New Campaign Uses Remcos RAT to Exploit Victims | FortiGuard Labs

By Xiaopeng Zhang

Published: 2024-11-08 · Archived: 2026-04-02 11:15:01 UTC

Affected platforms: Microsoft Windows
Impacted parties: Windows Users
Impact: Fully remotely control a victim's computer
Severity level: High

Fortinet's FortiGuard Labs recently noticed a phishing campaign in the wild. It is initialized with a phishing email containing a malicious Excel document. Upon researching the campaign, I found it was spreading a new variant of the Remcos RAT.

Overview

Remcos is a commercial RAT (remote administration tool) sold online. It provides purchases with a wide range of advanced features to remotely control computers belonging to the buyer. However, threat actors have abused Remcos to collect sensitive information from victims and remotely control their computers to perform further malicious acts.

Figure 1 displays the Remcos webpage.

Figure 1: Remcos RAT software is sold online

In this security blog, I will show how Remcos is delivered to a victim's device, what kinds of anti-analysis techniques it leverages to protect itself from being analyzed, how this variant of Remcos is deployed, how it achieves persistence on the victim's device, and what advanced features Remcos provides to remotely control a victim's device.

The Phishing Email

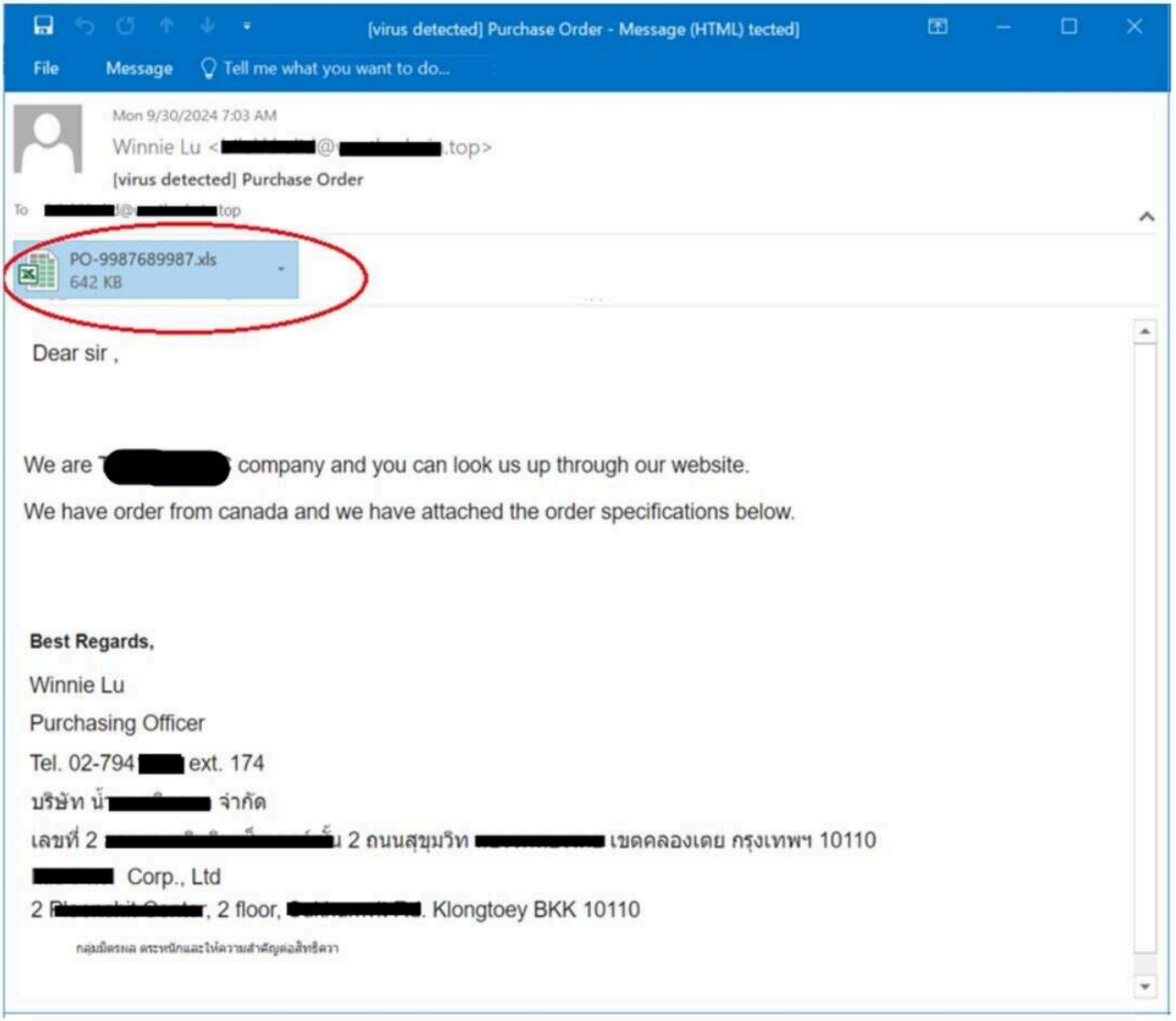

Figure 2: The phishing email

The phishing email is shown in Figure 2. It contains an attached malicious Excel file disguised as an order file to convince the recipient to open the document.

CVE-2017-0199 Exploited by the Excel Document

CVE-2017-0199 is a Remote Code Execution vulnerability that exploits how Microsoft Office and WordPad parse specially crafted files. Once the recipient opens the attached file, the MS Excel program shows the file content, as seen in Figure 3.

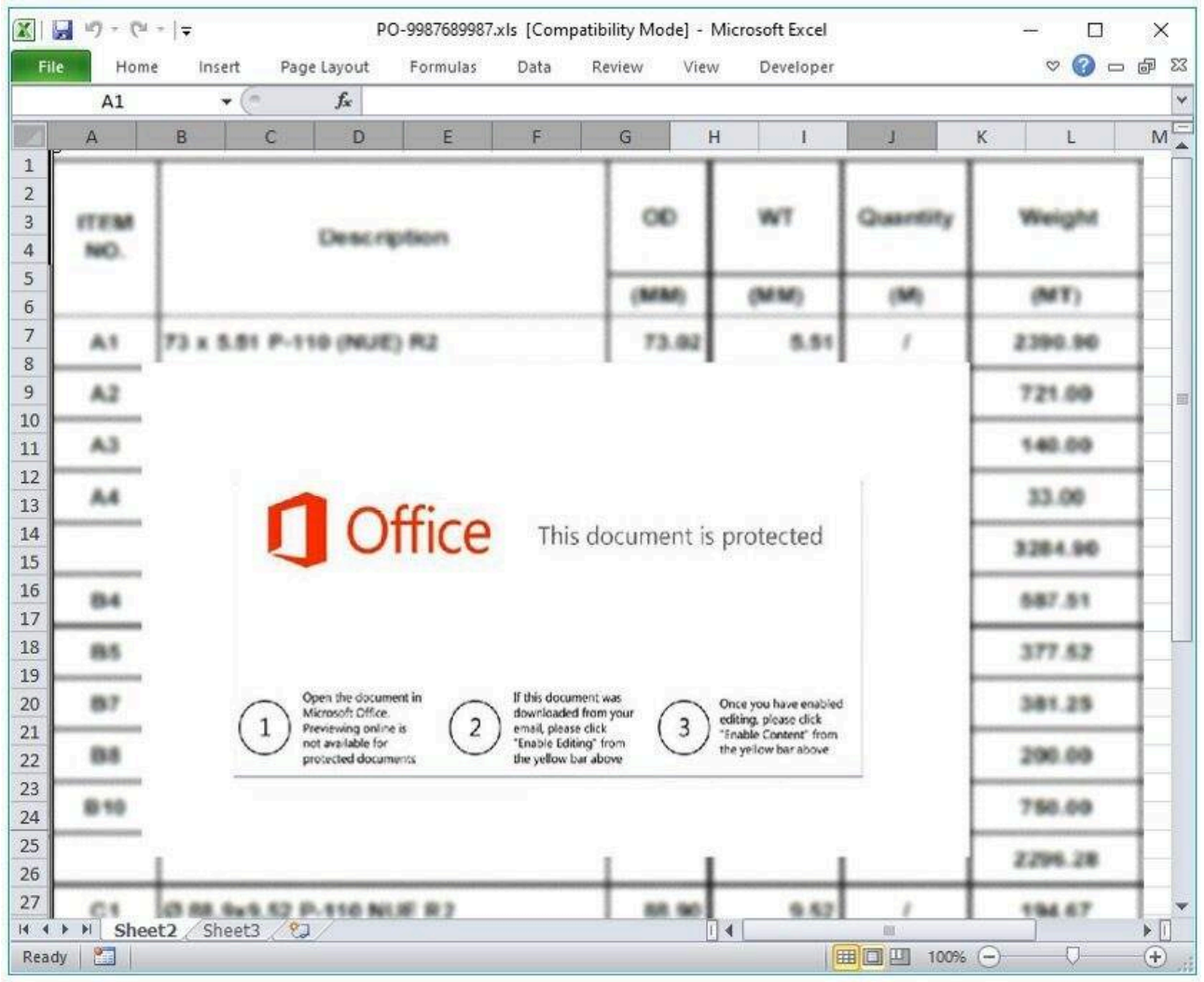

Figure 3: The file opened in Excel

In the background, the CVE-2017-0199 vulnerability is exploited to download an HTA file and execute it on the recipient's device.

As you may know, a crafted embedded OLE object leads to this vulnerability. Figure 4 demonstrates the content of the crafted embedded OLE object ("\x01Ole").

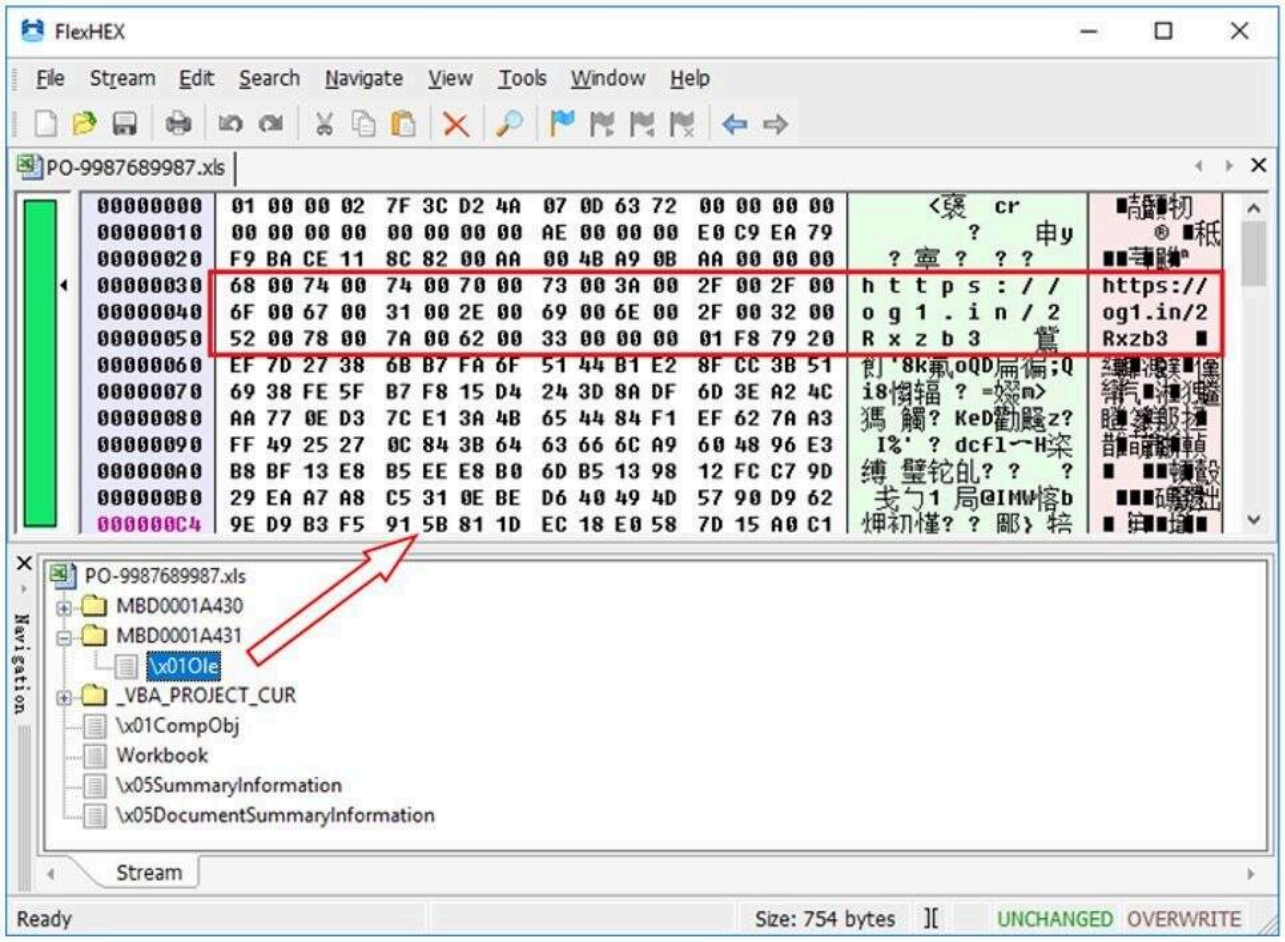

Figure 4: The crafted OLE object with a URL

MS Excel program accesses the short URL “hxxps://og1[.]in/2Rxzb3.” It is then redirected to another URL, “hxxp://192[.]3[.]220[.]22/xampp/en/cookienetbookinetcahce.hta”. The download packet, shown in Figure 5, is a Wireshark screenshot.

Figure 5: The downloaded HTA file

The HTA file is an HTML Application file. It is executed by a Windows-native application (mshta.exe) called by MS Excel using DCOM components.

Multiple Script Languages

Its code is wrapped in multiple layers using different script languages and encoding methods, including JavaScript, VBScript, Base64-encoded, URL-encoded, and PowerShell, to protect itself from detection and analysis.

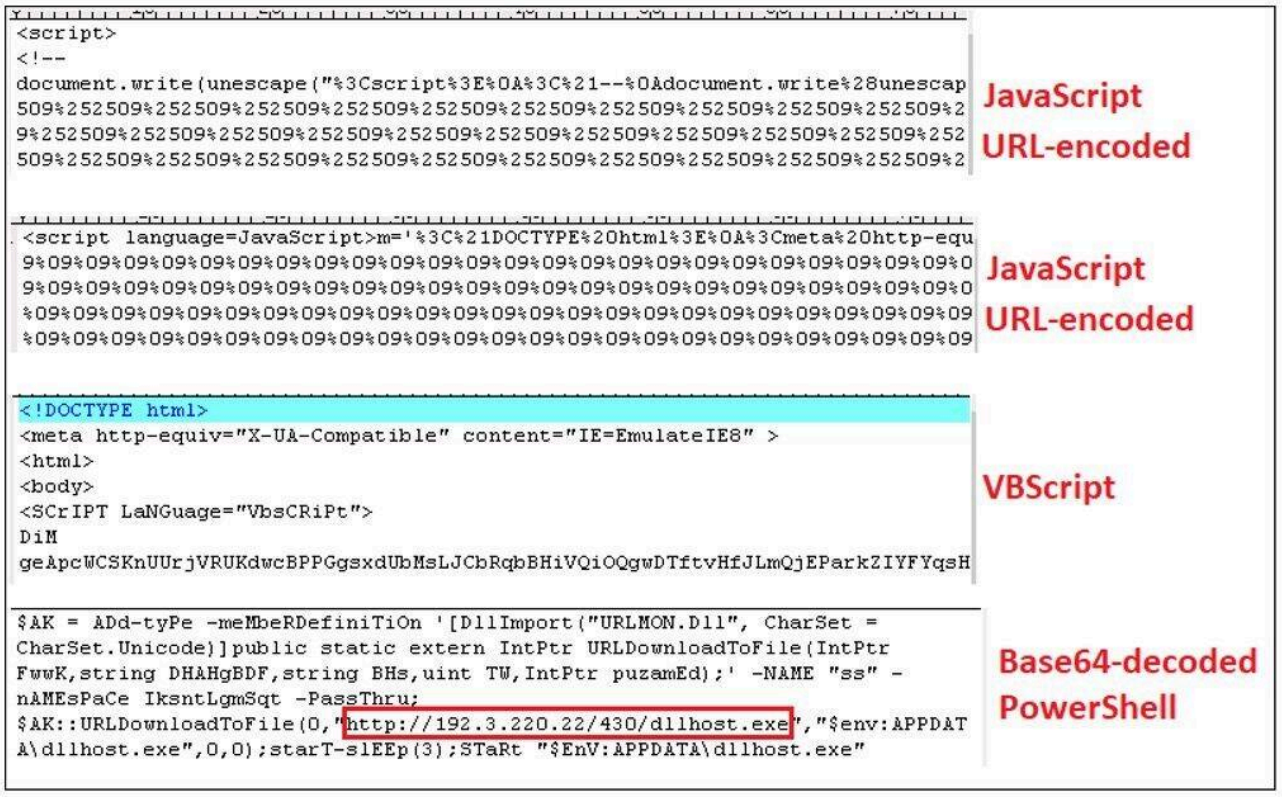

Figure 6: Examples of multiple script code

Figure 6 shows some script code examples. These are executed when the downloaded HTA file is parsed in mshta.exe.

Take a look at the Powershell code at the bottom of Figure 6. It calls the API URLDownloadToFile() to download an EXE file from the URL “hxxp://192[.]3[.]220[.]22/430/dllhost.exe” into a local file, “%AppData%\dllhost.exe.”

Executing “STaRt $EnV: APPDATA\dllhost.exe” starts the downloaded EXE file on the victim’s device.

Starting the Downloaded EXE

Once the downloaded EXE file, dllhost.exe, starts, it extracts a batch of files into the %AppData% folder. Figure 7 is a screenshot of the extracted files and sub-folders located in %AppData%\intercessionate\Favourablies117\sulfonylurea. Some of the key data are hidden in these files.

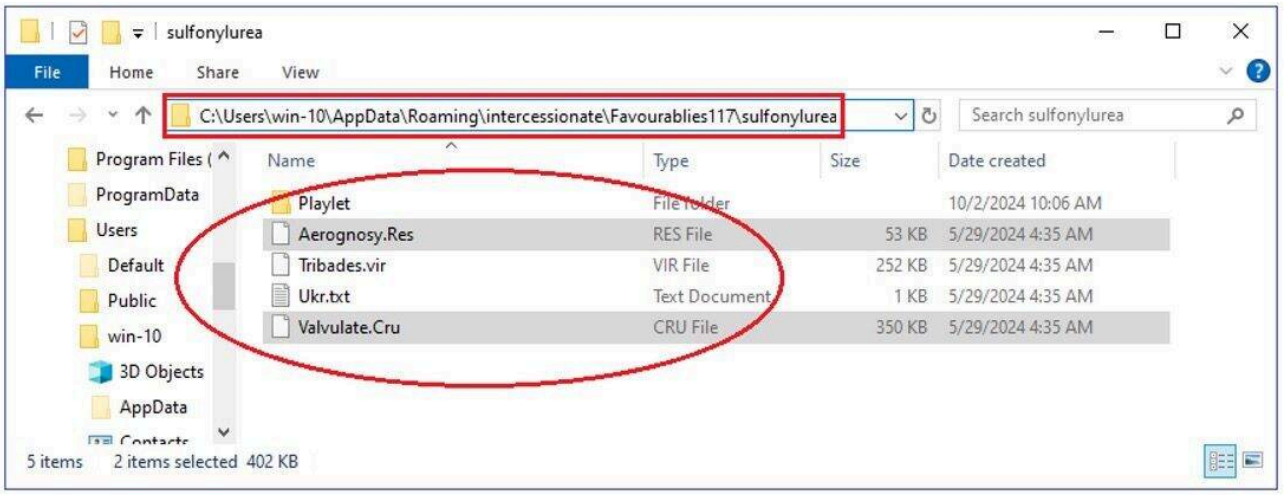

Figure 7: Screenshot of partially extracted files

dllhost.exe then runs the PowerShell program by calling the API CreateProcessW() to execute a piece of PowerShell code, as illustrated in Figure 8.

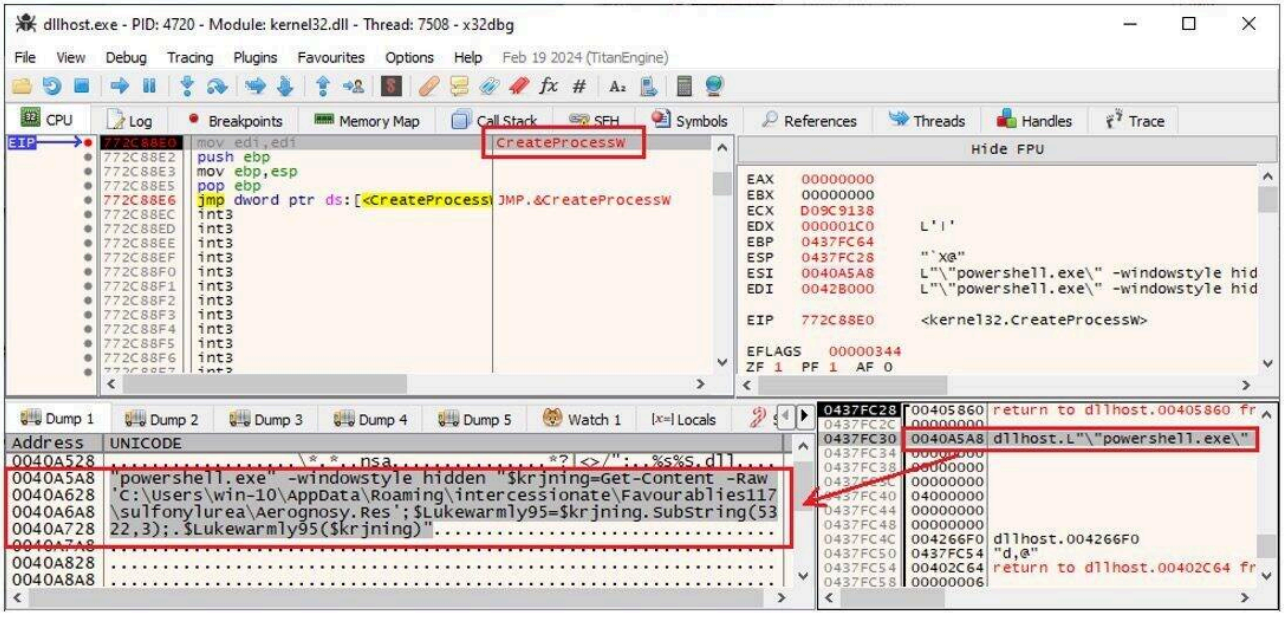

Figure 8: dllhost.exe about to run the PowerShell program

Since dllhost.exe is a 32-bit process, it runs a 32-bit PowerShell, “C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe.” This is important because the malicious code only works on the 32-bit PowerShell process.

The PowerShell code breaks down as follows.

$krjning=Get-Content -Raw '%AppData% \intercessionate\Favourablies117\sulfonylurea\Aerognosy.Res';
$Lukewarmly95=$krjning.SubString(5322,3);
.$Lukewarmly95($krjning)

It reads the content of “Aerognosy.Res,” an extracted file, into a local variable “$krjning,” which is full of PowerShell script. “iEx” is the return value of SubString(5322,3), which is short for an Invoke-Expression used to run a string as a command or expression. Finally, “iEx” executes the entire PowerShell script (“Aerognosy.Res”) after calling “.$Lukewarmly95($krjning).”

Again, the PowerShell code is thoroughly obfuscated and encoded. Based on my research on the code, I discovered it performs the following functions:

- Copies dllhost.exe to %temp% and renames it “Vaccinerende.exe.”
- It hides the current PowerShell process in the background.
- Loads malicious code from the extracted “Valvulate.Cru” file.
- Deploys the malicious code in memory by calling the APIs VirtualAlloc() and MemoryCopy().
- Executes the malicious code’s entry point by calling the API CallWindowProcA(), as listed in Figure 9.

Figure 9: Debugging the de-obfuscated PowerShell code from Aerognosy.Res

Malicious Code Runs Inside the PowerShell Process

Self-decrypting the malicious code:

As I mentioned, the copied malicious code relies on a 32-bit version of PowerShell. It first runs a piece of self-decryption code mixed with a huge amount of useless instructions, creating a big challenge for analysts. Figure 10 presents the end of the decrypting code, where it is about to execute the “call edi” instruction. The EDI register now points to the decrypted code, as shown on the right.

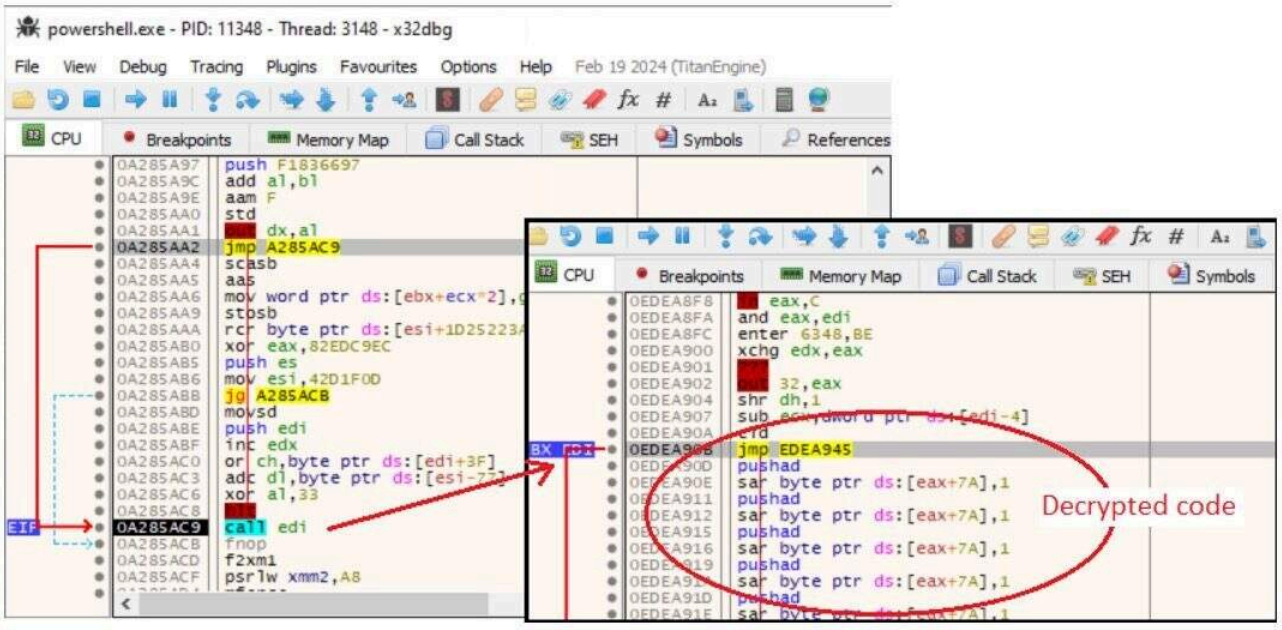

Figure 10: About to call the decrypted code

It leverages numerous complicated anti-analysis techniques to protect itself from being analyzed.

Anti-Analysis Techniques:

1. It installs a vectored exception handler.

Whenever an exception occurs, the exception handler is called to handle it. The exception code provides corresponding ways to restore the code to resume from another offset. There are numerous exception instructions inside the malicious code. In other words, this strategy drives the entire code.

Figure 11 shows an instruction that can raise an exception (0x8000003) at 0xEE16222, which the exception handler will then capture and handle.

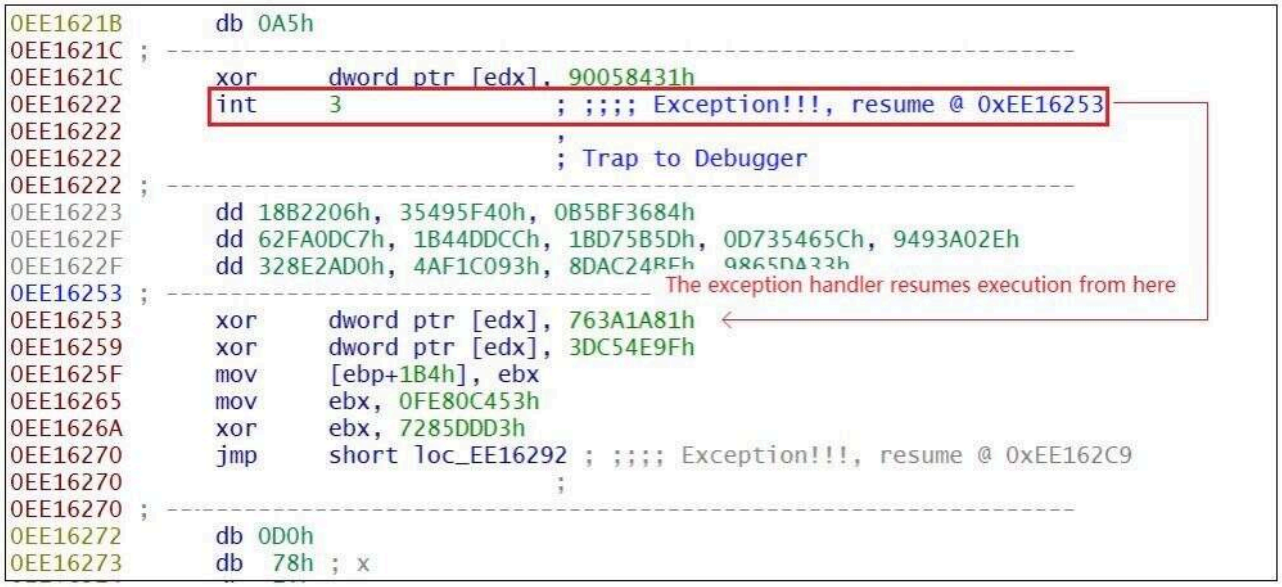

Figure 11: Example of an exception

The exception handler function also checks the DR registers (Debug Registers), which are Dr0, Dr1, Dr2, Dr3, Dr4, Dr5, Dr6, and Dr7. Their values are not 0 when a debugger is present.

2. System APIs are dynamically gained and called in a unique way.

There are no API name constant strings in the code. Instead, it keeps the hash codes of the API names. Whenever it calls an API, it uses a unique function that retrieves the API information from the PEB, which is pointed to by fs:[30h] to get their function address by matching the name's hash code. This raises the difficulty of performing static analysis.

In addition, it has another function that is called every time an API is called. This function detects if the debugger has set the API breakpoint.

It also encrypts the code from the caller's return address to the base address, which cleans up the code just executed.

3. It is called ZwSetInformationThread(), and it performs anti-debugging.

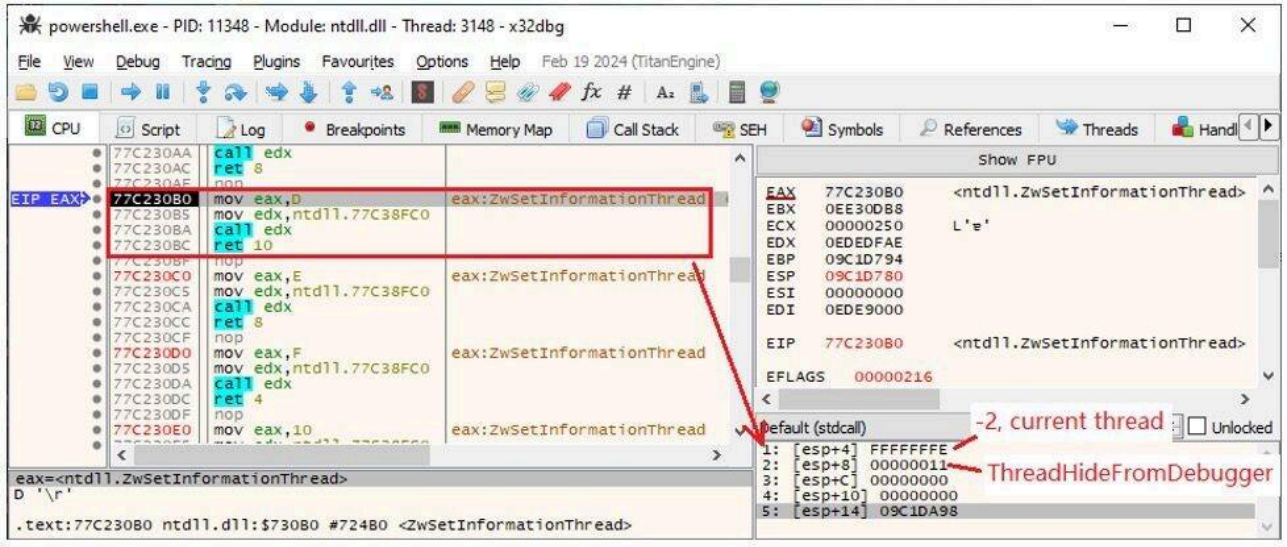

Figure 12: It breaks on the API ZwSetInformationThread()

The malicious code calls API ZwSetInformationThread() with the argument ThreadHideFromDebugger (0x11) and the current thread (0xFFFFFFFE). This mechanism in Windows can conceal a thread's existence from debuggers. Figure 12 illustrates how it calls this API with the associated arguments.

If a debugger is attached to the current process, it exits immediately once the API is called.

4. It checks the result value of API ZwQueryInformationProcess().

It calls API ZwQueryInformationProcess() with the ProcessDebugPort (7) argument to detect if the debugPort is set (non-zero value). If yes, this means a debugger is attached to the current process (PowerShell.exe).

5. All the constant values are gained at run time.

Please refer to the code snippet below to see how it splits “mov ebx 100h” into three instructions.

```
[...]
0EDEE0F4        pop    edx
0EDEE0F5        push   ebx
0EDEE0F6        mov    ebx, 5056107Ch
0EDEE0FB        xor    ebx, 902EFE8Fh
0EDEE101        add    ebx, 3F87120Dh  ; It implements “mov ebx,  100h”
0EDEE107        push   esi
0EDEE108        pushf
0EDEE109        mov    esi, esp
 [...]
```

6. It uses an API hooking technique for several APIs.

The malicious code simulates executing multiple API instructions (say, two instructions) at the beginning and then jumps to the API to execute the rest of the instructions (beginning with the 3rd instruction).

Below is an example for CreateProcessInternalW(). The highlighted codes are simulated, which can invalidate any API breakpoints.

```
[...]
0EE38137   push   690AF764h
0EE3813C   test   eax, ecx
0EE3813E   add    dword ptr [esp], 0D62E941Dh
0EE38145   xor    dword ptr [esp], 0B67F037Eh
0EE3814C   sub    dword ptr [esp], 894678FFh   ; The instructions equal to push
1000h
0EE38153   cmp    al, cl
0EE38155   test   ch, 0E2h
0EE38158   jmp    ecx: The ecx points to the rest of the API instructions
[...]
```

Whenever any of the above detection conditions are triggered, the current process (PowerShell.exe) can become unresponsive, crash, or exit unexpectedly.

Process Hollowing (Malicious Code into Vaccinerende.exe):

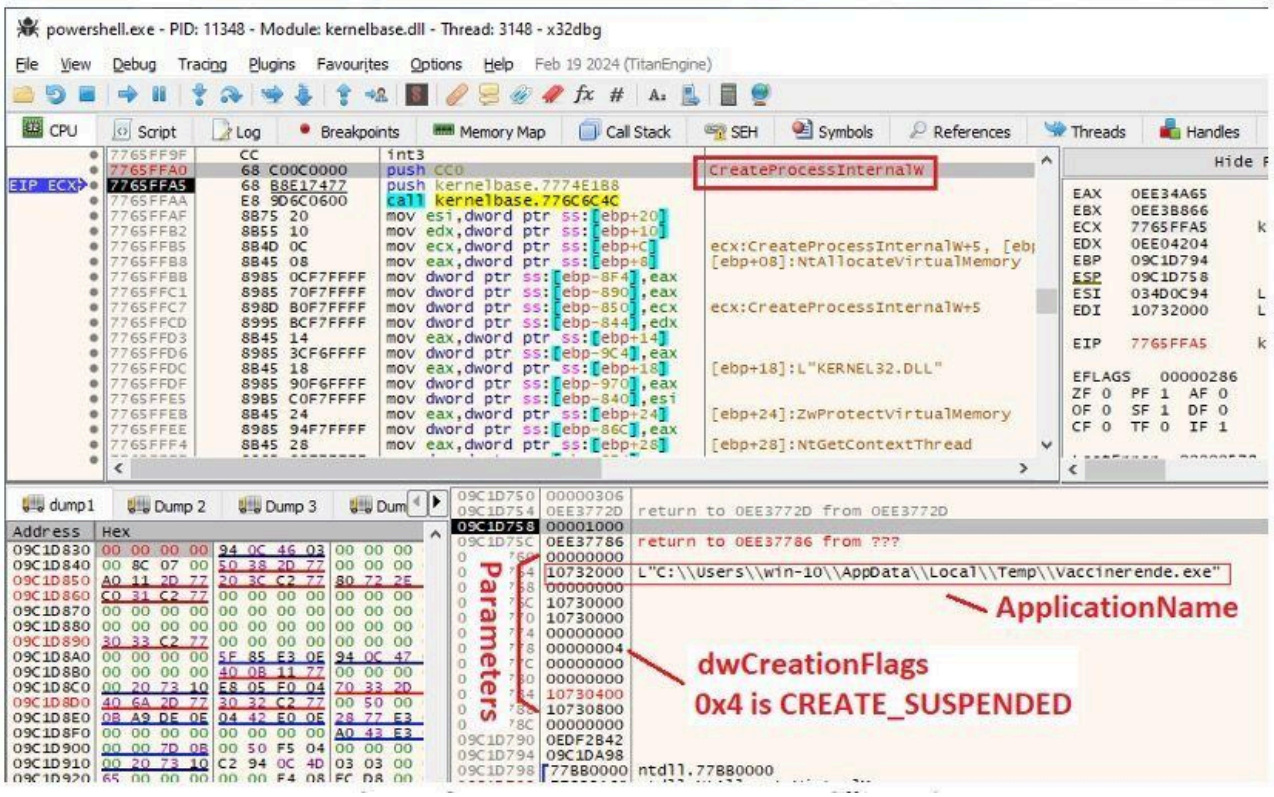

Figure 13: Display of CreateProcessInternal() and its parameters

The malicious code performs process hollowing to put itself into a newly created Vaccinerende.exe process (copied from dllhost.exe). To do this, it calls the API CreateProcessInternalW() with CreatFlags of CREATE_SUSPENDED (0x4), which will suspend the new process after it is created. It then calls some related APIs to transfer all the malicious code to the new process and run it.

The relevant APIs are NtAllocateVirtualMemory(), ZwCreateSection(), NtMapViewOfSection(), NtGetContextThread(), NtSetContextThread(), and NtResumeThread().

Malicious Code Runs inside Vaccinerende.exe

It performs all the anti-analysis detections described in the above section and then uses a workflow different from the PowerShell process.

According to my research, it finishes some tasks, like maintaining persistence, downloading and decrypting the Remcos payload execution, and starting the downloaded Remcos in memory.

Maintaining Persistence:

The malicious code adds a new auto-run item to the system registry to maintain persistence and maintain control of the victim's device when it is restarted.

Figure 14 shows the malicious code as it is about to run the REG (reg.exe) process to add the auto-run item and how it appears in the Registry Editor. It calls the API ShellExecuteW() to run cmd.exe with a command line string of /c REG ADD HKCU\Software\Microsoft\Windows\CurrentVersion\Run /f /v "Chivey57" /t

REG_EXPAND_SZ /d "%Misbehavers% -windowstyle 1 $Frligheden=(gp -Path 'HKCU:\Software\Roscoelite\').Aftvttedes;%Misbehavers% ($Frligheden)."

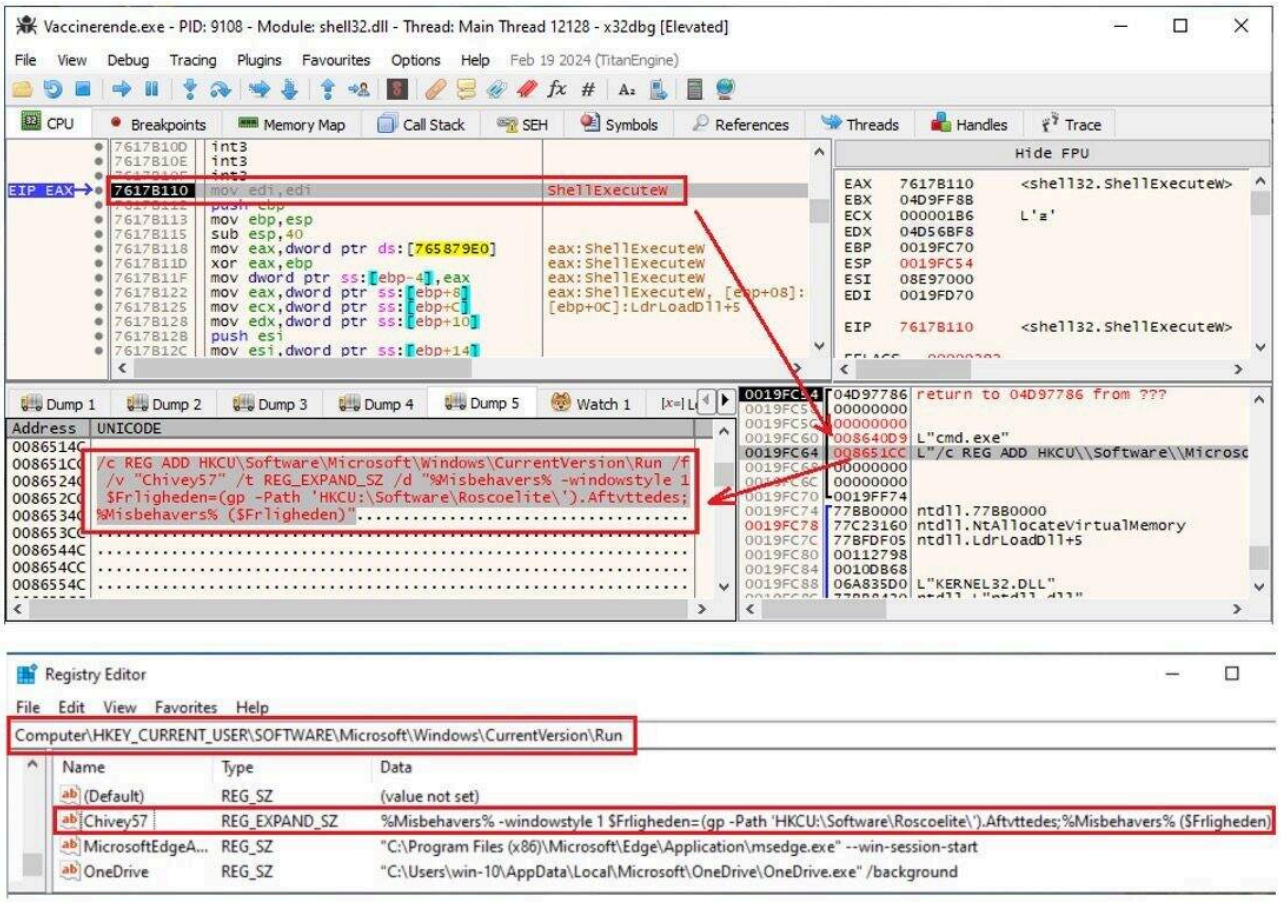

Figure 14: Adding an auto-run item in the system registry

“%Misbehavers%” has been defined as the full path to the 32-bit PowerShell.exe in the system environment. It reads a piece of PowerShell code, which is the same as the PowerShell code that dllhost.exe executes, from a string value called “Aftvttedes” in the system registry.

Downloading and running Remcos:

Next, the malicious code downloads an encrypted file from the URL “hxxp[:]//192[.]3[.]220[.]22/hFXELFSwRHRwqbE214.bin.” The file contains the encrypted Remcos malware. To download the file, some relevant APIs, like InternetOpenA(), InternetOpenUrlA(), and InternetReadFile(), are called in a row.

After decrypting the downloaded file, I found a new variant of Remcos. Rather than saving the Remcos file into a local file and running it, it directly deploys Remcos in the current process's memory (Vaccinerende.exe). In other words, it is a fileless variant of Remcos.

It then starts Remcos on a thread, where the thread function (StartAddress) is the entry point. To start the thread, it calls an undocumented API, NtCreateThreadEx(). Figure 15 shows a screenshot of the debugger as it is about to call the API to start Remcos.

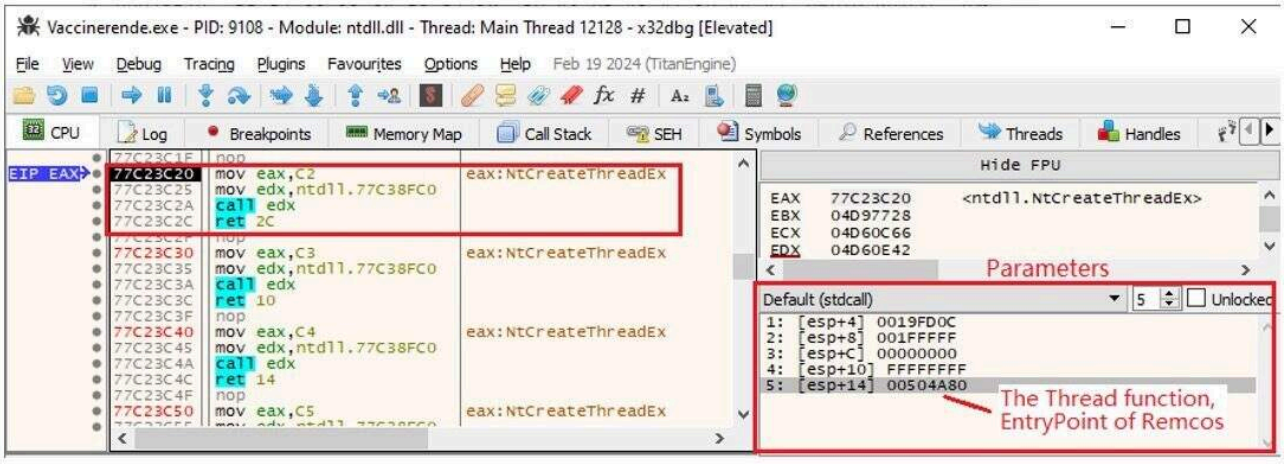

Figure 15: Starting the Remcos payload in a thread

Initializing Remcos

Each Remcos variant has a setting block with a batch of configurations that control how Remcos operates on the victim's device. The setting block is encrypted and saved in the Resource section named "SETTINGS," which gets decrypted at the start and initializes Remcos with the setting block.

Look at Figure 16 to examine the decrypted setting block in memory.

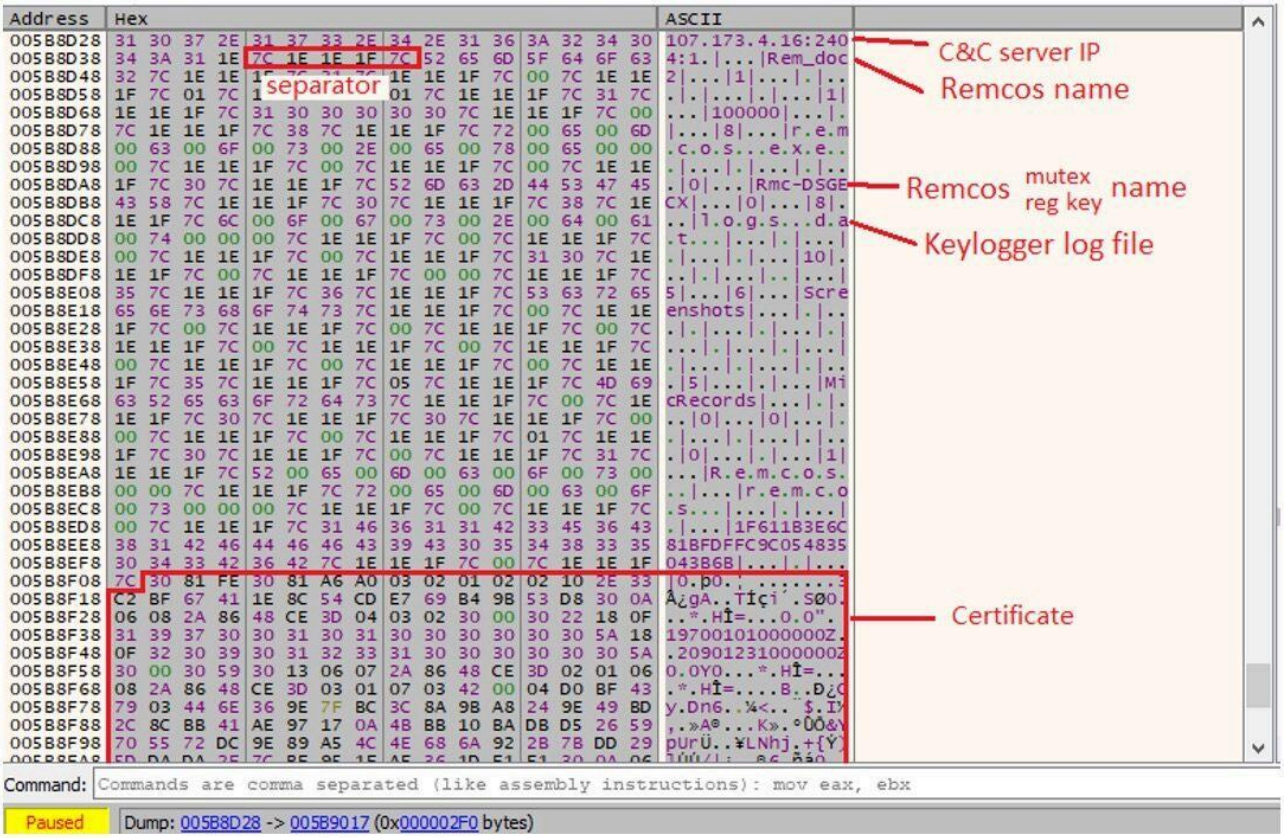

Figure 16: Memory view of the decrypted setting block

The setting block has 57 values in total, which are separated by "\x7C\x1E\x1E\x1F\x7C", as illustrated in Figure 16. The values in the setting block are retrieved by their index, from 0 to 56, through a special function.

The setting values tell Remcos how to do its work on the victim's device, including the C&C server IP address and port, Remcos' name, Remcos' mutex name (also registry key name), a Remcos license number, the keylogger's local log file, a couple of certificates used to verify and communicate with the C&C server, and several switch flags indicating if a feature is enabled or disabled, such as Keylogger, Screenshot, Watchdog, Record audios, Reset browsers' login, and more.

The C&C server IP&Port string at index 0 is "107[.]173[.]4[.]16:2404:1," where "107[.]173[.]4[.]16" is the IP address, "2404" is the TCP port, and the last "1" means that it enables TLS to communicate with the C&C server.

Remcos collects some basic information from the victim's device. It then encrypts and sends the collected data to its C&C server to register that the victim's device is online and ready to be controlled. This is the first command packet sent to the C&C server.

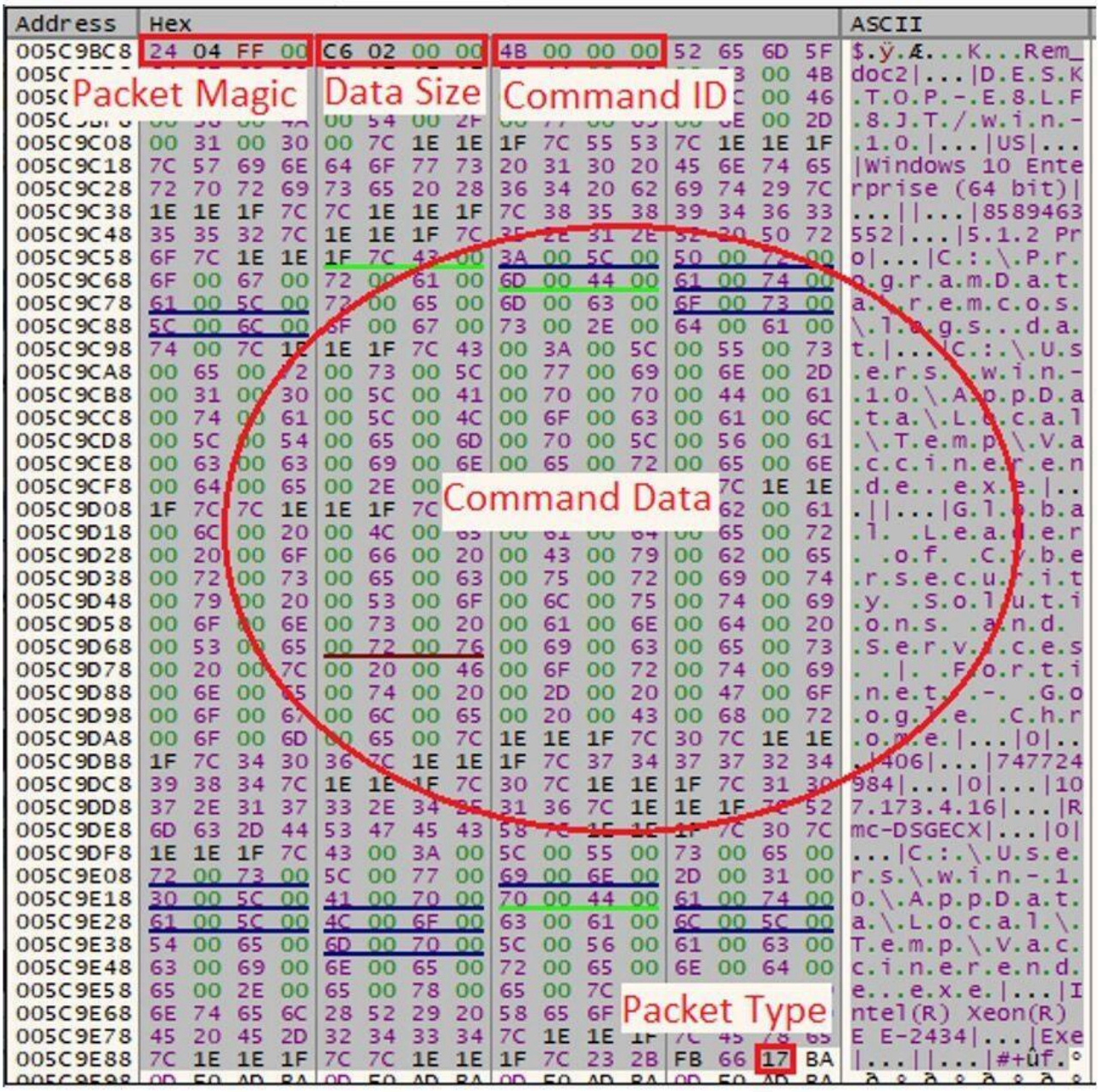

Figure 17: Example of the register packet with some basic information

The memory dump data in Figure 17 shows the content of the register packet (command ID 4Bh) that is about to be encrypted.

When TLS is enabled (as set in the settings block), packets follow the same structure, whether sending or receiving. These packets consist of **Packet Magic** (like 0xFF0424) + **Command Data Size** (like 0x2C6) + **Command ID** (like 0x4B) + **Command Data** + **Packet Type** (like 0x17, 0x16, and 0x15).

I will break down the command data of the 4B to explain what basic information Remcos collects from the victim's device. The command data has many separators ("\x7C\x1E\x1E\x1F\x7C") to separate the collected basic information.

- The processor's information.

- The memory status.
- The user's privilege level.
- The location of the victim's device.
- Remcos' file type ("EXE"), its full path on the victim's device, and the installed time.
- Remcos is assigned the name "Rmc-DSGECX," which is defined in the setting block.
- The IP address of the C&C server.
- The victim's device run time since its last start by calling the API GetTickCount().
- The idle time of the victim's device.
- The Remcos Keylogger local file path.
- The active program's title information on the device.
- The device name and user name of the affected device.
- The version of this variant of Remcos, which is the hardcoded string "5.1.2 Pro".
- The OS information, "Windows 10 Enterprise (64 bit)."

Features and Control Commands

After registering the victim's device on the C&C server, it receives a control command packet from the server to perform further work on the victim's device. These features and corresponding commands are detailed below.

This example of control command 06h asks Remcos to obtain all running process lists from the victim's device.

```
24 04 FF 00 | 04 00 00 00 | 06 00 00 00|17
```

Remcos sends a 4Fh command packet with the collected process list consisting of the process name, PID, architecture (64bit or 32bit), and the full path, as shown in Figure 18.

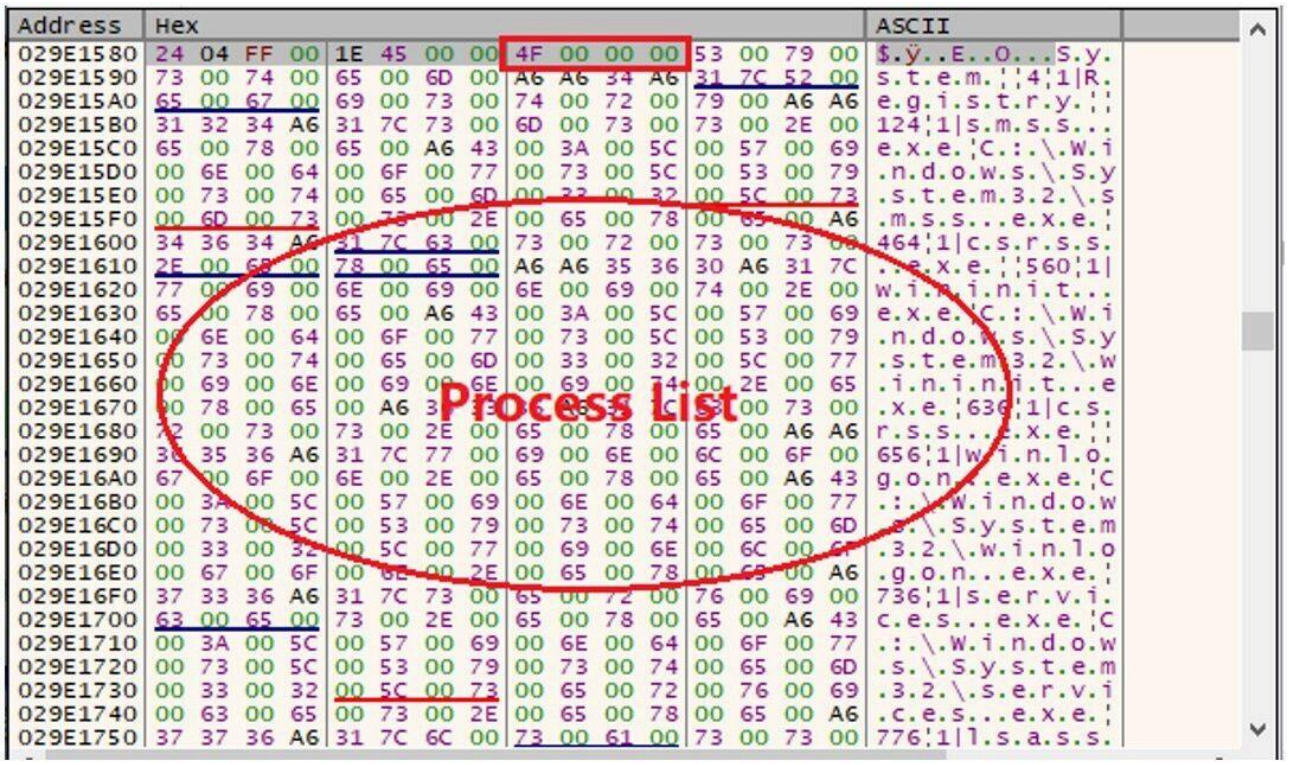

Figure 18: Send process list to C2 server

Figure 19 is a screenshot of the C&C server view of the process list.

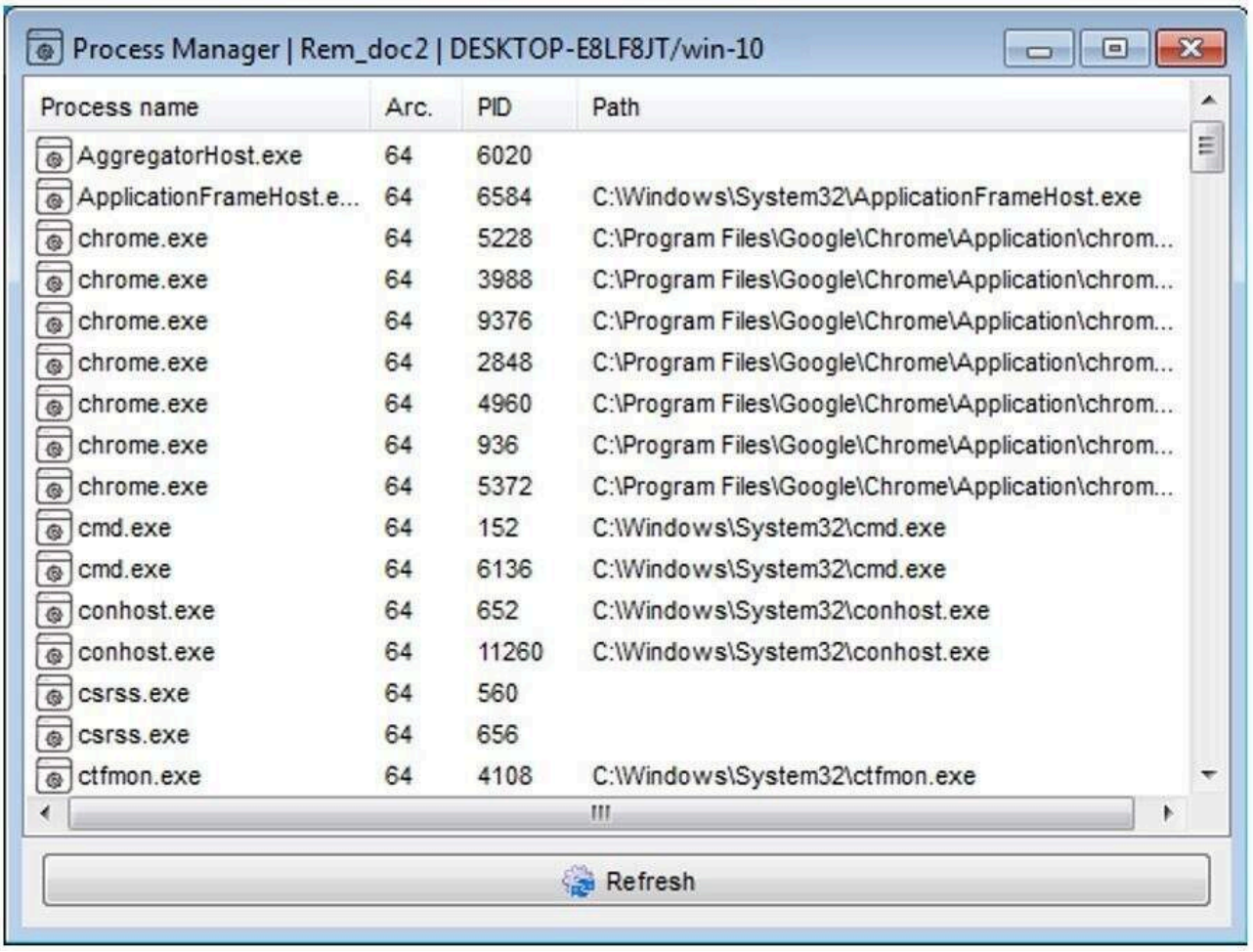

Figure 19: Process Manager on the C&C server

Remcos includes a function that parses the received control command data from the server and then performs the corresponding action on the victim's device.

My analysis of this function shows that Remcos has the features and commands listed in the following chart.

Features	C&C Number	Description
File Manager	98h, 88h	Browser all drivers (c:\,d:\...), and folders.
	8Fh	Search files by key words.
	8Ah, 8Bh,B5h	Transfer (Download / Upload) file or entire folder.
	89h	Open a file on victim's device.
	C2h, C3h	Zip or Unzip file or folder and download.
	8Eh	Create a folder.
	8Dh	Rename a specified file on the victim's device.
	C7h	Set specified file's attribute (hide, system, readonly...).
	8Ch	Delete a file or folder.
Process Manager	06h	Obtain all running processes from the victim's device.
	07h, C8h, C9h	Terminate, suspend and resume a process by PID.
Services	34h (with sub-cmds)	Manage the system services (Stop, Start, Pause ...).
Window Manager	08h	List all windows with handle on the victim's system.
	0Ah, 09h	Maximize / minimize a window by its handle.
	94h	Set a window's title by its handle.
	0Bh, ADh, AEh	Show / hide a window by its handle.
	0Ch	Terminate a process a window belongs to by its handle.
Registry Editor	2Fh, 35h	Obtain the registry data from the victim's device.
	39h, 37h	Add / delete a sub-key.
	38h, 36h	Add / delete a value. Rename a value, etc.
Programs Manager	03h	Retrieve installed programs from victim's device.
	0Dh	Remotely uninstall a program.
Shell	0Eh	Establish shell between the victim's device and C&C server.
Sys command	0Dh	Execute a remote command on the device.
Scripts	45h	Execute Javascript, VBS or batch remotely.
Clipboard	28h, 2Ah, 29h	Get and set data to and from the victim's clipboard.
Wallpaper	92h	Set victim's desktop wallpaper with a given picture.
Power Control	27h	Set Log off, Sleep, Hibernate, Shut down and Restart the remote device.
Camera	1Bh, 1Ch	Remotely turn the camera on or off.
Microphone	1Dh, 1Eh	Remotely turn the microphone on or off.
Keylogger	13h, 14h, 12h	Start / Stop the online keylogger.
Screenlogger	10h	Start the screenlogger.
Password	1Fh	Password recovery.
Network Monitor	CEh	Obtain a list of processes using the network.
Proxy	47h, 49h	Set or stop a proxy server on the victim's device.
Downloader	05h, 04h	Download a file from C&C server or URL and execute.
DNS redirection	CAh, CBh	Obtain or modify the content of the "hosts" file.
Open Webpage	0Fh	Open a webpage in the victim's default webbrowser.

Webpage		
Chat, Msgbox	30h, 3Bh, 26h	Chatting with the victim. Pop up victim a message box.
Play Sound	A5h, A6h,A2h, 61h	Play an alert sound or play a transferred sound file.
Dll Loader	2Ch, 2Bh	Download a fileless dll file and load it in Remcos and run.
Clean Logins	18h	Clean web browser's saved logins and cookies.
Extra Feature	43h	Enable or disable victim's input feature (keyboard, mouse). Hide or show the taskbar or start button. Turn the victim's monitor power off or on. Open or close CD driver.
Remcos Control	31h	Rename the Remcos on the victim's device.
	01h	Heartbeat.
	23h	Restart Remcos.
	ACh	Show the victim a Remcos logo window.
	27h	Elevate Remcos's privilege.
	25h, 24h	Update Remcos.
	21h	Terminate Remcos from the victim's device.

Summary

In this analysis, I walked through the entire process of the phishing campaign. It begins with a phishing email with an attached OLE Excel document. This disguised email is used to trick the recipient into opening the attached Excel document.

The CVE-2017-0199 vulnerability is exploited once the Excel file is opened on the victim's device. It then downloads an HTA file and executes it on the device. Multiple script languages are leveraged to download an EXE file (dllhost.exe), which then starts the 32-bit PowerShell process to load the malicious code from extracted files and execute it in the PowerShell process.

Next, I explained what anti-analysis techniques are used in the code, such as a vectored exception handler, dynamically gained APIs, dynamically calculated constant numbers, the APIs ZwSetInformationThread() and ZwQueryInformationProcess(), and API hooking.

After passing the detections introduced in the anti-analysis part, it performs process hollowing to run the malicious code in the new process "Vaccinerende.exe," which not only ensures persistence on the victim's device but also downloads and decrypts the Remcos payload file.

I then elaborated on how it keeps Remcos in memory and starts its entry point function in a thread (API NtCreateThreadEx()).

Subsequently, I explained how Remcos works with its setting block on the victim's device, how Remcos communicates with its C&C server, and what the format of the traffic packet looks like.

Finally, I focused on introducing the features this Remcos variant can perform on the victim's device and listing the relevant control commands for each feature.

Figure 20 shows the entire process of the phishing campaign.

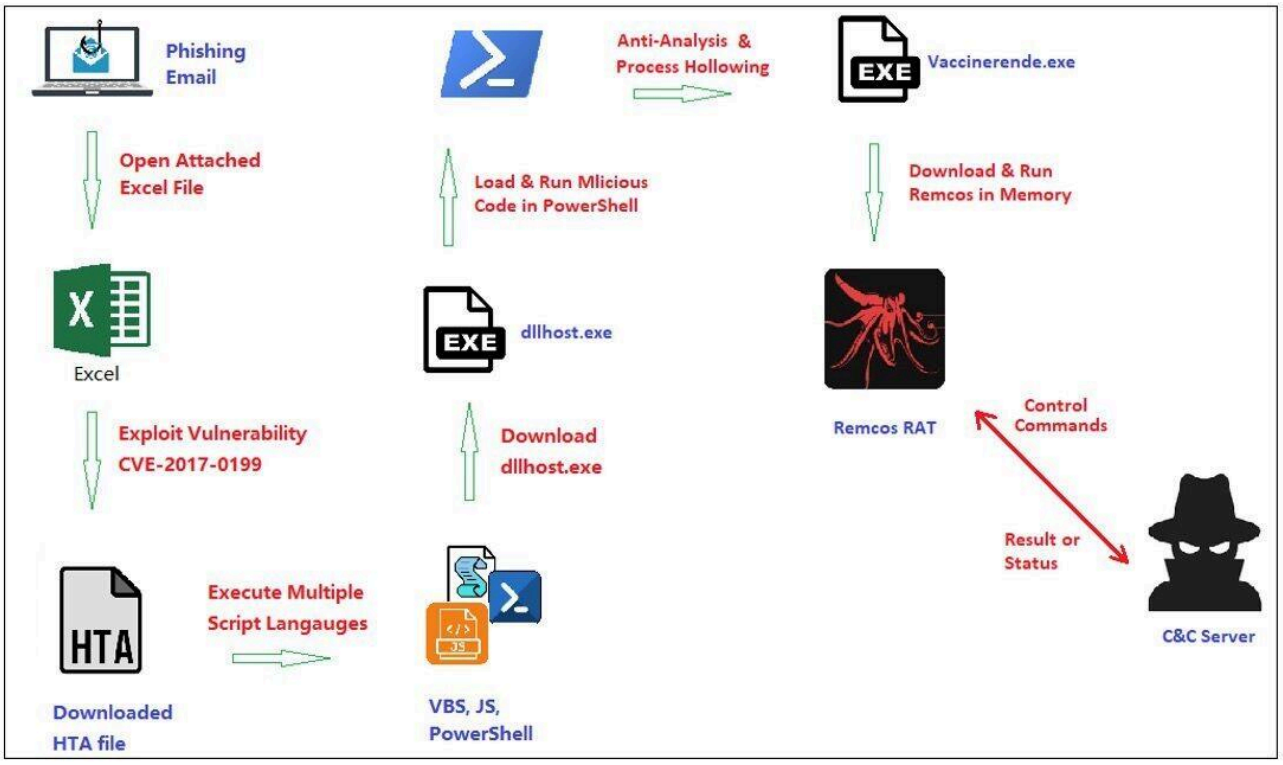

Figure 20: Workflow of the entire phishing campaign

Fortinet Protections

Fortinet customers are already protected from this campaign with FortiGuard's AntiSPAM, Web Filtering, IPS, and AntiVirus services as follows:

The relevant URLs are rated as "**Malicious Websites**" by the FortiGuard Web Filtering service.

FortiMail recognizes the phishing email as "virus detected." In addition, real-time anti-phishing provided by FortiSandbox embedded in Fortinet's FortiMail, web filtering, and antivirus solutions offers advanced protection against both known and unknown phishing attempts.

FortiGuard IPS service detects the vulnerability exploit against CVE-2017-0199 with the signature "**MS.Office.OLE.autolink.Code.Execution**".

FortiGuard Antivirus service detects the original Excel document, the HTA file, the downloaded executable file, the data/script files and the Recom executable file with the following AV signatures.

MSExcel/CVE-2017-0199.REM!exploit
JS/Remcos.CB!tr.dldr
PowerShell/Remcos.SER!tr
Data/Remcos.LAV!tr
W32/Remcos.LD!tr

FortiGate, FortiMail, FortiClient, and FortiEDR support the FortiGuard AntiVirus service. The FortiGuard AntiVirus engine is part of each solution. As a result, customers who have these products with up-to-date

protections are already protected.

The FortiGuard CDR (content disarm and reconstruction) service can disarm the embedded link object inside the Excel document.

To stay informed of new and emerging threats, you can sign up to receive future alerts.

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

If you believe this or any other cybersecurity threat has impacted your organization, please contact our Global FortiGuard Incident Response Team.

IOCs

URLs:

hxxps://og1[.]in/2Rxzb3
hxxp://192[.]3[.]220[.]22/xampp/en/cookienetbookinetcahce.hta
hxxp://192[.]3[.]220[.]22/hFXELFSwRHRwqbE214.bin
hxxp://192[.]3[.]220[.]22/430/dllhost.exe

C2 server:

107[.]173[.]4[.]16:2404

Relevant Sample SHA-256:

[PO-9987689987.xls]
4A670E3D4B8481CED88C74458FEC448A0FE40064AB2B1B00A289AB504015E944

[cookienetbookinetcahce.hta]
F99757C98007DA241258AE12EC0FD5083F0475A993CA6309811263AAD17D4661

[dllhost.exe / Vaccinerende.exe]
9124D7696D2B94E7959933C3F7A8F68E61A5CE29CD5934A4D0379C2193B126BE

[Aerognosy.Res]
D4D98FDBE306D61986BED62340744554E0A288C5A804ED5C924F66885CBF3514

[Valvulate.Cru]
F9B744D0223EFE3C01C94D526881A95523C2F5E457F03774DD1D661944E60852

[Remcos / Decrypted hFXELFSwRHRwqbE214.bin]
24A4EBF1DE71F332F38DE69BAF2DA3019A87D45129411AD4F7D3EA48F506119D

Source: https://www.fortinet.com/blog/threat-research/new-campaign-uses-remcos-rat-to-exploit-victims