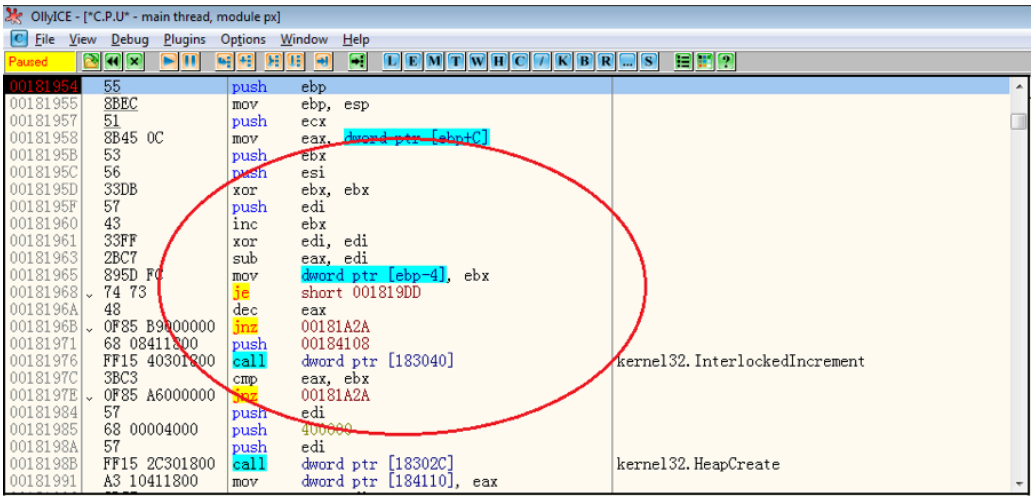

Figure 3.1. Unpacked DllEntryPoint() function

According to my analysis, this function's main task is to decompress another PE file that is secretly kept in the ".reloc" section. Next, it loads this PE file into a file mapping memory by calling API CreateFileMappingW() and MapViewOfFile()) and executing the code in it. This decompressed PE file is the core module of this variant of Ursnif.

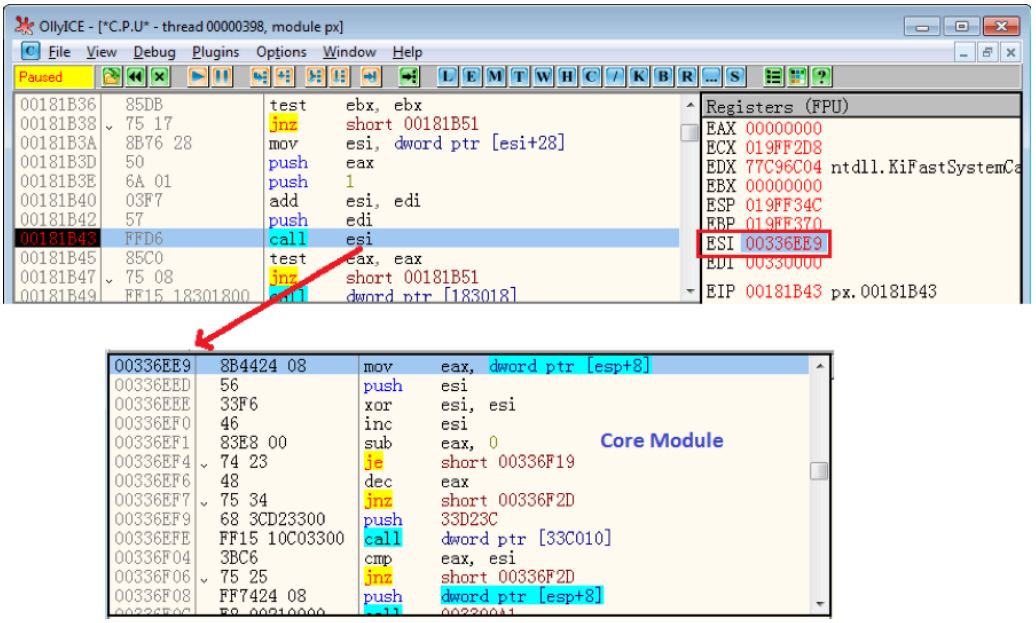

Figure 3.2. Call core module's entry function

In Figure 3.2, we can see that the malware is about to call the entry function of the core module that has been loaded at the base address 0x330000.

Decrypting the Configuration Block in the Core Module

Ursnif has an encrypted configuration block inside the PE's ".bss" section to protect its data from easy analysis. So, decrypting the configuration block is the first thing that it needs to do. The code snippet responsible for decrypting the configuration block is shown in Figure 4.1, below.

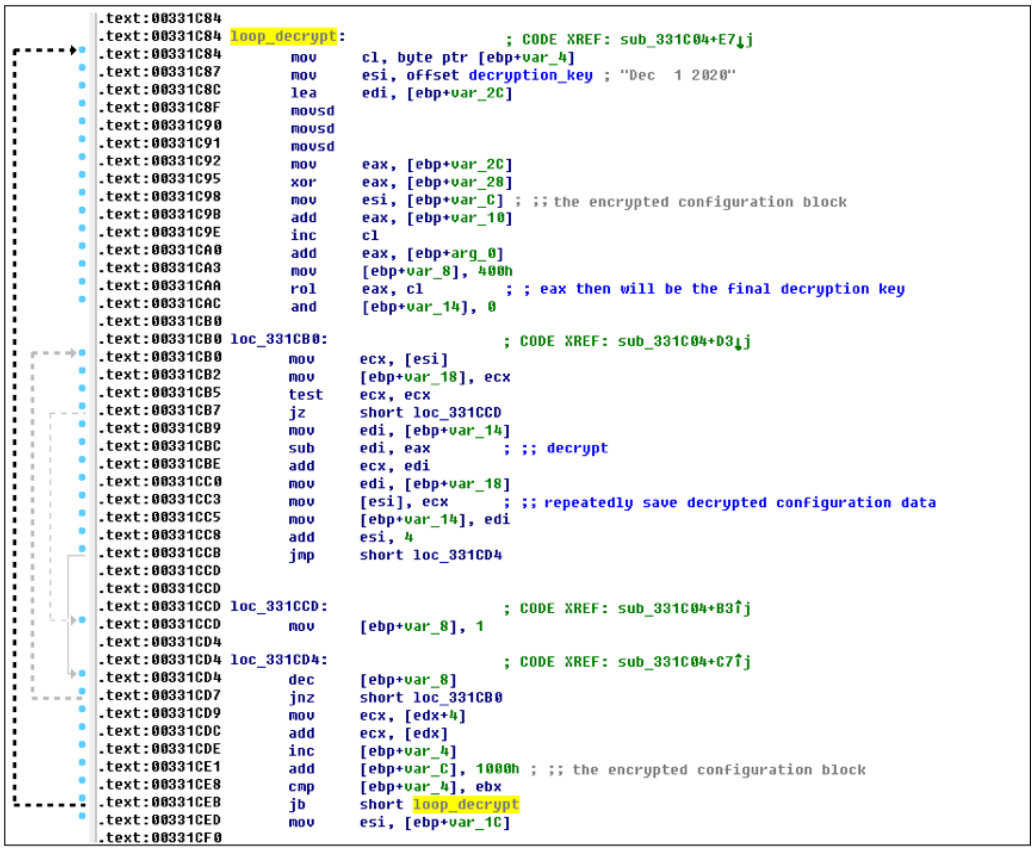

Figure 4.1. Code snippet for decrypting the configuration block

Going through the decrypted configuration block, we can see that it contains a number of ASCII and Unicode strings, as well as binary data. The values from the configuration block are accessed throughout the core module. Let’s see what it can get from the decrypted configuration block. Following is a partial list of its strings.

"invalidcert"

"overridelink"

"%08X-%04X-%04X-%04X-%08X%04X"

"StdRegProv"

"/images/"

"version=%u&soft=%u&user=%08x%08x%08x%08x&server=%u&id=%u&type=%u&"

"name=%s"

"Content-Disposition: form-data; name="upload_file"; filename="%s""

"DeleteKey"

"Mozilla/4.0 (compatible; MSIE 8.0; Windows NT %u.%u%s)"

"soft=%u&version=%u&user=%08x%08x%08x%08x&server=%u&id=%u&crc=%x"

"&uptime=%u"

"CreateProcessA"

".avi"

"Software\Microsoft\Windows\CurrentVersion\Run"

"rundll32 "%s",%S"

"&ip=%s"

"&os=%s"

"&tor=1"

"&dns=%s"

"&whoami=%s"

"/C "copy "%s" "%s" /y && rundll32 "%s",%S""

"SOFTWARE\Microsoft\Windows NT\CurrentVersion"

"WScript.Shell"

");%S.Run("

"powershell iex ([System.Text.Encoding]::ASCII.GetString(("

"IE8RunOnceLastShown_TIMESTAMP"

"SOFTWARE\Microsoft\Internet Explorer\"

…

Collecting Sensitive Information and Sending It to the C2 Server

In the core module, Ursnif collects sensitive information from the victim's device, such as current login Username, Computer Name, System Uptime, and so on. These are formatted as in the first packets sent to the C2 server. Figure 5.1 shows the ASM code snippet used to obtain Username and Computer Name.

```
.text:0033614A
.text:0033614A          push    ebp
.text:0033614B          mov     ebp, esp
.text:0033614D          sub     esp, 0Ch
.text:00336150          push    ebx
.text:00336151          lea     eax, [ebp+nSize]
.text:00336154          xor     ebx, ebx
.text:00336156          push    eax             ; nSize
.text:00336157          push    ebx             ; lpBuffer
.text:00336158          mov     [ebp+var_C], ebx
.text:0033615B          mov     [ebp+nSize], ebx
.text:0033615E          call    GetUserNameW
.text:00336164          mov     eax, [ebp+nSize]
.text:00336167          cmp     eax, ebx
.text:00336169          jz      loc_336217
.text:0033616F          mov     [ebp+var_8], eax
.text:00336172          lea     eax, [ebp+nSize]
.text:00336175          push    eax             ; nSize
.text:00336176          push    ebx             ; lpBuffer
.text:00336177          mov     [ebp+nSize], ebx
.text:0033617A          call    ds:GetComputerNameW
.text:00336180          mov     eax, [ebp+nSize]
.text:00336183          cmp     eax, ebx
.text:00336185          jz      loc_336217
.text:0033618B          mov     ecx, [ebp+var_8]
.text:0033618E          lea     eax, [ecx+eax+2]
.text:00336192          mov     [ebp+var_8], eax
.text:00336195          push    edi
.text:00336196          shl     eax, 2
.text:00336199          push    eax             ; dwBytes
.text:0033619A          call    sub_3358BE      ; HeapAlloc
.text:0033619F          mov     edi, eax
```

Figure 5.1. Calling APIs to obtain Username and Computer Name

It formats all of the collected data into key=value pairs, like the string shown in the memory section of Figure 5.2.

Figure 5.2. Collected data in a formatted string

Among the key-value pairs, the values of “soft”, “version”, and “crc” are hardcoded. The values of “user”, “server”, and “id” are hash values. “uptime” is a time value of how long the device has been running. “dns” is the computer name, and “whoami” is the full user name.

Next, Ursnif encrypts the string and encodes the cipher text with base64. Finally, it transforms the base64 string by replacing specified bytes and randomly inserting “/”. An example looks like this:

kGmre5fgzV/jtkp5rBSbtChvz3cy/avjHTlxQBzmZ/DH4YpgdvFWT/3ZOkljxvOGl5sK/DKS8qy7zKWSwKdmWKVkNW/3iLRdsyjEatQ2kbQ/97eIek93
TCOCQKSzO32C/EUgtTKuNa/PDAfPobZLDuYwMNjMJqk/BCloHCcWrST2Oq7lSc7/bX0t3dF5oQwhDoo8O83JoQ/KNQw3uRXGoOK3/bvb2q8
PoIH_2/F

It then adds a prefix string (the domain of the C2 server and “/images/”) and suffix string (“.avi”) before sending the information to the server.

Ursnif does not directly send the data within the process. Instead, it uses a COM component—“IE ActiveX Interface”—that is implemented in the module “IEProxy.dll”. Ursnif indirectly calls APIs from this module to perform communication with C2 server. This means the collected data will be sent by a newly created “iexplorer.exe” process that is started by the COM component. Figure 5.3 provides the detailed overview processes tree, from which you can not only observe the process “iexplorer.exe”, but also the relationship between the processes mentioned in this blog, such as “WinWord.exe” and “RegSvr32.exe”.

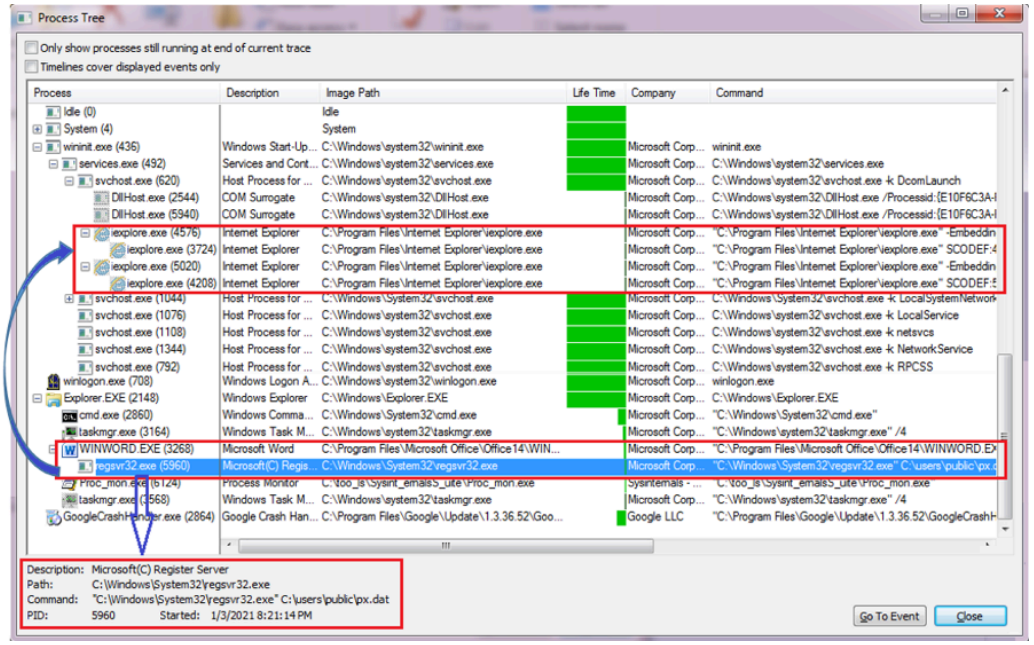

Figure 5.3. Process Tree of calling a COM component

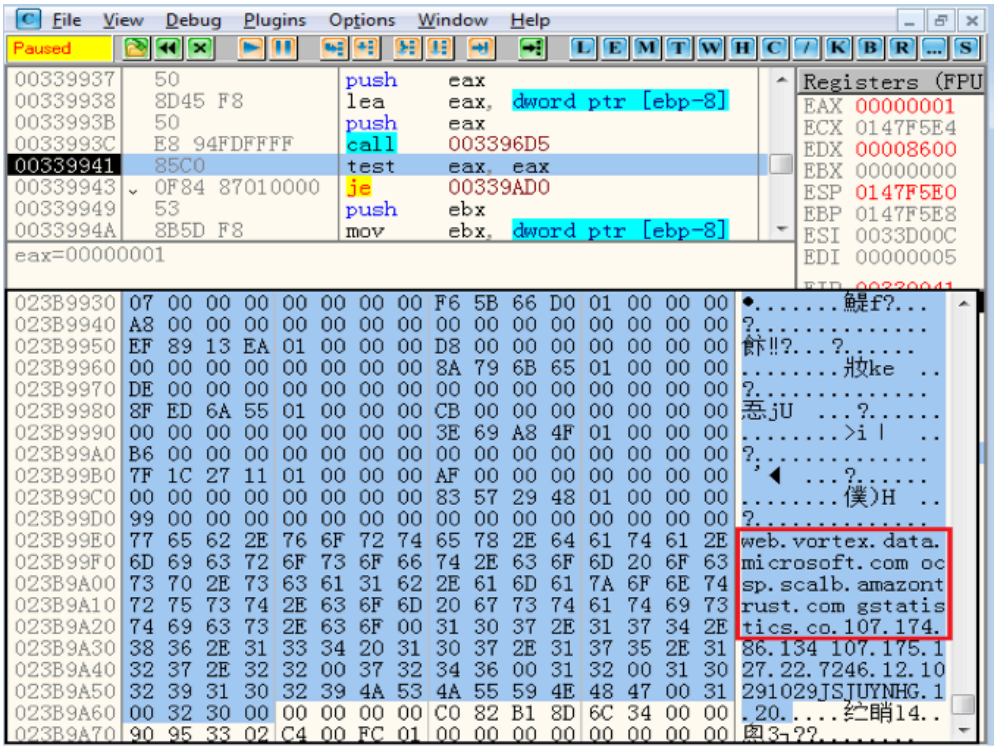

Figure 5.4. Decrypted C2 server domain string

The domain string of the C2 server comes from a secret data structure that is decrypted from a data block within the ".reloc" section.

In Figure 5.4, at the bottom of the memory segment, we can see the decrypted secret data structure. The string starting at offset 0xB0 are the domains: "web[.]vortex[.]data[.]Microsoft[.]com", "ocsp[.]sca1b[.]amazontrust[.]com", and "gstatistics[.]co". Through my analysis, only the last one is the real C2 server domain.

Figure 5.5 shows what the final packet sent to the C2 server looks like.

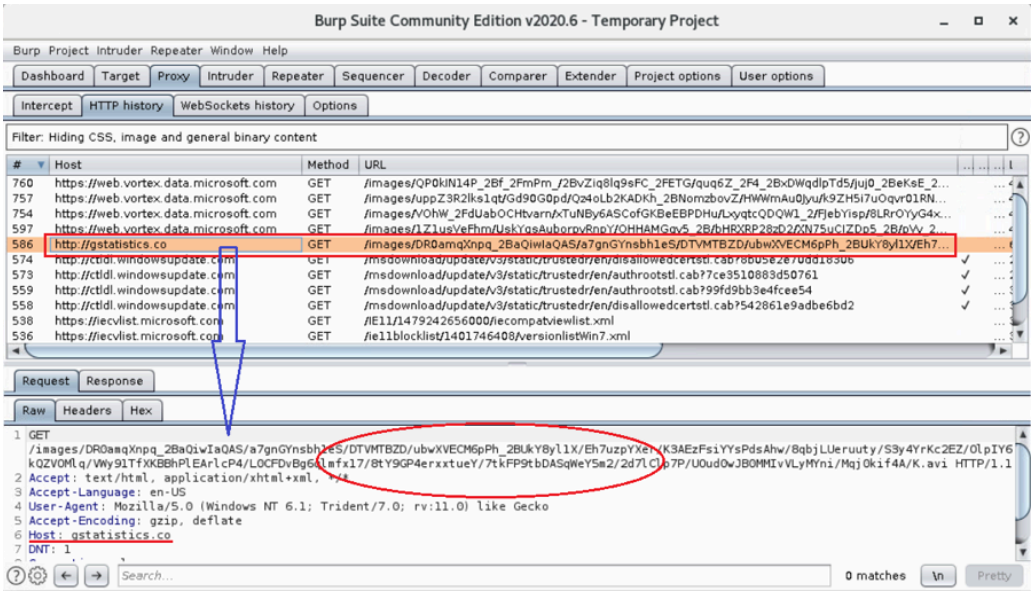

Figure 5.5. Packet sent to the C2 server

In the next stage, the malware would respond to the received packet with a module (dll, exe, etc.) which would be executed by Ursnif to perform further malicious attacks on victim's device. Unfortunately, we did not receive any response because the C2 server had been shut down. I will continue to try and contact the C2 server. If I am finally able to obtain the module file, I will post my analysis of it as well.

Fortinet Solution Coverage:

The Word document attached to the phishing email has been detected as "**VBA/Ursinf.3412!tr**" and the downloaded file has been detected as "**W32/Ursinf.KB!tr**" by the FortiGuard AntiVirus service.

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

The CDR (Content Disarm & Reconstruction) feature can also neutralize this threat by removing all malicious Macro code.

The FortiGuard AntiVirus service is supported by FortiGate, FortiMail, FortiClient and FortiEDR. And the CDR feature is supported by FortiGate and FortiMail.

We also suggest our readers to go through the free NSE training -- NSE 1 – Information Security Awareness, which has a module on Internet threats designed to help end users learn how to identify and protect themselves from phishing attacks.

IoCs:

URLs:

"hxxp://longline[.]cyou/p1cture3[.]jpg "

"hxxps://gstatistics[.]co/"

Sample SHA256:

residuo_8205843.doc:

E9732CDCA1B2503E02E8FEA9A4C68EDA940E10890E1C5ABE2CEB2290FE39C3DB

Downloaded DLL file (px.dat or p1cture3.jpg):

90D8648B2AAC0C837286A4C042F02064CFBB12F45B3DC6B00B2BECCC7FC35422

Learn more about FortiGuard Labs threat research and the FortiGuard Security Subscriptions and Services portfolio.

Learn more about Fortinet's free cybersecurity training initiative or about the Fortinet NSE Training program, Security Academy program, and Veterans program.
