The Latest Remcos RAT Driven By Phishing Campaign | FortiGuard Labs

By Xiaopeng Zhang

Published: 2022-04-06 · Archived: 2026-04-05 17:11:47 UTC

Remcos RAT (Remote Access Trojan) was originally designed as a professional tool to remotely control computers. Remcos RAT is recognized as a malware family because it has been abused by hackers to secretly control victims' devices since its first version was published on July 21, 2016. Remcos RAT is commercial software that is sold online.

Affected platforms: Microsoft Windows
Impacted parties: Microsoft Windows Users
Impact: Controls victim's device and collects sensitive information
Severity level: Critical

Figure 1: Example of Remcos RAT being sold online

On this webpage, it provides two versions: professional edition (with all features included) and free edition (with restricted features).

This analysis is based on Remcos RAT being used by hackers to control victims' devices delivered by a phishing campaign, which was caught by Fortinet's FortiGuard Labs recently.

In this analysis, you will learn:

- How the phishing campaign delivers Remcos RAT onto the victim's device
- How Remcos executes on the device
- What sensitive information it could steal from a victim
- How Remcos connects to its C2 server
- What commands this Remcos provides to control the victim's device

The Phishing Email

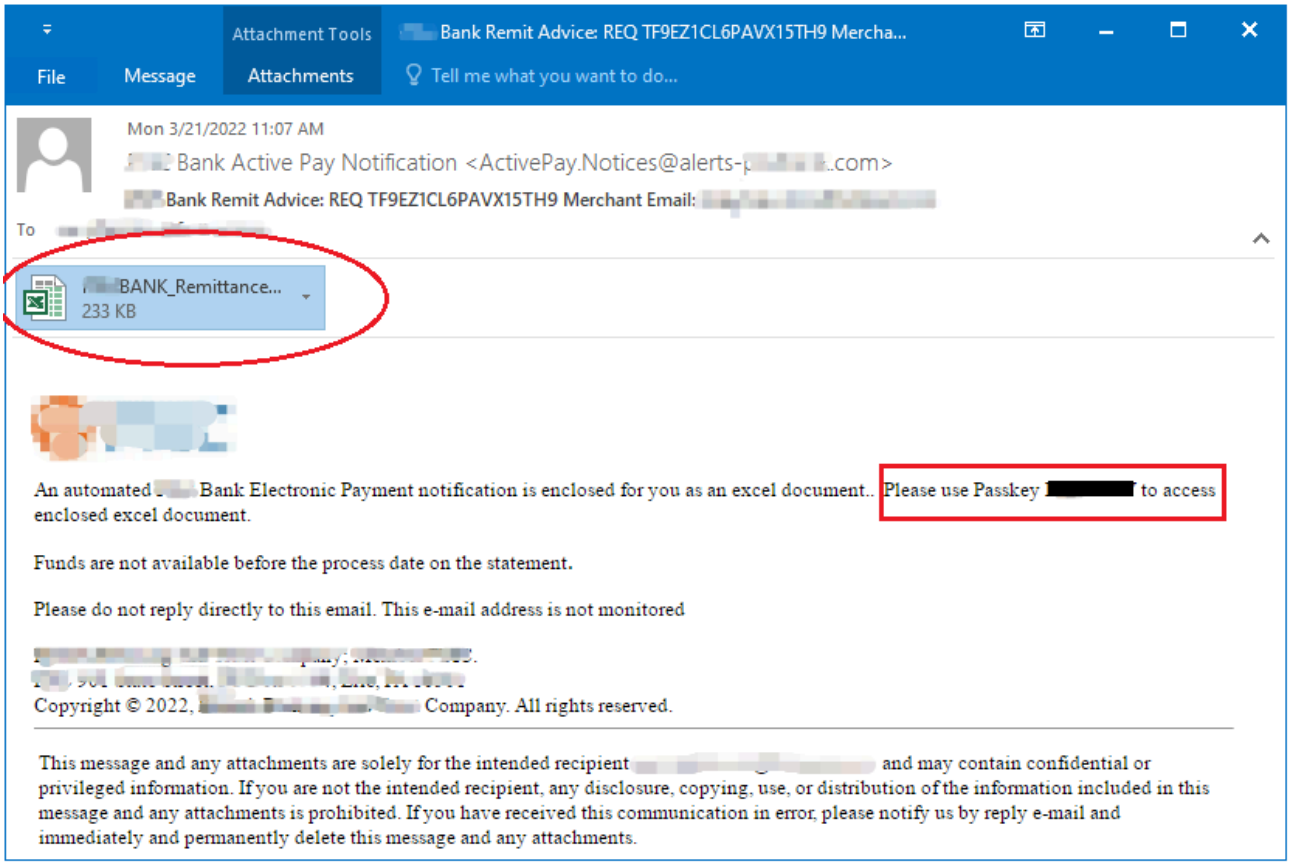

Figure 2: Screenshot of the phishing email content

As you can see from the email content shown in Figure 2, the hacker disguised the phishing email as a payment notification from a trusted bank and asked the recipient to open the attached Excel file that is protected by a password.

Excel File Leads to Download of Remcos via VBS and PowerShell

Once the attached Excel document is opened in the Excel program, it asks for a password to view the document, which has already been provided in the email. It then shows the document in the Excel program like Figure 3. Because the file contains Macro code, it shows a yellow security warning bar to warn the victim of the danger.

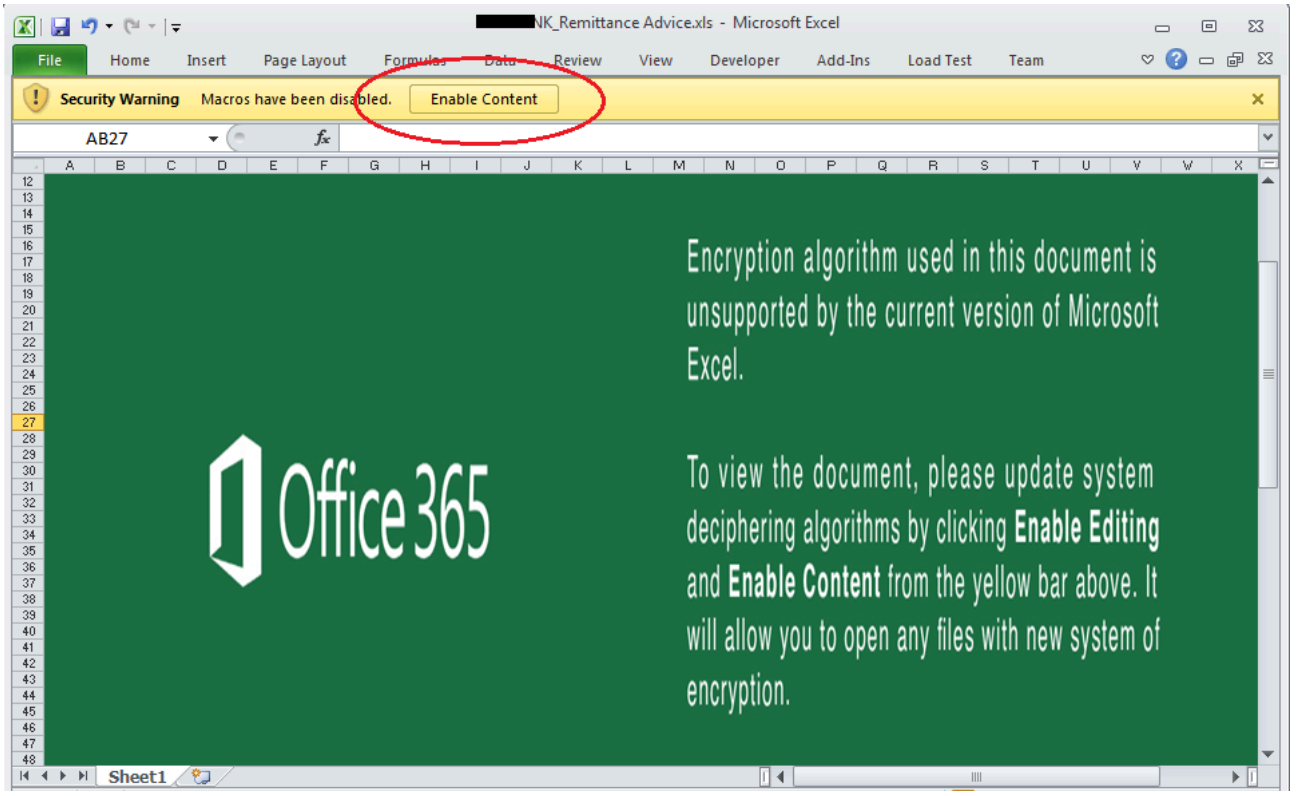

Figure 3: Screen shown when the Excel document is opened in the Excel program

The file message lures the victim into clicking the Enable Content button to bypass the warning and execute the malicious macro code.

The macro has a function called “Wookbook_Active()” that is called automatically when it opens. Its task is to extract VBS code from the cells into a file “%AppData%\HobYQ.vbs” and then execute it.

To protect the Remcos payload file, it uses a super sophisticated way to download it. In this way, it executes both VBS and PowerShell script codes.

“HobYQ.vbs” runs a segment of dynamically spliced PowerShell code to download another VBS file (“flip.vbs”) from the attacker’s server and run it. Next, “flip.vbs” continues to download a file (called “mem.txt”) from the server, which is a piece of encoded VBS code that will be executed later in “flip.vbs” to download the final file from the same server, which is called “faze.jpg”. In Figure 4, it shows the captured traffic for the three downloaded files, “flip.vbs,” “mem.txt,” and “faze.jpg.”

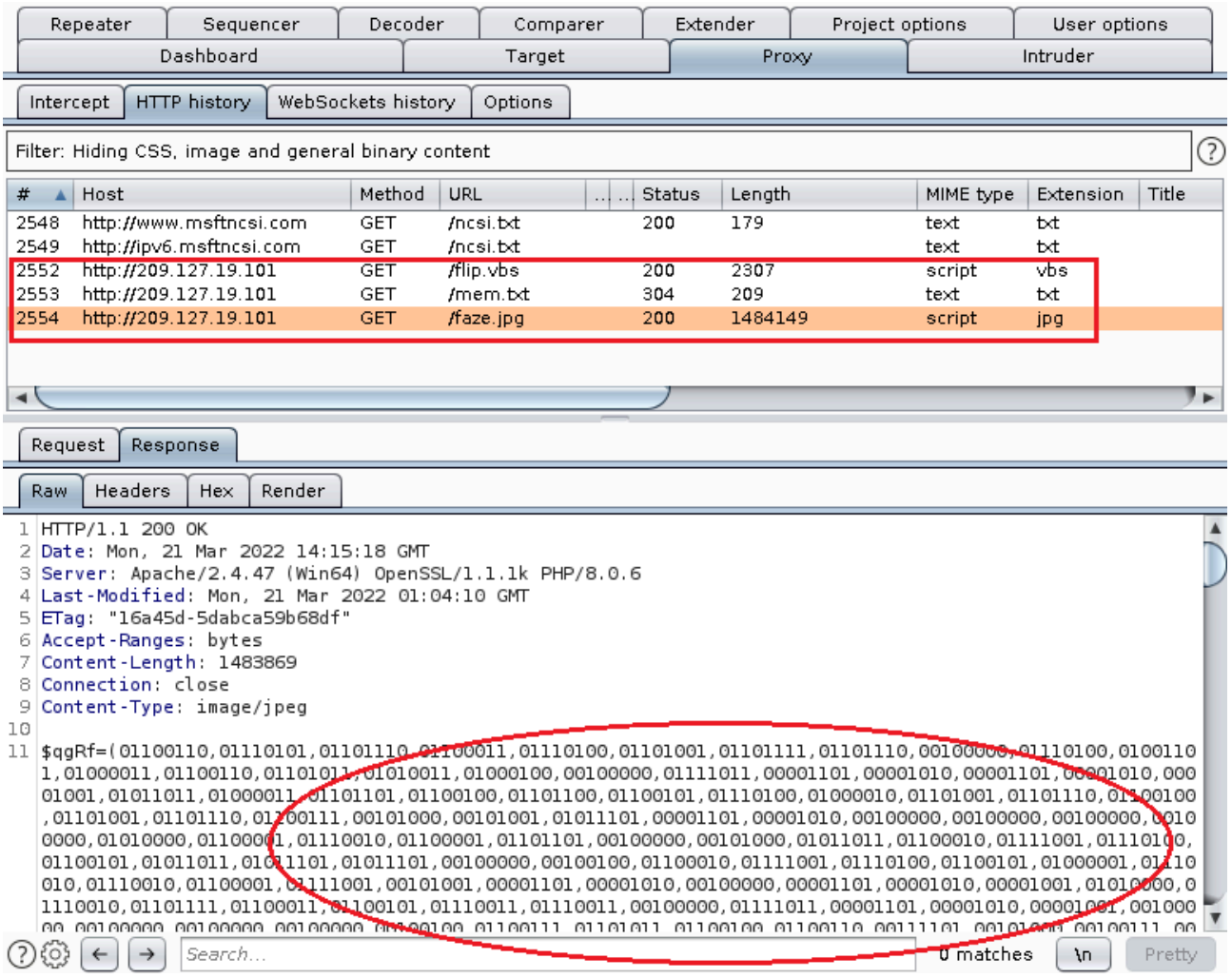

Figure 4: “HobYQ.vbs” leads to downloading three files

The bottom of Figure 4 shows part of the response packet of “faze.jpg”. Of course, it is not image file, but an obfuscated PowerShell code file. There are three pieces of encoded data defined in three array variables, which have been simplified in Figure 5 in three red boxes. The PowerShell code that is carried in “faze.jpg” is executed by “flip.vbs”.

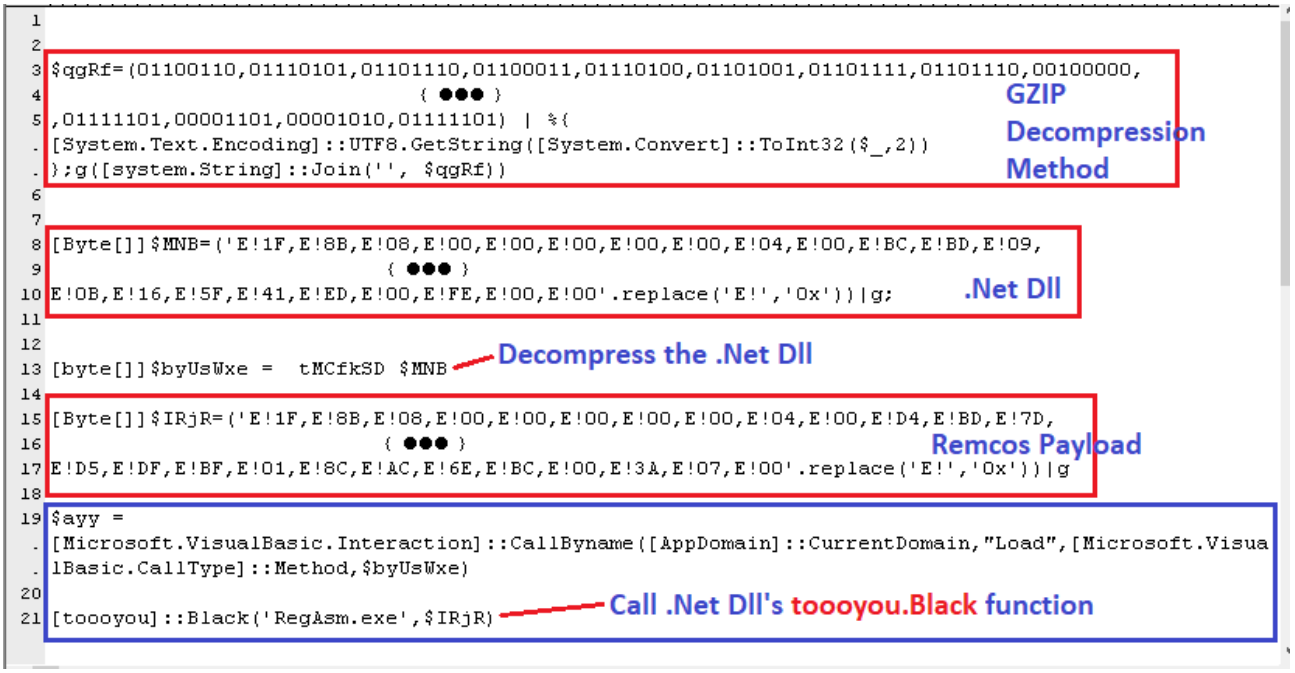

Figure 5: The simplified PowerShell code of “faze.jpg”

Let me then explain how the PowerShell code works here.

The values for the two variables, $MNB and $IRjR, are both encoded GZIP compression payloads (they start with “1F 8B…”). After decompression, $MNB is .Net Framework Dll file and $IRjR is the Remcos payload file.

The binary value that is set to variable “$qgRf” is a dynamic method called tMCfkSD() for decompression.

It calls tMCfkSD() to decompress the .Net Dll from $MNB into $byUsWxe. At last, it loads the .Net Dll into current PowerShell execution environment by calling “Load” and the function “Black()” from class “toooyou” is called with “RegAsm.exe” and compressed Remcos Payload ($IRjR).

.Net Framework Dll File Performs Process Hollowing

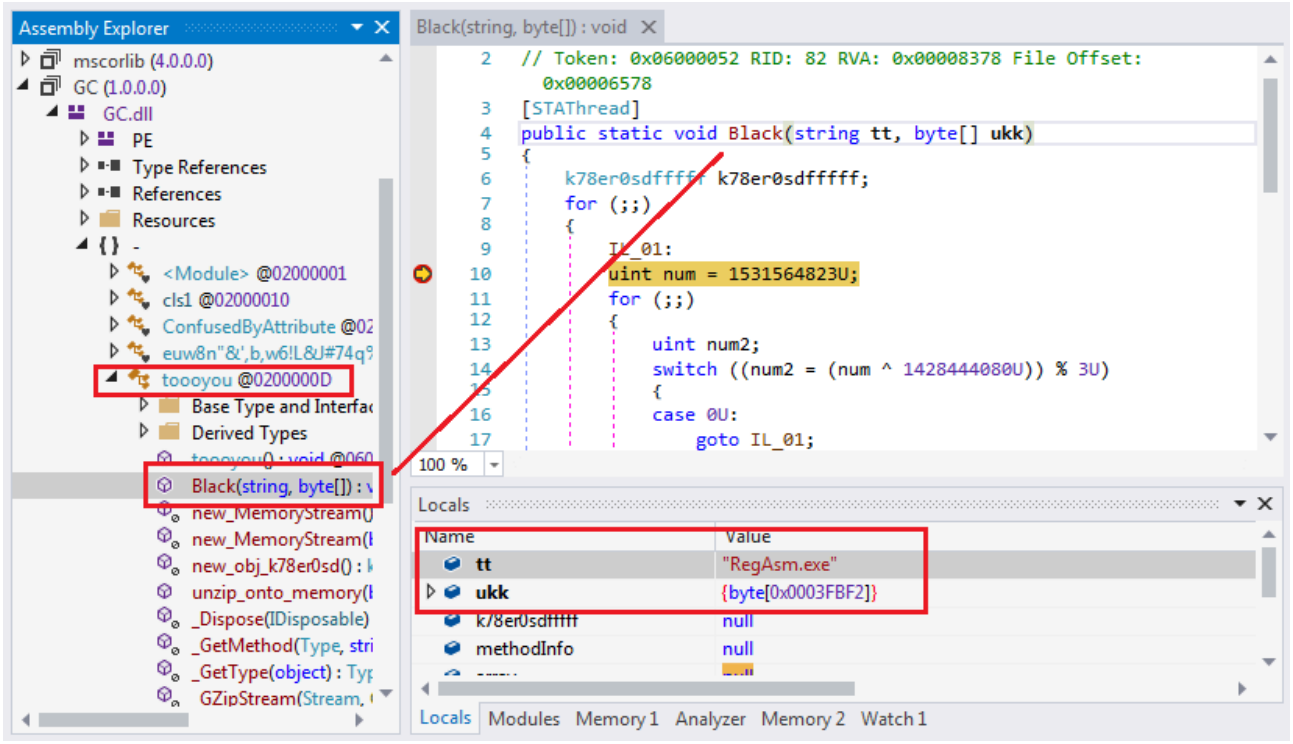

Figure 6: Break on .Net Dll toooyou.Black()

The .Net Dll is named GC.dll as you can see in Figure 6. The two passed parameters are shown in “Locals”. Its code is obfuscated. According to my analysis, it first dynamically extracts another Dll from its resource section named lime.dll. Next, it decompresses the Remcos payload, which will be passed to a function called "k78er0sdfffff.o70sdaf45gfg(System.String, Byte[])" that is from lime.dll at the time the function is called. Actually, this Dll is used to perform the process hollowing that is injecting the Remcos payload into a newly-created “RegAsm.exe” process. Once the function (k78er0sdfffff.o70sdaf45gfg()) is invoked, it finds “RegAsm.exe” from below locations on the victim’s device. In case that it fails to find the file, it exits from PowerShell without running the Remcos.

The hardcoded location list:

array[<Module>.C1790263187] = "C:\\WINDOWS\\syswow64\\";
array[<Module>.C2710025604] = "C:\\WINDOWS\\system32\\";
array[<Module>.C3326009313] = "C:\\WINDOWS\\";
array[<Module>.C931285936] = "C:\\WINDOWS\\syswow64\\WindowsPowerShell\\v1.0\\";
array[<Module>.const_4] = "C:\\WINDOWS\\system32\\WindowsPowerShell\\v1.0\\";
array[<Module>.C3873335087] = "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\";

In my testing environment, it has this file at "C:\Windows\\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe".

As you may know, it needs to call several APIs to finish the process hollowing, which are: CreateProcess() with CREATE_SUSPENDED flag, WriteProcessMemory(), GetThreadContext(), SetThreadContext() and so on. As shown in Figure 7, it is about to call API CreateProcessA() to create a suspended RegAsm.exe process from Lime.dll.

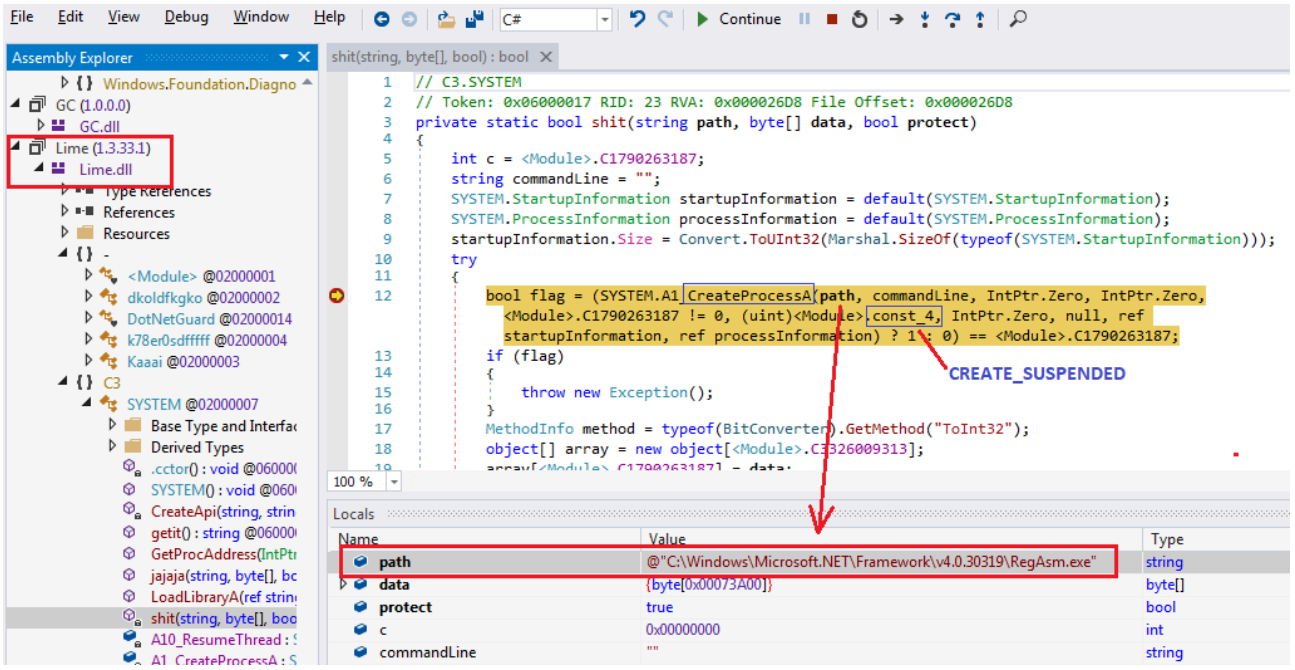

Figure 7: Call CreateProcessA() with CREATE_SUSPENDED flag

When the Remcos payload is injected and deployed onto the RegAsm.exe, API ResumeThread() will be called to have RegAsm.exe resume to run the Remcos RAT on the victim's device.

Dive into Remcos Payload

Per my analysis, Remcos was written in C++ language with templates. What we already captured is the latest version, 3.4.0 Pro, which was published on February 10, 2022. I dissected the Remcos payload file from this section to learn how it controls the victim's device.

From the analysis of its previous versions in the past years, Remcos used RC4 encryption to encrypt or decrypt both the local data and the traffic data between Remcos and C2 servers. From the version 3.0.0 Pro on, it has changed the encryption algorithm to AES-128 bit for encrypting or decrypting the traffic data. Therefore, it is now using both encryption algorithms, RC4 for local data and AES for traffic data in this variant.

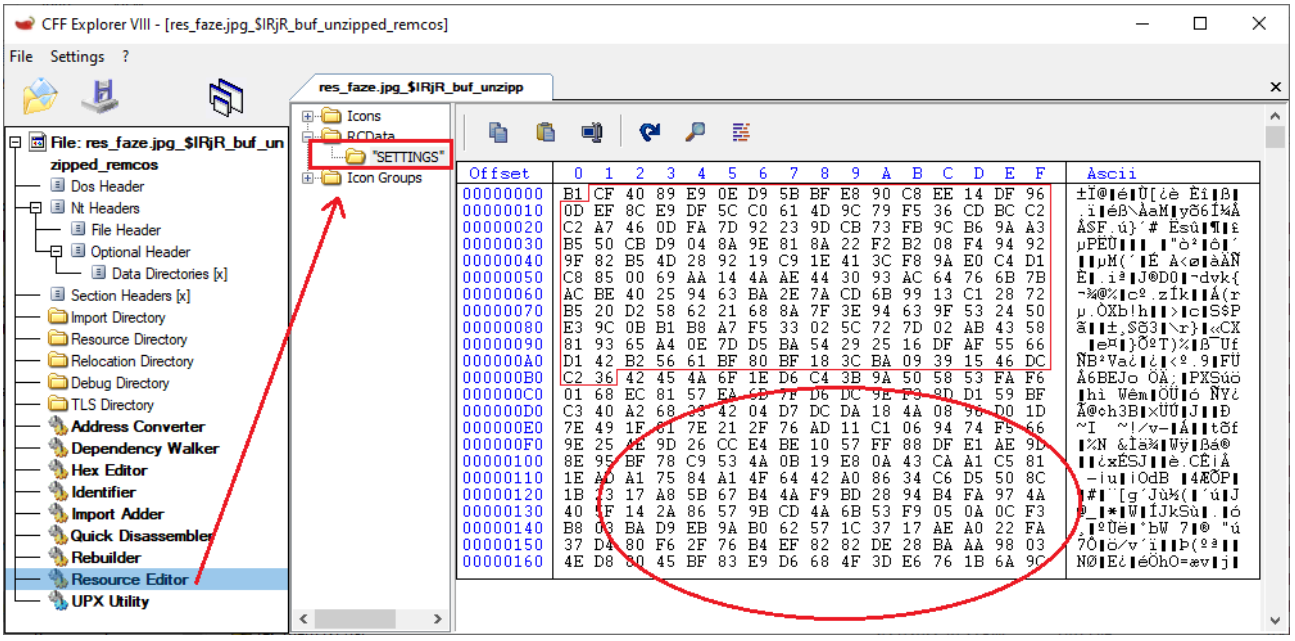

Figure 8: Remcos configuration block encrypted in “SETTINGS” resource

Every Remcos contains an RC4 encrypted configuration block in its PE resource section, named “SETTINGS” as shown in Figure 8, where the first byte “B1” is the size of the following RC4 key that is in a red box and the rest data is the encrypted Remcos configuration block.

The first thing Remcos does is to decrypt the configuration block, which will be referred to throughout Remcos lifetime. It contains but not limited to the C2’s server information, Remcos assigned name for attacker to recognize the victim, Remcos sub-key name in registry, the name of log file for recording victim’s keylogger and clipboard data, many flags telling Remcos how to start its features in the victim’s device, as well as the authentication data used to establish connection to the C2 server.

The workflow of Remcos is very clear that it starts many threads to perform auto-start work according to the flags defined in the configuration block. It includes:

- Adding Remcos to the auto-run group in the system registry
- Starting a watchdog program (Remcos’ daemon program)
- Recording the victim’s audio input from an input device (microphone)
- Capturing victim’s screenshots at startup
- Disabling UAC (User Account Control) on the victim’s device
- And so on

Remcos is able to record the victim’s sensitive information in a log file (file name is from the configuration block) from time to time, like keyboard inputs (keylogger), data on the system clipboard, and the title of the topmost program that the victim’s typing in. In order to do so, it needs to set a keyboard hook by calling API SetWindowsHookExA() and starts a thread to check every 500 microseconds. Figure 9 shows the ASM code snippet of setting such hook.

```
00408A43      push    edi
00408A44      xor     edi, edi
00408A46      mov     dword_46CAE8, esi
00408A4C      cmp     [esi], edi
00408A4E      jnz     short loc_408AB2
00408A50      push    edi             ; dwThreadId
00408A51      push    edi             ; lpModuleName
00408A52      call    ds:GetModuleHandleA
00408A58      push    eax             ; hmod
00408A59      push    offset keyboard_hook_fun ; lpfn
00408A5E      push    0Dh             ; WH_KEYBOARD_LL
00408A60      call    ds:SetWindowsHookExA ; SetWindowsHookExA
00408A66      mov     [esi], eax
00408A68      test    eax, eax
00408A6A      jnz     short loc_408AB2
00408A6C      call    ds:GetLastError
00408A72      mov     edx, eax
00408A74      lea     ecx, [esp+40h+var_34]
00408A78      call    sub_417F02
00408A7D      sub     esp, 18h
00408A80      mov     edx, offset aKeyloggerIniti ; "Keylogger initialization failure: error"..
00408A85      mov     ecx, esp
00408A87      push    eax
```

Figure 9: Set keyboard Windows Hook

Below is an example of “logs.dat” with what Remcos has obtained from my test environment, such as recording date and time, topmost program titles, victim’s idle time, and clipboard data.

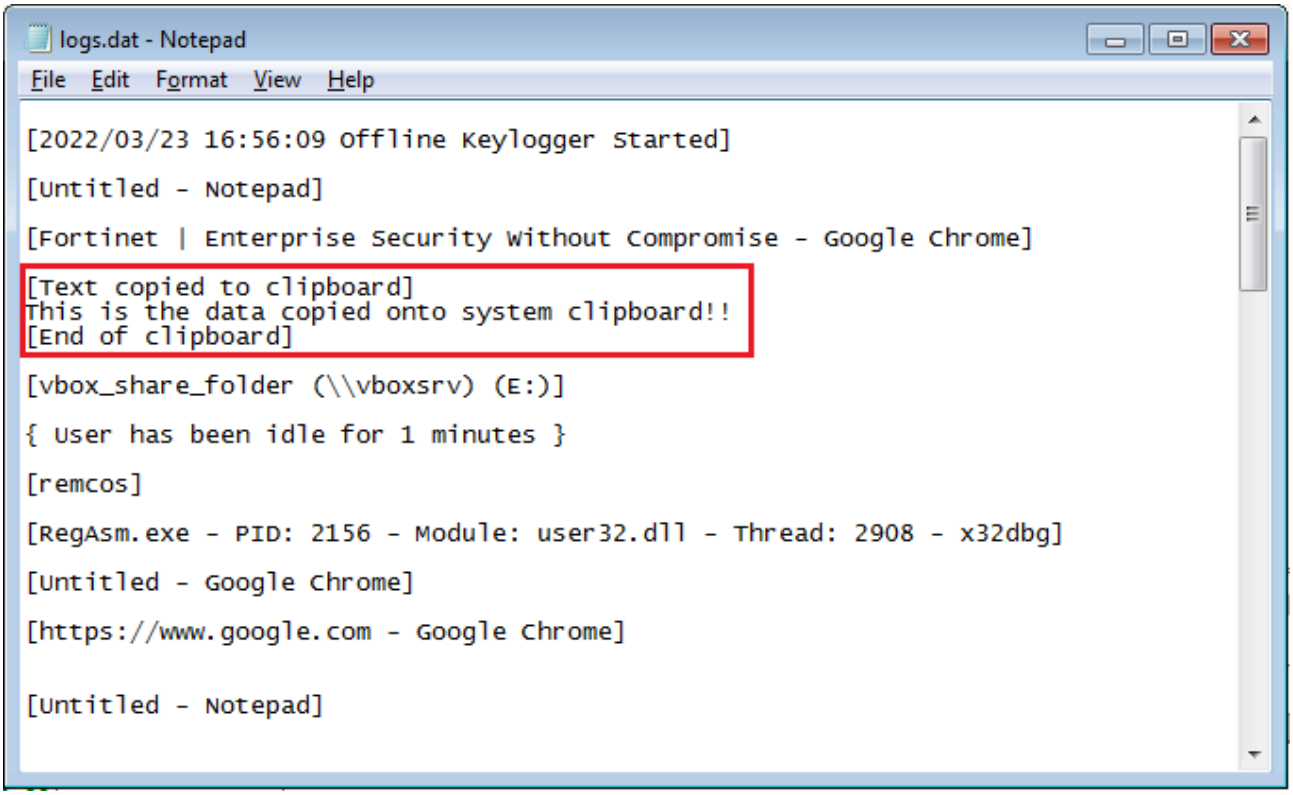

Figure: 10: Example of information saved in “log.dat”

The next step in the Remcos workflow is to connect to its C2 server per the information from the configuration block.

Communicating with the C2 Server

Remcos uses TLS v1.3 protocol to communicate with the C2 server, which is implemented by itself (not using Windows APIs) on the TLS handshake and authentication as I mentioned before.

Remcos then collects the basic information from the victim's system and submits it in the first packet to the C2 server. The packet number for the first packet is 4BH. The packet to go through AES encryption is shown below.

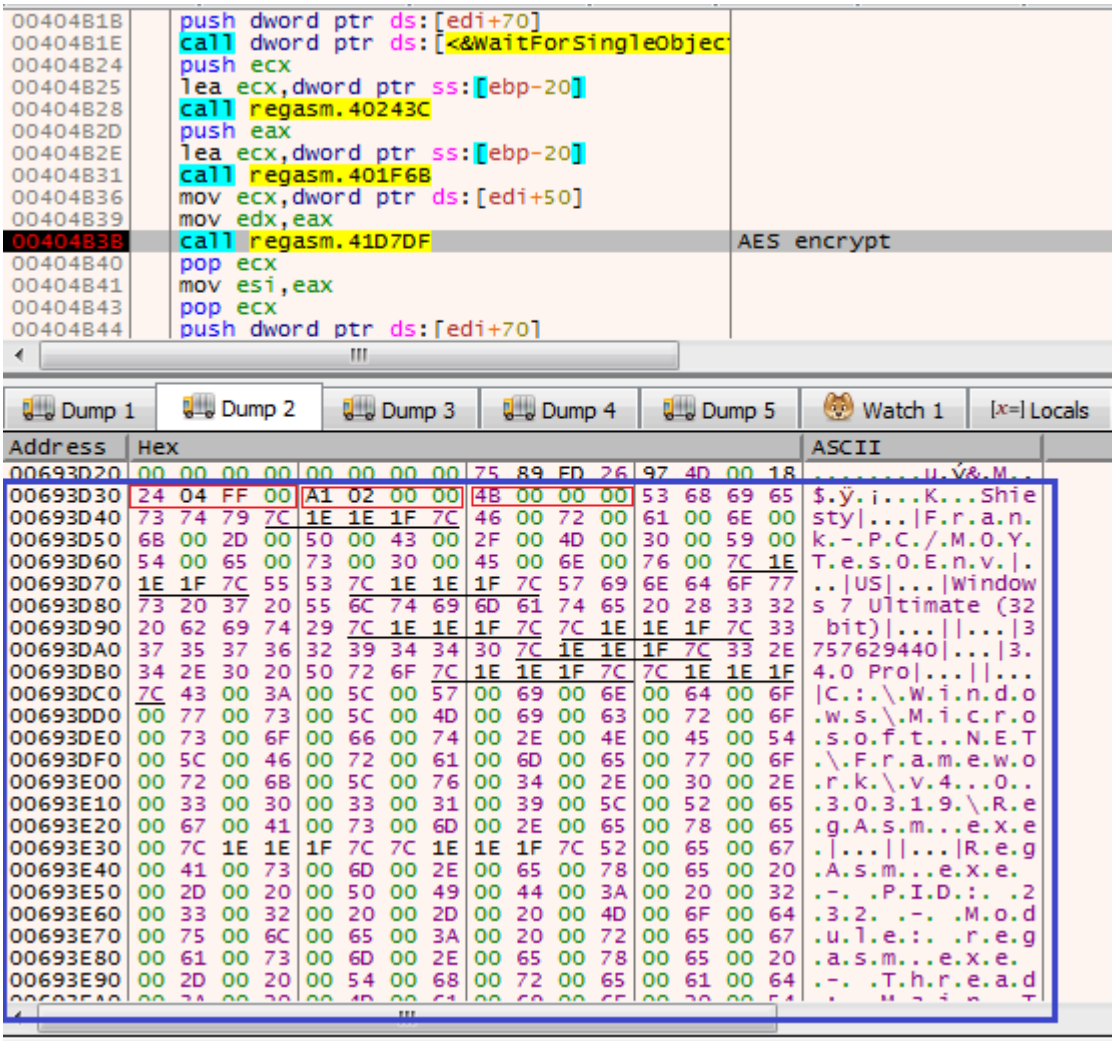

Figure 11: 4BH packet plaintext content before AES encryption

The victim's basic information is enclosed in this packet. Let's take a look at the packet structure.

The first "24 04 FF 00" is the packet magic ID that comes from the decrypted configuration block, the subsequent dword "A1 02 00 00" (21AH) is the size of following data, the next dword "4B 00 00 00" (4BH) is the packet number. The entire rest data are the collected basic information of the victim's device, which includes but not limited to:

- Remcos assigned name "Shiesty" (from configuration block)
- Victim's user name and computer name
- Windows edition information, total RAM (3757629400) in bytes

- Remcos version (3.4.0 Pro)
- The full path of current RegAsm.exe, the title of the currently active program (the victim's using)
- Victim's idle time
- The system's uptime
- CPU information
- C2 server host
- Remcos payload type (EXE or DLL)

All above value fields are split by a separator - "7C 1E 1E 1F 7C" (shown as "|…|" in string).

As long as the C2 server receives this 4BH packet, it shows the victim in the "Connection" subtab, as shown in Figure 12. Since then the attacker can control the victim's device by just right clicking on the item (red box) and selecting the commands they wanted.

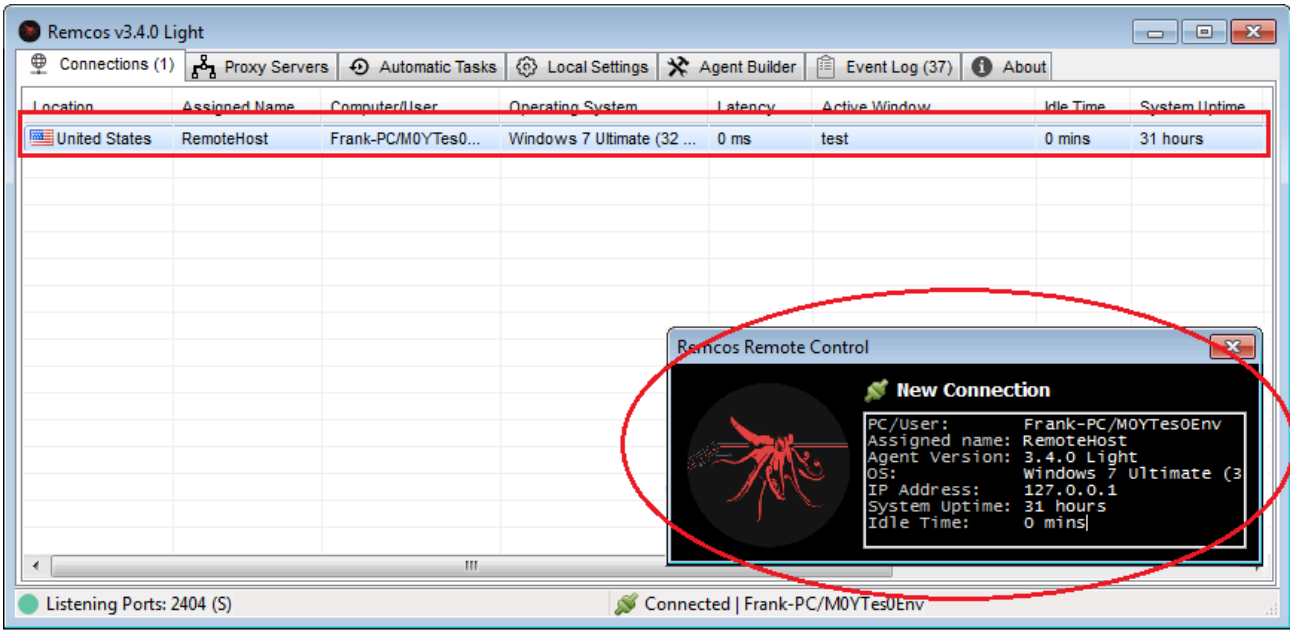

Figure 12: How C2 server looks when receiving a 4BH packet

Meanwhile, Remcos registers a callback function that parses the C2's commands and goes to an infinite loop to wait for the upcoming control commands from the attacker's C2 server.

Control Commands

From the registered callback function, we learned that this Remcos variant provides 87 control commands, which have been categorized in below groups:

- **System**: Screen Capture, File Manager, File Search, Process Manager, etc.
- **Surveillance**: Webcam, Microphone, Keylogger, Screenlogger, etc.
- **Network**: Proxy, Downloader, Open Webpage, etc.
- **Extra**: Dll Loader, Logins Cleaner, Audio Player, etc.
- **Remcos**: Reconnect, Restart, Show, Update, Close, Uninstall, etc.
- **Heartbeat packet**

The C2 server sends a heartbeat packet to Remcos every 40 seconds. Once Remcos has connected to the C2 server, the heartbeat makes sure this Remcos is alive. The C2's command packet has same format. I'll take the heartbeat packet as an instance to explain, which looks like:

24 04 FF 00 0C 00 00 00 01 00 00 00 30 7C 1E 1E 1F 7C 32 30

After the packet magic ID ("24 04 FF 00") and packet size (0x0C), "01 00 00 00" is heartbeat command number (0x01), and rest is command data being split by "7C 1E 1E 1F 7C" that are 30 (ASCII "0") and 32 30 (ASCII "20"). Remcos then obtains the title of currently active window as well as a time value and sends them to the C2 server in packet number 4CH.

The following is a control command list:

Name	**C&C Number**	**Description**
HeartBeat	01H	HeartBeat packet.
Screen Capture	10H	Control the victim's device in a remote desktop.
File Manager	98H	Manager file system on victim's device.
File Search	8FH	Search file on victim's device.
Process Manager	06H	Manager running process.
Service Manager	34H	Manager victim's system service.
Registry Editor	2FH	View, Edit victim's system registry.
Installed Program	03H	List all installed software.
Windows Manager	08H	Open a Task Manager similar interface.
Clipboard Manager	28H	View, Set, and Empty victim's system clipboard.

Command Line	0EH	Start a shell (cmd.exe) with command.
Execute Command	0DH	Execute a command in the victim’s device, like Notepad.
Remote Scripting	2EH	Execute JS, VBS, and Batch on the victim’s device.
Set Wallpaper	92H	Set the victim’s desktop wallpaper with a picture.
Power Manager	27H	Log off, Sleep, Hibernate, Shut down, and Restart.
Webcam	1BH	Control victim’s camera to work
Microphone	1DH	Turn on the victim’s audio input device, like Microphone.
Keylogger	13H	Start Keylogger.
Screenlogger	10H	Start Screenlogger.
Browser history	18H	Clear browser’s history.
Proxy	32H	Set proxy to victim’s device.
Open Webpage	0FH	Open a URL with the victim’s default browser.
Chat	30H	Pop up a chatting box to chat with the victim.
MessageBox	26H	Pop up a message to the victim.
Downloader	B2H	Download and execute a file on the victim’s device.

Dll Loader	2CH	Execute a Dll module on the victim’s device.
Audio Player	A3H	Play an audio sound to the victim.
Logins Cleaner	18H	Clear browser’s logins and cookies.
Update	24H	Update Remcos.
Uninstall	22H	Uninstall Remcos from the victim’s device.
Close	21H	Kill currently running Remcos.
Restart	23H	Restart Remcos.
Elevate	27H	Elevate Remcos’ privileges.

Besides the listed control commands, Remcos also has many sub-commands to support some of the control commands in sub-connections, like Service Manager command 34H with sub-commands: 03H to stop a service, 04H to pause a service, 01H to restart a service.

Conclusion

In this analysis blog, I explained how a phishing email delivers an Excel document with malicious Macro into the victim’s device.

Next, we went through how it executes multiple VBS and Powershell scripts to download the Remcos payload as well as how the Remcos payload is deployed by a .Net Dll into the “RegAsm.exe” process via Process Hollowing.

Then, I dissected Remcos’s workflow according to its code and how a configuration block is decrypted from the PE resource section. I also explained how Remcos established connection to its C2 server.

Finally, through several examples, I elaborated the structure of the control and command packets in plaintext as well as what commands Remcos is able to use to control the victim’s device and the control command list.

Fortinet Protections

Fortinet customers are already protected from this malware by FortiGuard's Web Filtering, Antivirus, FortiMail, FortiClient, FortiEDR services, IPS services, and CDR (Content Disarm and Reconstruction) services, as follows:

All relevant URLs have been rated as "**Malicious Websites**" by the FortiGuard Web Filtering service.

The captured Excel sample and the downloaded Remcos payload files are detected as "**VBA/Remcos.REM!tr** " and "**W32/Rescoms.M!tr**" and are blocked by the FortiGuard Antivirus service.

FortiEDR detects both the Excel file and Remcos payload file as malicious based on its behavior.

Fortinet also released IPS signature "Remcos.Botnet" to detect and block Remcos' C&C traffic to protect our customers.

FortiGuard Content, Disarm, and Reconstruction (CDR) can protect users from this attack by enabling the following option:

- Enable/disable stripping of linked objects in Microsoft Office documents.

In addition to these protections, Fortinet has multiple solutions designed to help train users to understand and detect phishing threats:

The FortiPhish Phishing Simulation Service uses real-world simulations to help organizations test user awareness and vigilance to phishing threats and to train and reinforce proper practices when users encounter targeted phishing attacks.

In addition to these protections, we suggest that organizations also have their end users go through our FREE NSE training: NSE 1 – Information Security Awareness. It includes a module on Internet threats that is designed to help end users learn how to identify and protect themselves from various types of phishing attacks.

IOCs

URLs:

hxxp://209[.]127[.]19[.]101/flip.vbs

hxxp://209[.]127[.]19[.]101/mem.txt

hxxp://209[.]127[.]19[.]101/faze.jpg

shiestynerd[.]dvrlists[.]com:10174

mimi44[.]ddns[.]net:2405

harveyautos110[.]ddns[.]net:2404

harveyautos111[.]hopto[.]org:2404

harveyautos112[.]ddns[.]net:2404

harvey205[.]camdvr[.]org:2404

harvey206[.]casacam[.]net:2404

harvey207[.]accesscam[.]org:2404

23[.]226[.]128[.]197:2404

achimumuazi[.]hopto[.]org:2311

xhangzhi[.]duckdns[.]org:2404

Sample SHA-256 Involved in the Campaign:

[Excel Document]

FBB0575DFD7C1CFE48FB3AA895FBE6C8A554F06899A7152D04CFC39D1D4744AD

[Captured Remcos samples]

8F6DD0DB9E799393A61D6C9CF6495C164E1B13CB8E6B153B32359D5F07E793D2
DA609D3211D60D5B11FEAEAA717834CBE86E18103A1ED4FC09C2EE3E1CFF9442
737E11913EFB64ACCF1B88532C7CE8606676684D8364DDD027926F9FFC6ECFFB
B263876EBC01B310A8BFC58477523981184EB7E8F2DC955F0CF8E62124EB679A
2C8B78FC6C4FE463DAC9D39FDE2871F1BB2605453BC0F2D57C7549CF5D07AA86
A1A1395D0602A473FCC81BA7D1D90C3FB154321D1721E0069722B902B1057CB0
6B816D84ACCC3E1EBCE3EF55B64B0C5E0485228790DF903E68466690E58B5009

Learn more about Fortinet's FortiGuard Labs threat research and intelligence organization and the FortiGuard Security Subscriptions and Services portfolio.
