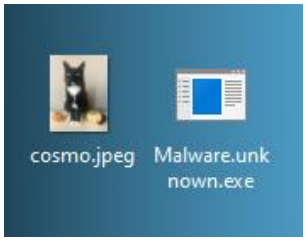
Practical Malware Analysis & Triage

Malware Analysis Report

Malware.unknown.exe



Data Exfiltration Malware

May 2024 | Tomer Merav | v1.0

Table of Contents

Executive Summary

SHA256 hash	81a10784ae60a58a969e858c9c4a2ae0d4ebe46e9bd6776992461c062f70099d

Malware.unknown.exe is a Data Exfiltration malware made for lab purposes.
This malware targets windows systems and was first identified in the wild in January 30,2022.
Upon detonation, the malware creates a remote connection to a remote server and establish permanent user account with elevated permissions.
After successful connection with the remote server, the malware waits for internet connectivity interruption(such as logging off) and then search the targeted exfiltration file. in our case it's an image file named "cosmo.jpeg".
When Internet connection is be back on, the malware will re-authenticate using its elevated permissions to a remote authentication server, followed by another query to the data exfiltration domain, finally result in an active process of data exfiltration between the compromised endpoint to the data exfiltration server.
This active data transfer, which is benign to the user, allows the attacker to maintain sustainable connection with the compromised endpoint while ensuring the process will continue until all the relevant data is exfiltrated to its intended place.

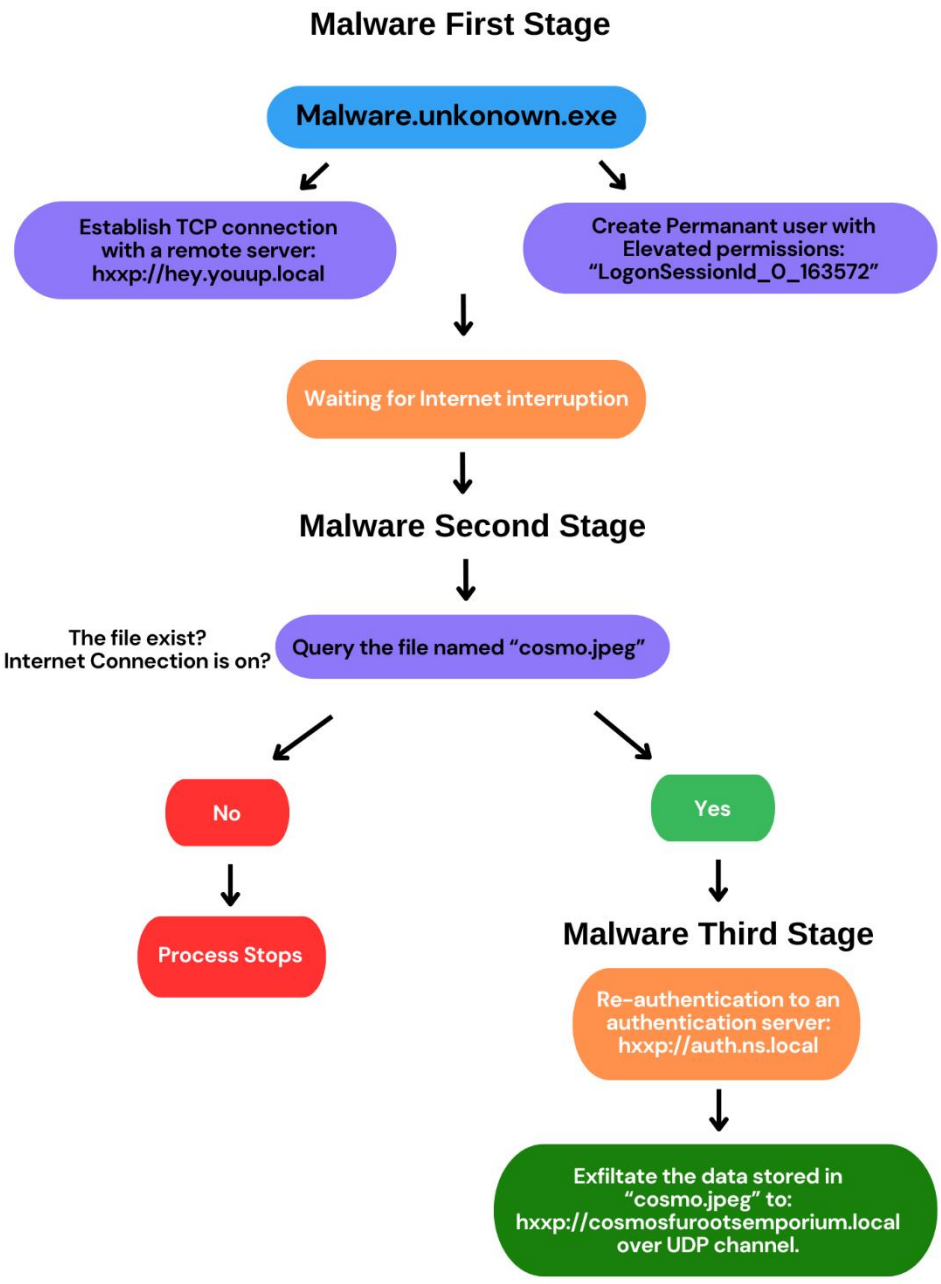
Symptoms of infection include frequent beaconing to any of the URLs listed in Appendix B.

YARA signature rules are attached in Appendix A. Malware sample and hashes have been submitted to VirusTotal for further examination.

High-Level Technical Summary

This Malware is based upon three execution stages.
The first stage is the actual detonation of the Malware. In this stage the malware will rely upon active internet connection in order to launch. When internet connection is available, the malware will create a permanent user account with elevated privileges to impersonate the endpoint's client.
Upon successful creation, the malware will open a communication channel with a remote server, supposedly controlled by the attacker.
Once this connection is established, the malware will wait for any internet connection interruption (such as logging off) while in the meantime no other actions will be taken. This methodology may suggest that this is part of an evasion technique in order to avoid detection.
When an action such as logging off will occur, the second stage of the malware process will be taken care of.
In this stage, the malware will query its targeted file and will search for its presence at its intended path.
If such file does not exist, the malware will stop its execution process and will not follow up with any other interaction with the system.
In case such file exists, it will set the tone for the third stage of the malware execution process, in which the malware will first re-authenticate using its elevated privileges, and then start to exfiltrate the data to a third remote server, considered as the data exfiltration domain.

While this malware demands a few key terms in order to further launch, it could be said that eventually the main two key points in the malware execution flow are the first user creation, and the presence of the targeted file. without them no other execution process will take place and the malware will not launch its intended processes.



Malware Composition

Malware.unknown.exe consists of the following components:

Malware.unknown.exe
The initial executable of the Malware.

"LogonSessionId_0_163572"
The name of the permanent user created by the malware with its elevated privileges.

hxxp://hey.youup.local
The first Callback URL the malware reaches to upon detonation.

hxxp://auth.ns.local
An authentication server the malware reaches to in order to reauthenticate before the final stage of exfiltrating the data.

cosmo.jpeg
The Image file that the Malware targets in order to exfiltrate its data.

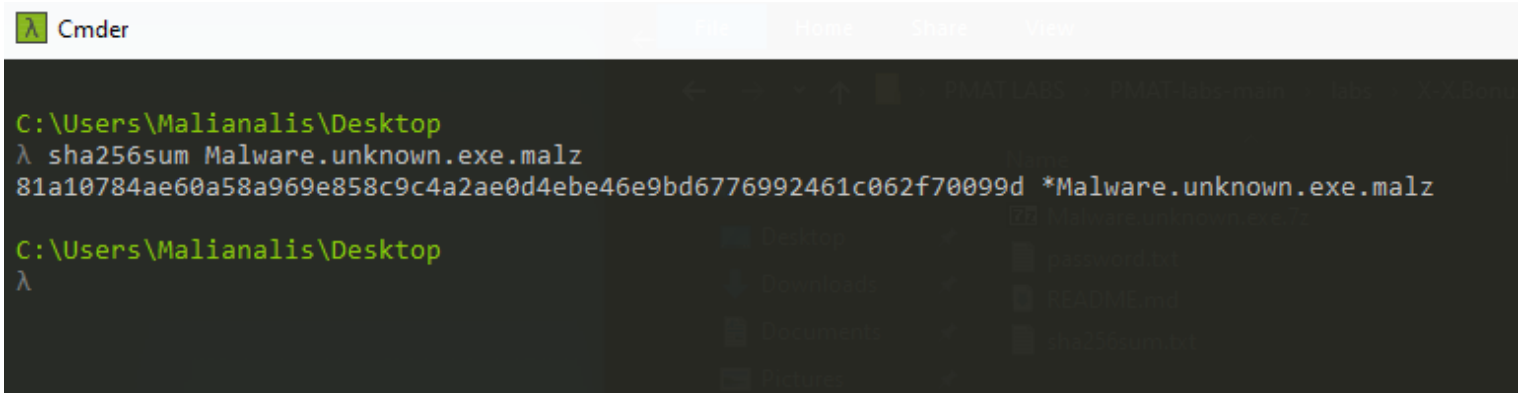
hxxp://cosmosfurootsemporium.local
The Data Exfiltration Domain that is being used by the malware in order to exfiltrate the data from the image file "cosmo.jpeg".

Basic Static Analysis

As we start our Basic Static Analysis Phase we are looking to find as many details as possible regarding the Malware sample without actually execute it.

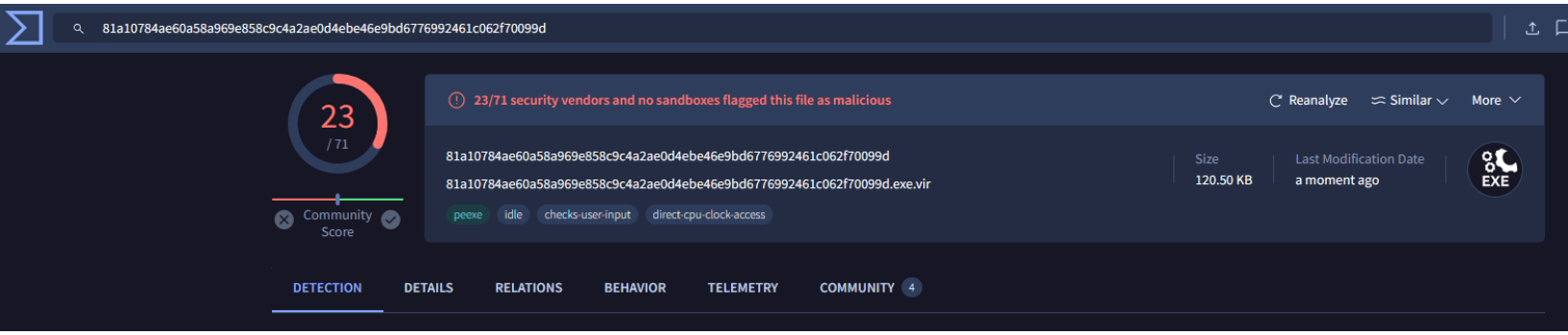
The first step in this methodology is to gather the relevant hashes from the file.

Running the command "sha256sum" provided me the sha256 hash of our sample:



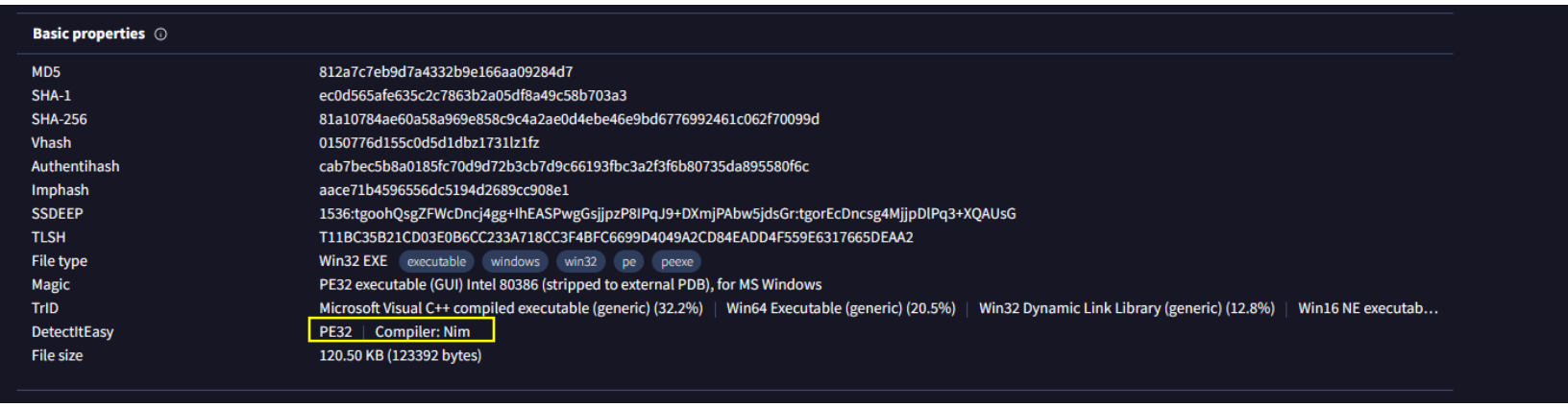
Next, I moved to VT and submitted the hash I got.

From our hash submission we can see 23/71 flagged the Malware sample as malicious.

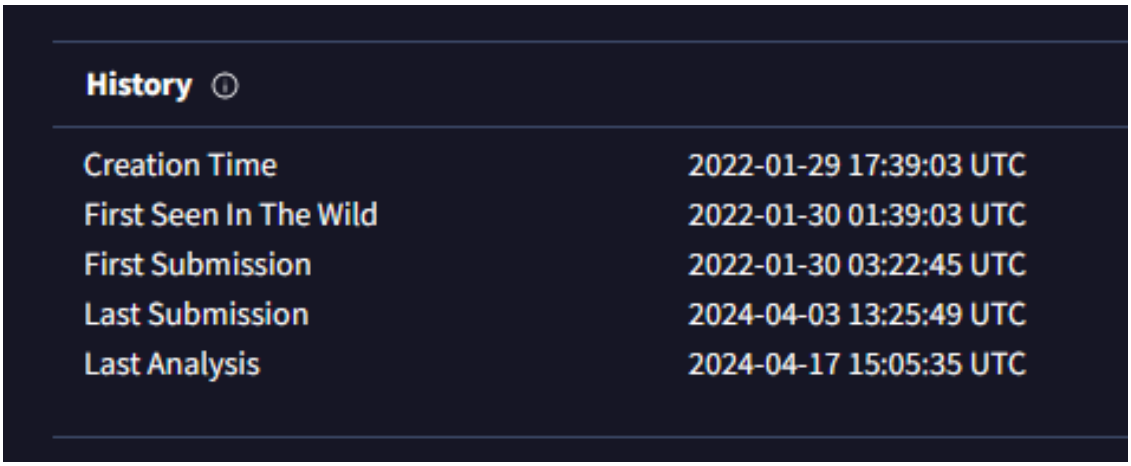


Through this screenshot we can learn as well that our file is an executable and ends with ".vir" extension-meaning some kind of Anti-virus software already identified this file as a file that contains malicious content.

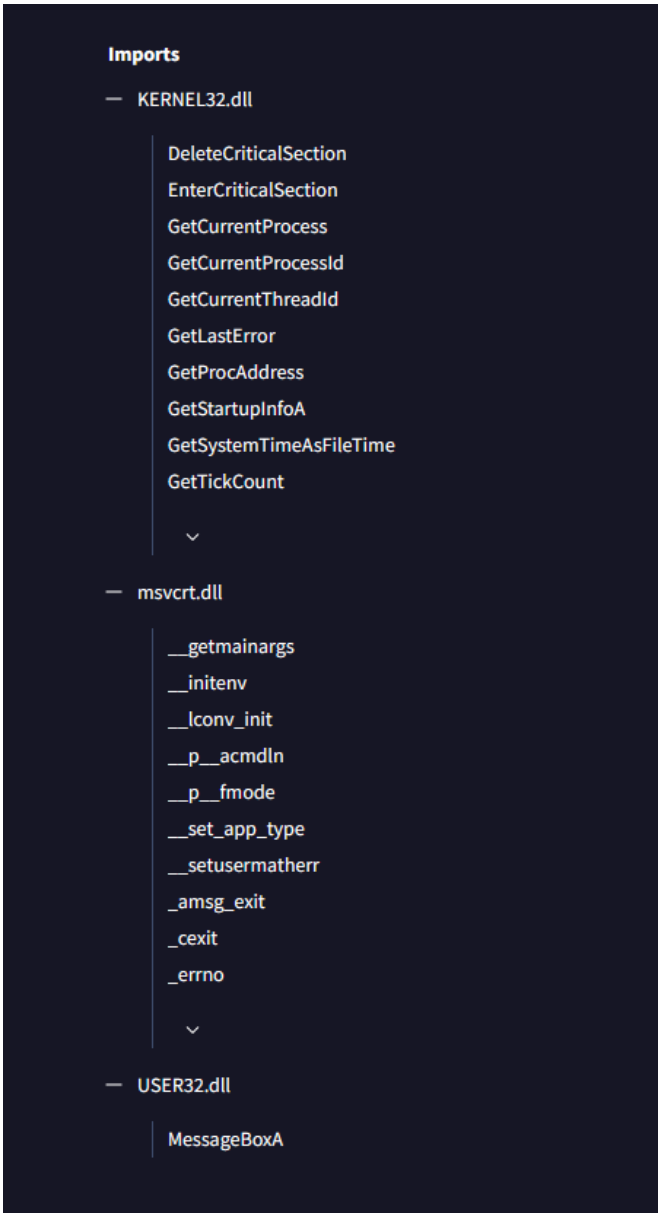
Moving on to the "Details" section we gather the basic properties of this file, such as its other hashes and its size. From this section we can also get another important detail –this file has been compiled with "Nim", one of the most popular methods malware authors use to execute their malicious commands.



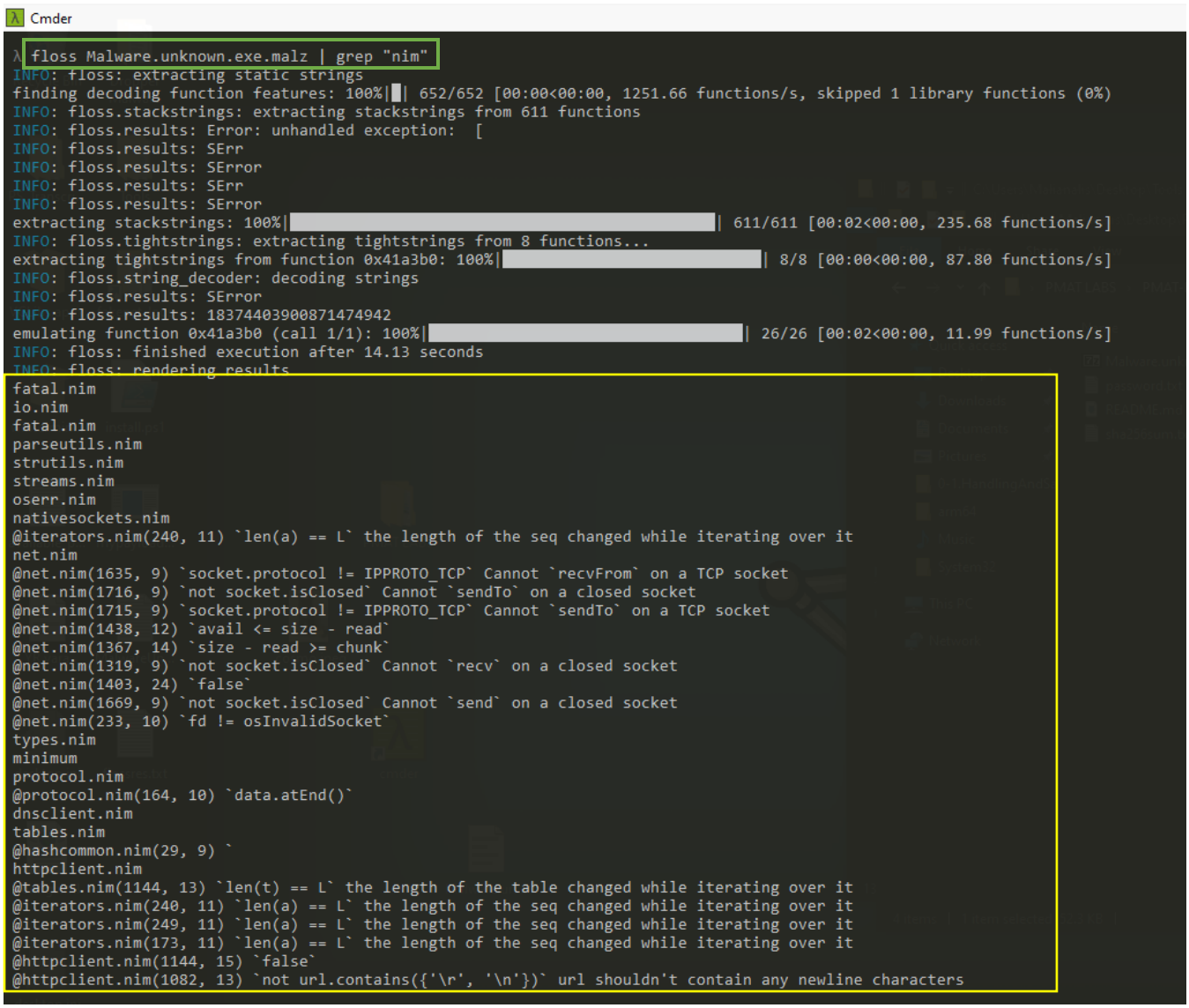
Moving on to "History" section we can see the file creation time : 2022-01-29, and also the first time it was seen in the wild, just a day after.



Another important information we gather from VT is the .dll libraries that is being used by the malware in order to execute its actions. a library such as "KERNEL32.dll" is a crucial library since this library provides direct relationship with the operating system in the compromised endpoint.



Moving back to our Windows lab Machine, I used the "floss" command to retrieve some suspicious strings that lays inside the sample. since we found out through VT that this file has been compiled with "Nim", I'll first try to search for suspicious strings that contains the phrase "nim", using the "grep" command:

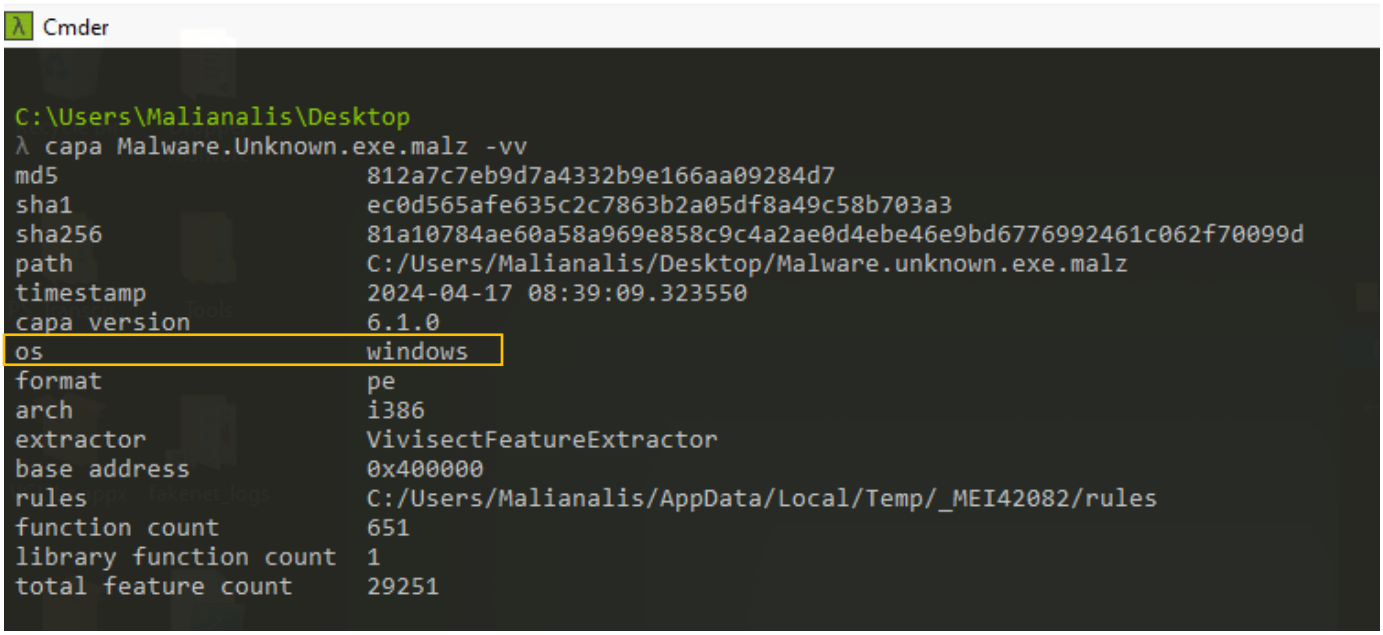


As we can see, we got a lot of results regarding "nim". but most important, we got our first indicators that this file uses some kind of internet connectivity.

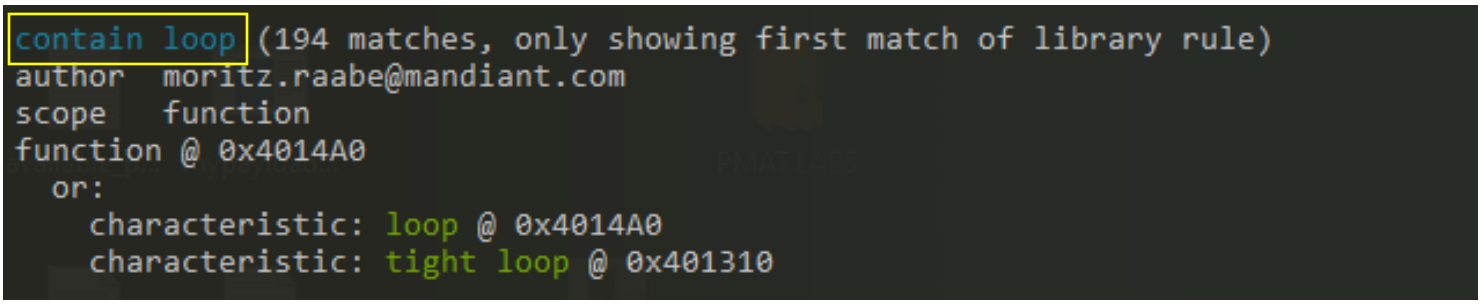
Strings such as "dnsclient.nim", and "httpclient.nim" confirm that assumption.

Moving on with this I used the "capa" command in order to find details regarding the background processes that are being executed by the malware.

As we can see the system that is being targeted by the malware is "windows".



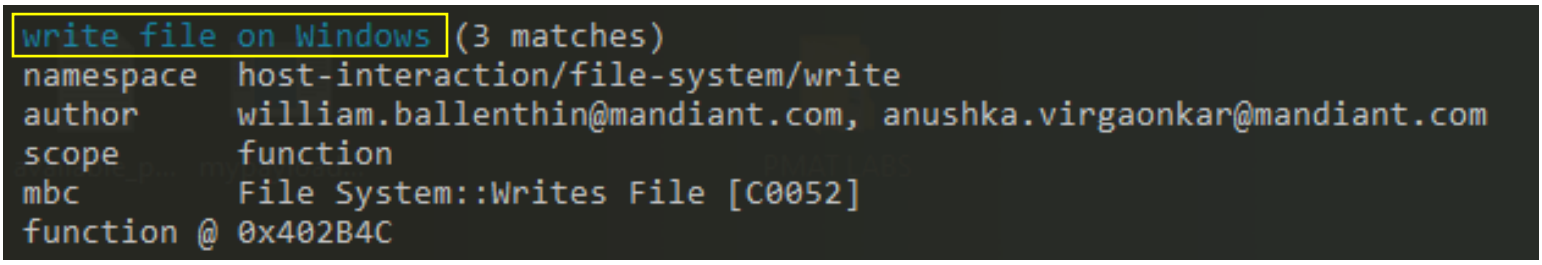
Next, let's look at some of its background processes:

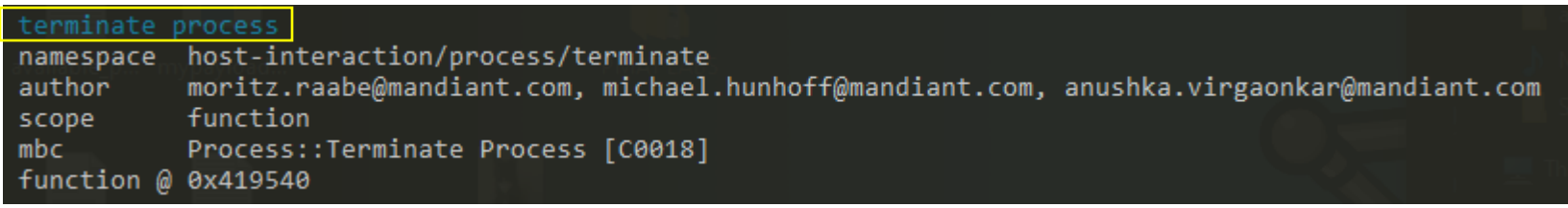


```
delay execution (library rule)
author      michael.hunhoff@mandiant.com, @ramen0x3f
scope       basic block
mbc         Anti-Behavioral Analysis::Dynamic Analysis Evasion::Delayed Execution [B0003.003]
references  https://docs.microsoft.com/en-us/windows/win32/sync/wait-functions, https://github.com/LordNoteworthy/al-kha
basic block @ 0x4011B0 in function 0x4014A0
  or:
    and:
      os: windows
      or:
        api: kernel32.Sleep @ 0x4011B7
```

```
encode data using Base64
namespace  data-manipulation/encoding/base64
author     moritz.raabe@mandiant.com, anushka.virgaonkar@mandiant.com, michael.hunhoff@mandiant.com
scope      function
att&ck     Defense Evasion::Obfuscated Files or Information [T1027]
mbc        Defense Evasion::Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02], Data::Encode Data::E
function @ 0x412168
```

```
read file on Windows (2 matches)
namespace  host-interaction/file-system/read
author     moritz.raabe@mandiant.com, anushka.virgaonkar@mandiant.com
scope      function
mbc        File System::Read File [C0051]
function @ 0x40249D
```





Let's summarize those findings:

- This file contains some sort of a loop process.
- It "reads" a file on the system- meaning It searches for a specific file.
- it "writes" to the file some data/content.
- It encodes the data using "Base64" as the encoding method.
- It uses delayed execution, maybe in order to evade detection.
- It uses the API call "Terminate process" to end its processes in some circumstances.

So all in all, we found a great information using the "capa" command. But this is not all.

Running again the "floss" command, this time without searching for specific strings, we came up with very interesting information:

```
@Desktop\cosmo.jpeg
@200 OK
@Authorization
@Host
@httpclient.nim(1144, 15) `false`
@Transfer-Encoding
@Content-Type
@Content-Length
@httpclient.nim(1082, 13) `not url.contains({'\r', '\n'})` url shouldn't contain any newline characters
@Nim httpclient/1.6.2
@hwtwtwpw:w/w/whwewyw.wywowuwuwpw.wlwowcwawlw
@axuxtxhx.xnxsx.xlxoxcxaxlx
@.BcBoBsBmBoBsBfBuBrBbBoBoBtBsBeBmBpBoBrBiBuBmB.BlBoBcBaBlB
```

.

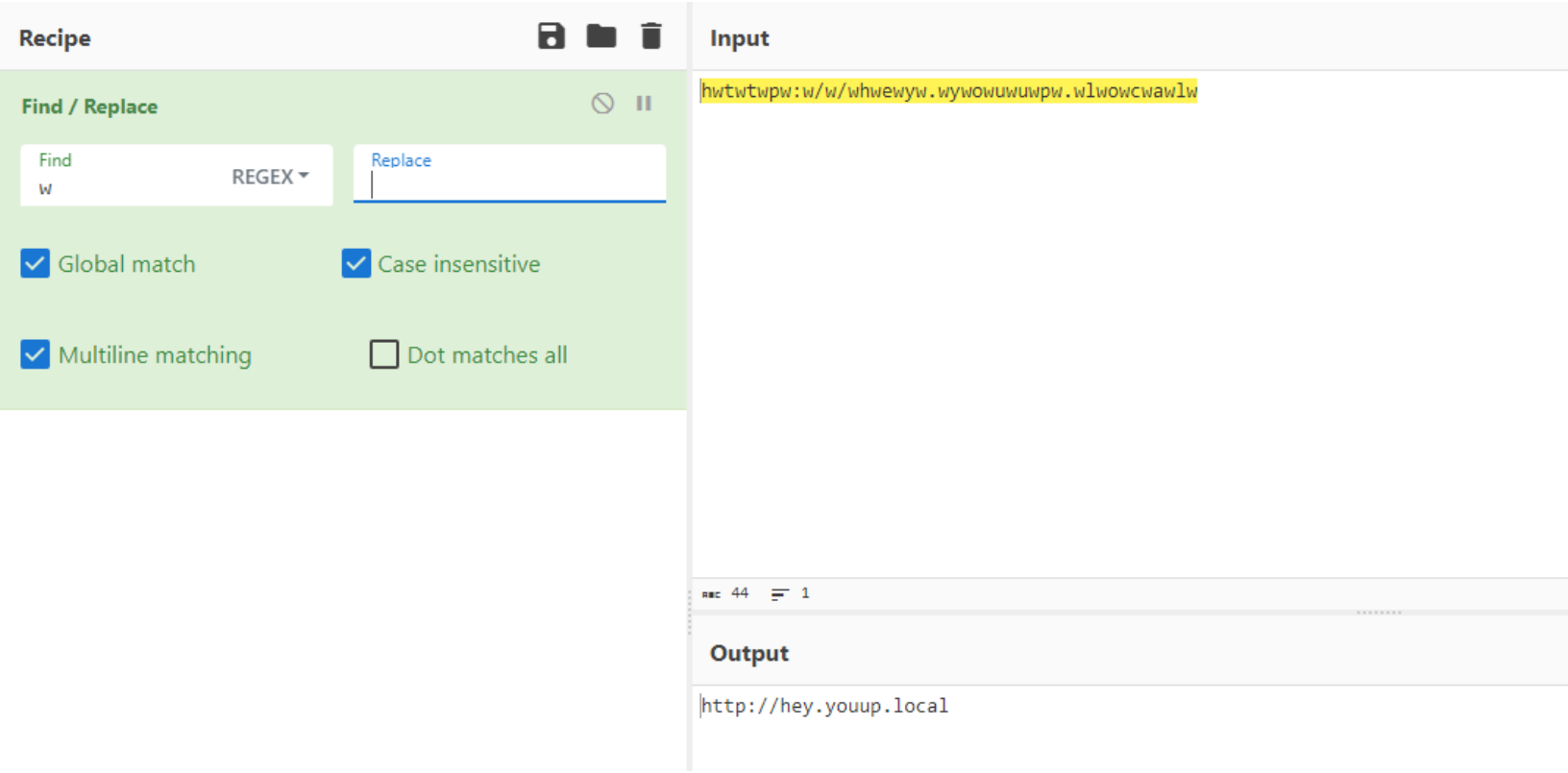
As we saw before, a clue for remote connection appears in here. But most important we found out a few other things:

- A file named "cosmo.jpeg" at the following path: "Desktop\cosmo.jpeg".
- Down the list, 3 suspicious strings that looks like an obfuscated internet URLs.

In order to check those addresses we will use "CyberChef" as our de-obfuscated tool.

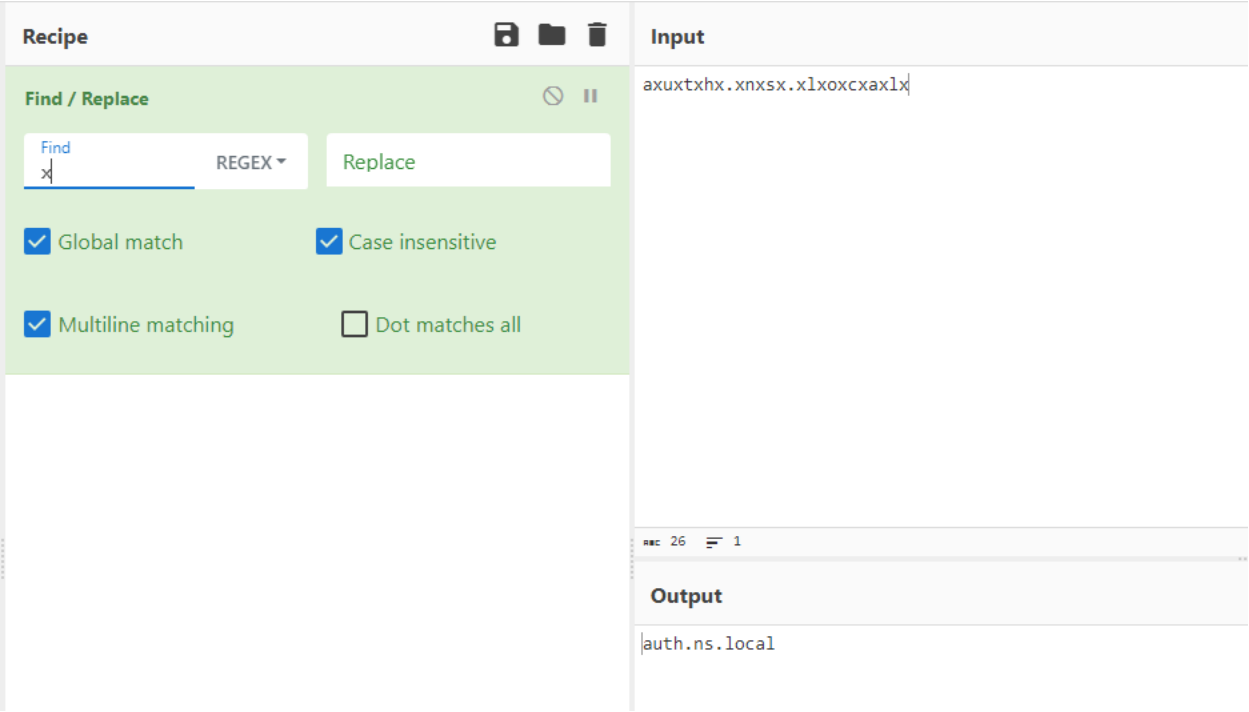
On our first string ("hwtwtwpw:w/w/whwewyw.wywowuwuwpw.wlwowcwawlw")

We can see the letter "w" is repetitive in here, so we will replace it with blank space what would give as a clean view of the string.



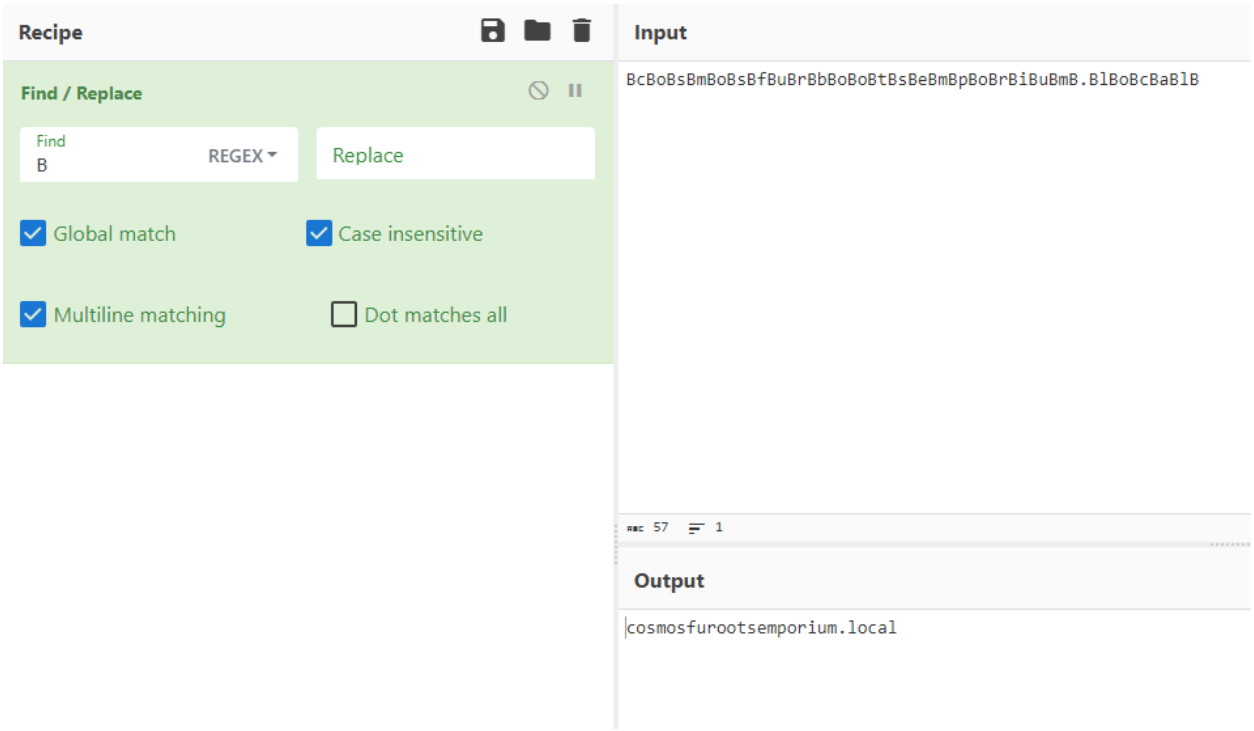
As we can see, our obfuscated string has become: **hxxp://hey.youup.local** after our de-obfuscated process.

Moving on with the same methodology, we can now take the second suspicious string ("axuxtxhx.xnxsx.xlxoxcxaxlx") and run it on CyberChef. This time replacing the "x" letter.



And there we have it! We got another URL: **auth.ns.local**

Lastly, let's check for the third string: "BcBoBsBmBoBsBfBuBrBbBoBoBtBsBeBmBpBoBrBiBuBmB.BlBoBcBaBlB" and this time replace the letter "B".



And as wonderful it can it be, we got our third URL that lays under this malware sample: **hxxp://cosmosfurootsemporium.local.**

So all in all. From this analysis we were able to retrieve 3 suspicious URL Strings:

hxxp://hey.youup.local

auth.ns.local

hxxp://cosmosfurootsemporium.local.

Let's move now to Pestudio to examine more details about the sample.

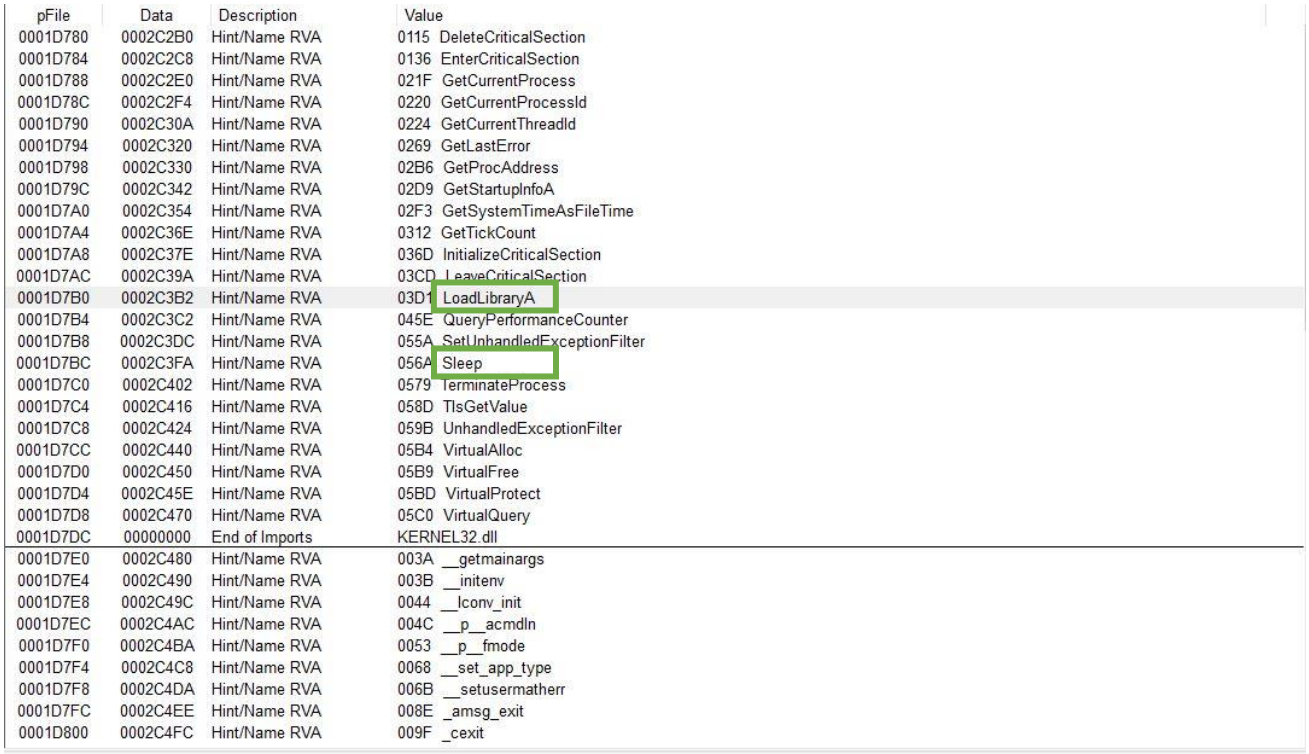
As we open the file details, we can confirm it is an executable (first-bytes-hex starts with 4D 5A) and also the compilation time match the details what we found in VT.

property	value
footprint > sha256	81A10784AE60A58A969E858C9C4A2AE0D4EBE46E9BD6776992461C062F70099D
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
first-bytes-text	M Z . . . . . @ . . . . .
file > size	123392 bytes
entropy	6.402
signature	n/a
tooling	n/a
file-type	executable
cpu	32-bit
subsystem	GUI
file-version	n/a
description	n/a
stamps	
compiler-stamp	Sat Jan 29 17:39:03 2022 | UTC

Moving on to the strings section we can see few suspicious strings such as "socket", "connect" and "WSAStartup", all three of them gives us another indicators regarding the malware remote communication.

encoding (2)	size (bytes)	location	flag (26)	label (77)	group (10)	technique (6)	value
ascii	19	section:.idata	x	import	reconnaissance	T1057 | Process Discovery	GetCurrentProcessId
ascii	25	section:.rdata	x	-	reconnaissance	-	QueryPerformanceFrequency
ascii	7	section:.rdata	x	utility	network	-	connect
ascii	4	section:.rdata	x	utility	network	-	send
ascii	6	section:.rdata	x	utility	network	-	select
ascii	9	section:.rdata	x	-	network	-	inet_ntop
ascii	10	section:.rdata	x	-	network	-	WSAStartup
ascii	6	section:.rdata	x	-	network	-	socket
ascii	11	section:.rdata	x	-	network	-	closesocket
ascii	8	section:.rdata	x	-	network	-	WSAIoctl
ascii	11	section:.rdata	x	-	network	-	getaddrinfo
ascii	12	section:.rdata	x	-	network	-	freeaddrinfo
ascii	12	section:.rdata	x	-	network	-	__WSAFDIsSet
ascii	4	section:.rdata	x	-	network	-	recv
ascii	6	section:.rdata	x	-	network	-	sendto
ascii	8	section:.rdata	x	-	network	-	recvfrom
ascii	9	section:.rdata	x	-	network	-	inet_ntoa
ascii	18	section:.rdata	x	-	network	-	WSAAddressToString
ascii	6	section:.rdata	x	-	network	-	socket
ascii	6	section:.rdata	x	-	network	-	socket
ascii	12	section:.idata	x	import	memory	T1055 | Process Injection	VirtualAlloc
ascii	14	section:.idata	x	import	memory	T1055 | Process Injection	VirtualProtect
ascii	13	section:.rdata	x	-	file	T1083 | File and Directory Discovery	FindFirstFile
ascii	17	section:.idata	x	import	execution	T1057 | Process Discovery	GetCurrentProcess
ascii	18	section:.idata	x	import	execution	T1057 | Process Discovery	GetCurrentThreadId
ascii	16	section:.idata	x	import	execution	-	TerminateProcess

Moving to the Import section, we identified few suspicious API Calls such as "LoadLibraryA" that generates the call for the relevant dll's and the "Sleep" API Call that can be used in order to evade detection.



pFile	Data	Description	Value
0001D780	0002C2B0	Hint/Name RVA	0115 DeleteCriticalSection
0001D784	0002C2C8	Hint/Name RVA	0136 EnterCriticalSection
0001D788	0002C2E0	Hint/Name RVA	021F GetCurrentProcess
0001D78C	0002C2F4	Hint/Name RVA	0220 GetCurrentProcessId
0001D790	0002C30A	Hint/Name RVA	0224 GetCurrentThreadId
0001D794	0002C320	Hint/Name RVA	0269 GetLastError
0001D798	0002C330	Hint/Name RVA	02B6 GetProcAddress
0001D79C	0002C342	Hint/Name RVA	02D9 GetStartupInfoA
0001D7A0	0002C354	Hint/Name RVA	02F3 GetSystemTimeAsFileTime
0001D7A4	0002C36E	Hint/Name RVA	0312 GetTickCount
0001D7A8	0002C37E	Hint/Name RVA	036D InitializeCriticalSection
0001D7AC	0002C39A	Hint/Name RVA	03CD LeaveCriticalSection
0001D7B0	0002C3B2	Hint/Name RVA	03D1 LoadLibraryA
0001D7B4	0002C3C2	Hint/Name RVA	045E QueryPerformanceCounter
0001D7B8	0002C3DC	Hint/Name RVA	055A SetUnhandledExceptionFilter
0001D7BC	0002C3FA	Hint/Name RVA	056A Sleep
0001D7C0	0002C402	Hint/Name RVA	0579 TerminateProcess
0001D7C4	0002C416	Hint/Name RVA	058D TlsGetValue
0001D7C8	0002C424	Hint/Name RVA	059B UnhandledExceptionFilter
0001D7CC	0002C440	Hint/Name RVA	05B4 VirtualAlloc
0001D7D0	0002C450	Hint/Name RVA	05B9 VirtualFree
0001D7D4	0002C45E	Hint/Name RVA	05BD VirtualProtect
0001D7D8	0002C470	Hint/Name RVA	05C0 VirtualQuery
0001D7DC	00000000	End of Imports	KERNEL32.dll
0001D7E0	0002C480	Hint/Name RVA	003A __getmainargs
0001D7E4	0002C490	Hint/Name RVA	003B __initenv
0001D7E8	0002C49C	Hint/Name RVA	0044 __lconv_init
0001D7EC	0002C4AC	Hint/Name RVA	004C __p__acmdln
0001D7F0	0002C4BA	Hint/Name RVA	0053 __p__fmode
0001D7F4	0002C4C8	Hint/Name RVA	0068 __set_app_type
0001D7F8	0002C4DA	Hint/Name RVA	006B __setusermatherr
0001D7FC	0002C4EE	Hint/Name RVA	008E _amsg_exit
0001D800	0002C4FC	Hint/Name RVA	009F _cexit

To summarize this section of analysis, we gathered very important information regarding the sample actions, such as it origin, the way it was written, key factors regarding the actions of the sample, and some suspicious addresses that this malware may connect with.

While all of this is very useful information, we still cannot be sure regarding what we found until we'll run our sample. for that, we'll move now to the next section of analysis, also called the Dynamic Analysis Phase.

Basic Dynamic Analysis

This is now the time to start examining the malware actions in real time. for this phase of analysis, I used Process Montior (Procmon),Wireshark and TCPView as the tools in which will examine our malware actions.

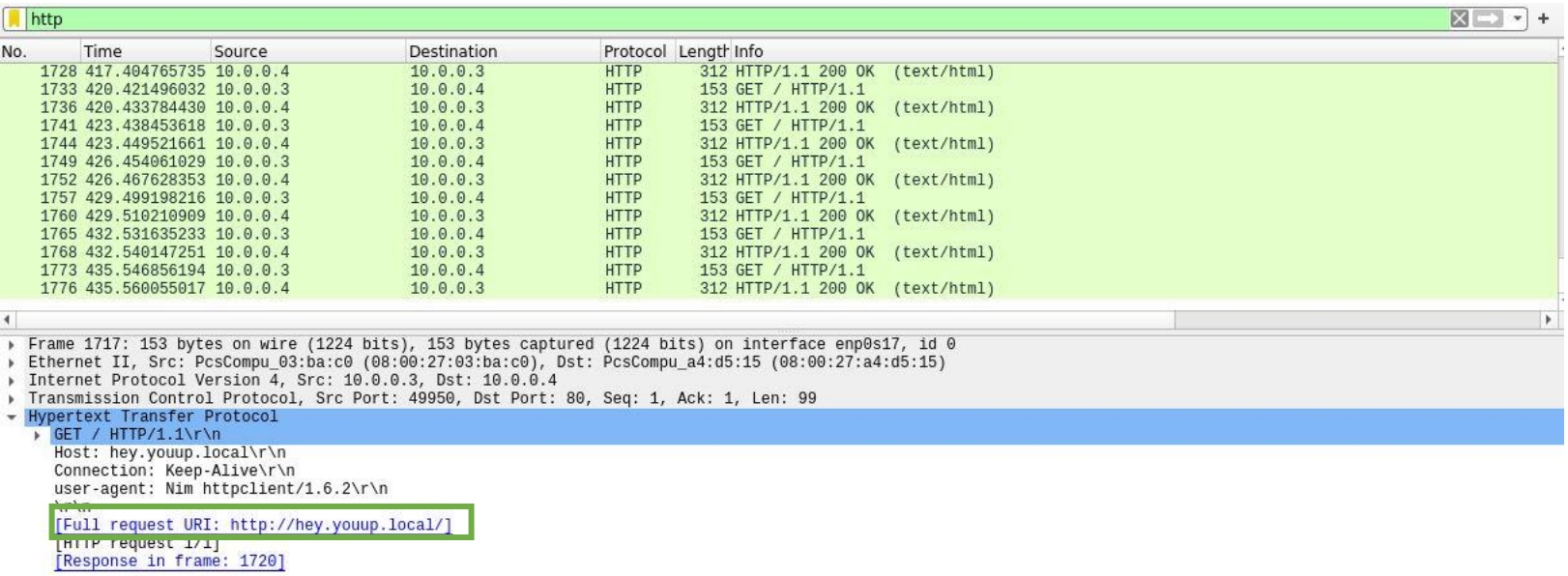
Upon detonating the malware, I opened up Procmon and observed the following TCP connections:

10:29:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49765 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:29:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49766 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:29:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49767 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:29:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49768 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:29:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49769 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:29:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49770 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:29:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49771 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:29:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49772 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:29:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49773 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:29:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49774 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:29:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49775 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:29:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49776 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49777 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49778 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49779 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49780 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49781 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49782 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49783 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49784 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49785 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49786 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49787 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49788 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49789 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49790 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49791 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49792 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49793 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49794 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49795 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:30:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49796 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:31:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49797 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:31:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49798 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:31:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49799 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:31:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49800 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:31:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49801 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:31:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49802 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:31:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49803 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:31:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49804 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:31:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49805 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:31:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49806 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:31:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49807 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...
10:31:...	Malware.unkno...	820	TCP Connect	DESKTOP-PUAFCAM:49808 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 14...

Another validation of this activity has been observed using TCPView. As we can see, a connection has been made between various local ports to port 80.(HTTP)

Process Name	Process ID	Protocol	State	Local Address	Local Port	Remote Address	Remote Port	Create Time	Module Name	Sent Packets
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49774	10.0.0.4	80	5/7/2024 5:26:29 AM	Malware.unknown.exe	
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49775	10.0.0.4	80	5/7/2024 5:26:32 AM	Malware.unknown.exe	1
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49776	10.0.0.4	80	5/7/2024 5:26:35 AM	Malware.unknown.exe	
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49777	10.0.0.4	80	5/7/2024 5:26:38 AM	Malware.unknown.exe	1
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49778	10.0.0.4	80	5/7/2024 5:26:41 AM	Malware.unknown.exe	
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49779	10.0.0.4	80	5/7/2024 5:26:44 AM	Malware.unknown.exe	1
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49780	10.0.0.4	80	5/7/2024 5:26:47 AM	Malware.unknown.exe	
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49781	10.0.0.4	80	5/7/2024 5:26:50 AM	Malware.unknown.exe	1
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49782	10.0.0.4	80	5/7/2024 5:26:53 AM	Malware.unknown.exe	
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49783	10.0.0.4	80	5/7/2024 5:26:56 AM	Malware.unknown.exe	1
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49784	10.0.0.4	80	5/7/2024 5:26:59 AM	Malware.unknown.exe	
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49785	10.0.0.4	80	5/7/2024 5:27:02 AM	Malware.unknown.exe	1
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49786	10.0.0.4	80	5/7/2024 5:27:05 AM	Malware.unknown.exe	1
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49787	10.0.0.4	80	5/7/2024 5:27:08 AM	Malware.unknown.exe	
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49788	10.0.0.4	80	5/7/2024 5:27:11 AM	Malware.unknown.exe	1
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49789	10.0.0.4	80	5/7/2024 5:27:14 AM	Malware.unknown.exe	
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49790	10.0.0.4	80	5/7/2024 5:27:17 AM	Malware.unknown.exe	1
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49791	10.0.0.4	80	5/7/2024 5:27:20 AM	Malware.unknown.exe	
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49792	10.0.0.4	80	5/7/2024 5:27:23 AM	Malware.unknown.exe	1
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49793	10.0.0.4	80	5/7/2024 5:27:26 AM	Malware.unknown.exe	
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49794	10.0.0.4	80	5/7/2024 5:27:29 AM	Malware.unknown.exe	1
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49795	10.0.0.4	80	5/7/2024 5:27:32 AM	Malware.unknown.exe	
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49796	10.0.0.4	80	5/7/2024 5:27:35 AM	Malware.unknown.exe	1
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49797	10.0.0.4	80	5/7/2024 5:27:39 AM	Malware.unknown.exe	
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49798	10.0.0.4	80	5/7/2024 5:27:42 AM	Malware.unknown.exe	1
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49799	10.0.0.4	80	5/7/2024 5:27:45 AM	Malware.unknown.exe	
Malware.unknown.exe	2892	TCP	Close Wait	10.0.0.3	49800	10.0.0.4	80	5/7/2024 5:27:48 AM	Malware.unknown.exe	1

Then I moved into Wireshark and saw a lot of HTTP requests that are being sent to hxxp://hey.youup.local- our first found string from the basic static analysis phase. It's seems like this string is actually a C&C server that the malware reach to upon detonation.



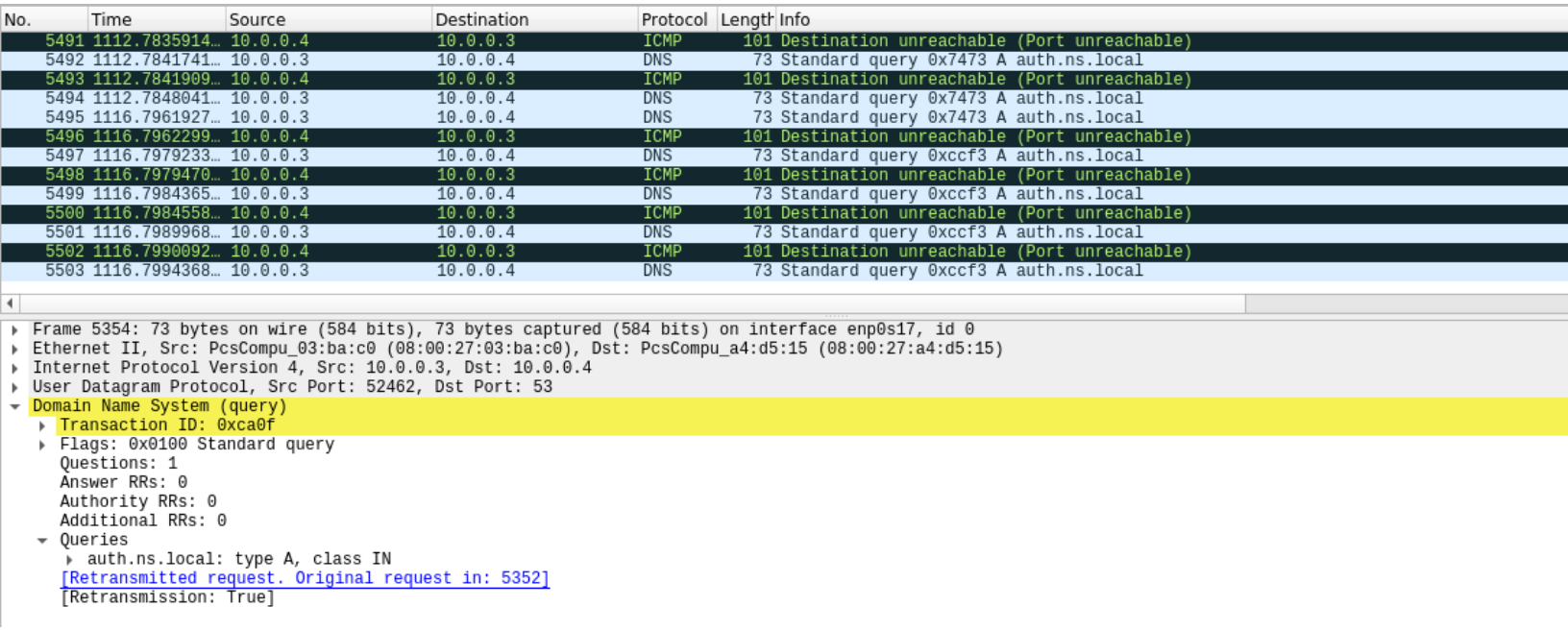
Since we suspect the malware targets specific file on the file system (cosmo.jpeg) we want to search for such indicators.

While moving on through those processes in Procmon, I didn't find any indication that this is what the malware does, so then I decided to interrupt the connection between our endpoint to the C&C server and disconnected from the internet.

After 4 attempts to reconnect, the TCP Connection got disconnected, and after few processes, we can see it search specifically the "cosmo.jpeg" file.

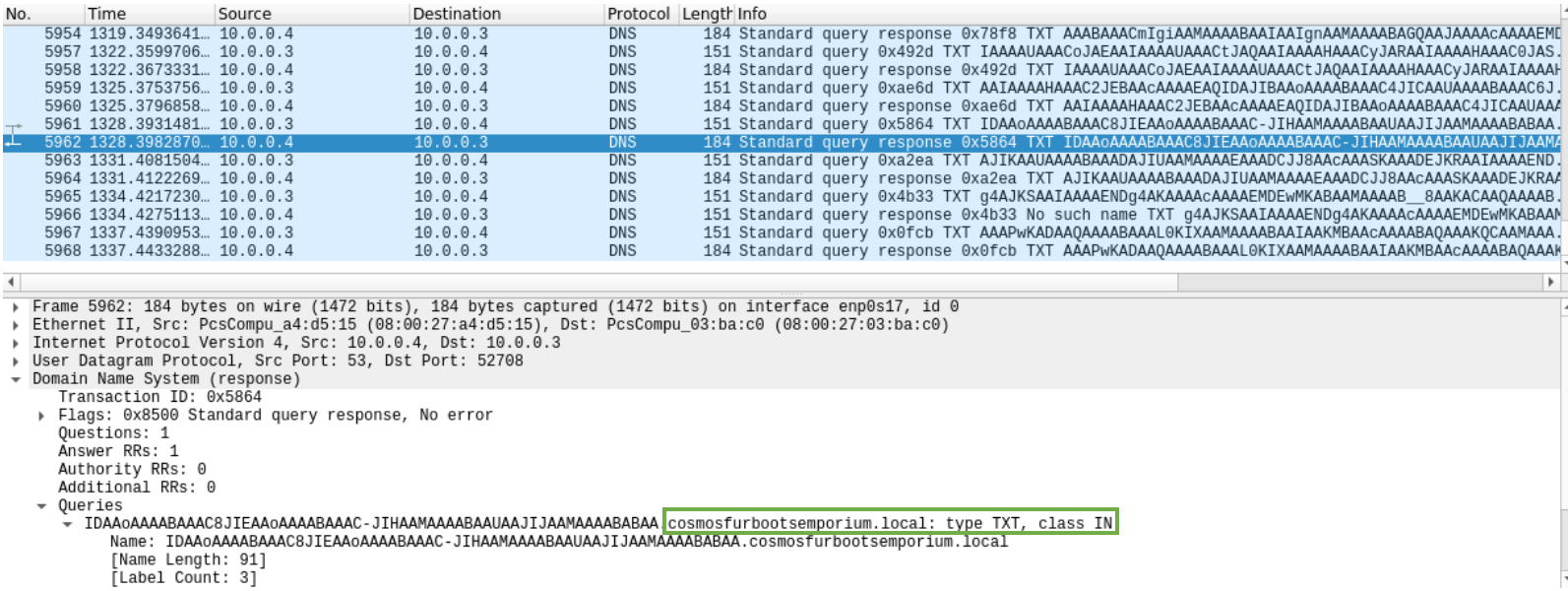
2:51:0...	Malware.unkno...	2512	TCP Reconnect	SUCCESS	DESKTOP-PUAFCAM:49816 -> www.inetsim.org:http	Length: 0, seqnum:...
2:51:0...	Malware.unkno...	2512	TCP Reconnect	SUCCESS	DESKTOP-PUAFCAM:49816 -> www.inetsim.org:http	Length: 0, seqnum:...
2:51:0...	Malware.unkno...	2512	TCP Reconnect	SUCCESS	DESKTOP-PUAFCAM:49816 -> www.inetsim.org:http	Length: 0, seqnum:...
2:51:0...	Malware.unkno...	2512	TCP Reconnect	SUCCESS	DESKTOP-PUAFCAM:49816 -> www.inetsim.org:http	Length: 0, seqnum:...
2:51:0...	Malware.unkno...	2512	TCP Disconnect	SUCCESS	DESKTOP-PUAFCAM:49816 -> www.inetsim.org:http	Length: 0, seqnum:...
2:51:0...	Malware.unkno...	2512	RegOpenKey	REPARSE	HKLM\Software\WOW6432Node\Microsoft\Language...	Desired Access: R...
2:51:0...	Malware.unkno...	2512	RegOpenKey	NAME NOT FOUND	HKLM\SOFTWARE\Microsoft\LanguageOverlay\Overl...	Desired Access: R...
2:51:0...	Malware.unkno...	2512	CreateFile	NAME NOT FOUND	C:\Windows\SysWOW64\en-US\KERNELBASE.dll.mui	Desired Access: R...
2:51:0...	Malware.unkno...	2512	CreateFile	SUCCESS	C:\Windows\System32\en-US\KernelBase.dll.mui	Desired Access: R...
2:51:0...	Malware.unkno...	2512	CreateFileMapp...	FILE LOCKED WI...	C:\Windows\System32\en-US\KernelBase.dll.mui	SyncType: SyncTy...
2:51:0...	Malware.unkno...	2512	QueryStandardI...	SUCCESS	C:\Windows\System32\en-US\KernelBase.dll.mui	AllocationSize: 1,3...
2:51:0...	Malware.unkno...	2512	CreateFileMapp...	SUCCESS	C:\Windows\System32\en-US\KernelBase.dll.mui	SyncType: SyncTy...
2:51:0...	Malware.unkno...	2512	ReadFile	SUCCESS	C:\Windows\SysWOW64\msvcrt.dll	Offset: 513,024, Le...
2:51:0...	Malware.unkno...	2512	CreateFile	SUCCESS	C:\Users\Malianalis\Desktop\cosmo.jpeg	Desired Access: G...
2:51:0...	Malware.unkno...	2512	QueryStandardI...	SUCCESS	C:\Users\Malianalis\Desktop\cosmo.jpeg	AllocationSize: 1,7...
2:51:0...	Malware.unkno...	2512	ReadFile	SUCCESS	C:\Users\Malianalis\Desktop\cosmo.jpeg	Offset: 0, Length: 1...
2:51:0...	Malware.unkno...	2512	ReadFile	SUCCESS	C:\Users\Malianalis\Desktop\cosmo.jpeg	Offset: 0, Length: 1...
2:51:0...	Malware.unkno...	2512	ReadFile	SUCCESS	C:\Users\Malianalis\Desktop\cosmo.jpeg	Offset: 1,753,088, ...
2:51:0...	Malware.unkno...	2512	ReadFile	END OF FILE	C:\Users\Malianalis\Desktop\cosmo.jpeg	Offset: 1,754,626, ...
2:51:0...	Malware.unkno...	2512	CloseFile	SUCCESS	C:\Users\Malianalis\Desktop\cosmo.jpeg	

In the meantime, I opened up Wireshark, and as we can see the malware tries to re-establish a connection with another remote server- hxxp://auth.ns.local- which is the second string we found in the basic static analysis phase. This local remote server may be used as a proxy server, in which the user will authenticate before moving to the actual data exfiltration domain. In this case, since the internet connection was unavailable, the query failed to re-establish that connection with the authentication server.



In order to examine what happens next, I retrieved back the internet and checked again both Wireshark and Procmon.

In Wireshark, an ongoing process has been started with a lot of DNS TXT queries, meaning the authentication process succeeded and the malware started active data transfer using those DNS TXT queries.



This is the time when we may start to suspect that those strings are actually the data the malware takes from our compromised file – "cosmo.jpeg", but since they are still under base64 form, we cannot be sure yet.

What we do able to see is that those TXT queries are being sent to hxxp://cosmosfurbootsemporium.local- the third string we found during the basic static analysis phase. this could suggest that this domain is the one in which the malware reach to in order to exfiltrate the data.

Moving on to Procmon, we can see a lot of UDP traffic that has been observed at the same time, what may point out that a large amount of data is being taking care of in a minimum time frame.

10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52699 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52700 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52700 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52701 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52701 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52702 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52702 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52703 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52703 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52704 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52704 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52705 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52705 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52706 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52706 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52707 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52707 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52708 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52708 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52709 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52709 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52710 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52710 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52711 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52711 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52712 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52712 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52713 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52713 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52714 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52714 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52715 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52715 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:51:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52716 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:51:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52716 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:52:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52717 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:52:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52717 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:52:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52718 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:52:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52718 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:52:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52719 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:52:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52719 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:52:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52720 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:52:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52720 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:52:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52721 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:52:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52721 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...
10:52:...	Malware.unkno...	820	UDP Send	DESKTOP-PUAFCAM:52722 -> www.inetsim.org:domain	SUCCESS	Length: 109, seqn...
10:52:...	Malware.unkno...	820	UDP Receive	DESKTOP-PUAFCAM:52722 -> www.inetsim.org:domain	SUCCESS	Length: 142, seqn...

Also in TCPView, same pattern has been observed:

TCPView - Sysinternals: www.sysinternals.com

File Edit View Process Connection Options Help

4 TCP v4 | 6 TCP v6 | 4 UDP v4 | 6 UDP v6 | malware

Process Name	Process ID	Protocol	State	Local Address	Local Port	Remote Address	Remote Port	Create Time	Module Name
Malware.unknown.exe	2992	UDP		0.0.0.0	52897	*		5/7/2024 8:57:14 AM	Malware.unknown.exe
Malware.unknown.exe	2992	UDP		0.0.0.0	52898	*		5/7/2024 8:57:17 AM	Malware.unknown.exe
Malware.unknown.exe	2992	UDP		0.0.0.0	52899	*		5/7/2024 8:57:20 AM	Malware.unknown.exe
Malware.unknown.exe	2992	UDP		0.0.0.0	52900	*		5/7/2024 8:57:23 AM	Malware.unknown.exe
Malware.unknown.exe	2992	UDP		0.0.0.0	52901	*		5/7/2024 8:57:26 AM	Malware.unknown.exe
Malware.unknown.exe	2992	UDP		0.0.0.0	52902	*		5/7/2024 8:57:29 AM	Malware.unknown.exe
Malware.unknown.exe	2992	UDP		0.0.0.0	52903	*		5/7/2024 8:57:32 AM	Malware.unknown.exe
Malware.unknown.exe	2992	UDP		0.0.0.0	52904	*		5/7/2024 8:57:35 AM	Malware.unknown.exe
Malware.unknown.exe	2992	UDP		0.0.0.0	52905	*		5/7/2024 8:57:38 AM	Malware.unknown.exe
Malware.unknown.exe	2992	UDP		0.0.0.0	52906	*		5/7/2024 8:57:41 AM	Malware.unknown.exe

So all in all, it seems that malware uses DNS TXT Queries in order to exfiltrate data over UDP Channel, reaching hxxp://cosmosfurbootsemporium.local as its data exfiltration domain.

This process has successfully launched due to successful first connection with the remote sever, followed by successful authentication, that set to the tone for the malware to launch its exfiltration process.

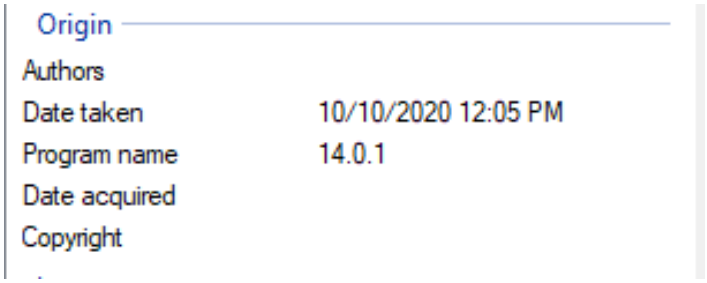
In our next section of analysis, we will try to examine more carefully those base64 strings in order to confirm our assumption that these strings are actually the data that has been taken from "cosmo.jpeg" file to the attacker remote server. Moreover, we will also be examining the malware actions using tools such as "Cutter" and "x32dbg" in order to examine those malware actions in a deeper manner.

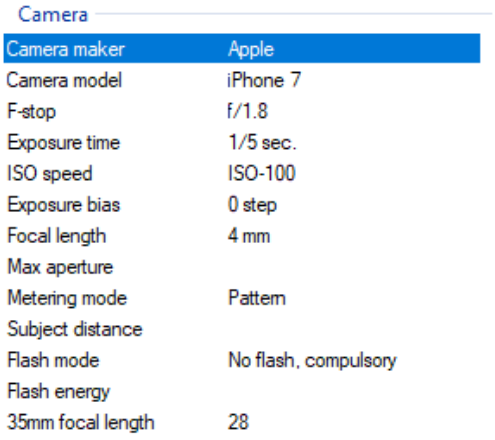
Advanced Analysis

As we approach to this section of analysis, we want to summarize what we found out so far:

- The malware exfiltrate data in some kind of a loop process over UDP channel, using base64 encoded strings.
- It supposedly targets "cosmo.jpeg" file.
- It reaches the following remote server when exfiltrating the data: hxxp://cosmosfurbootsemporium.local

What is still missing in here is what kind of data is being transferred. since we already saw proofs that the malware search for our "cosmo.jpeg" file, we may suspect that this file data is the one that is being targeted. to ensure this assumption, let's start with a simple step of gathering the file main details.





As we can see, this image file has been taken in 10/10/2020, 12:05PM.

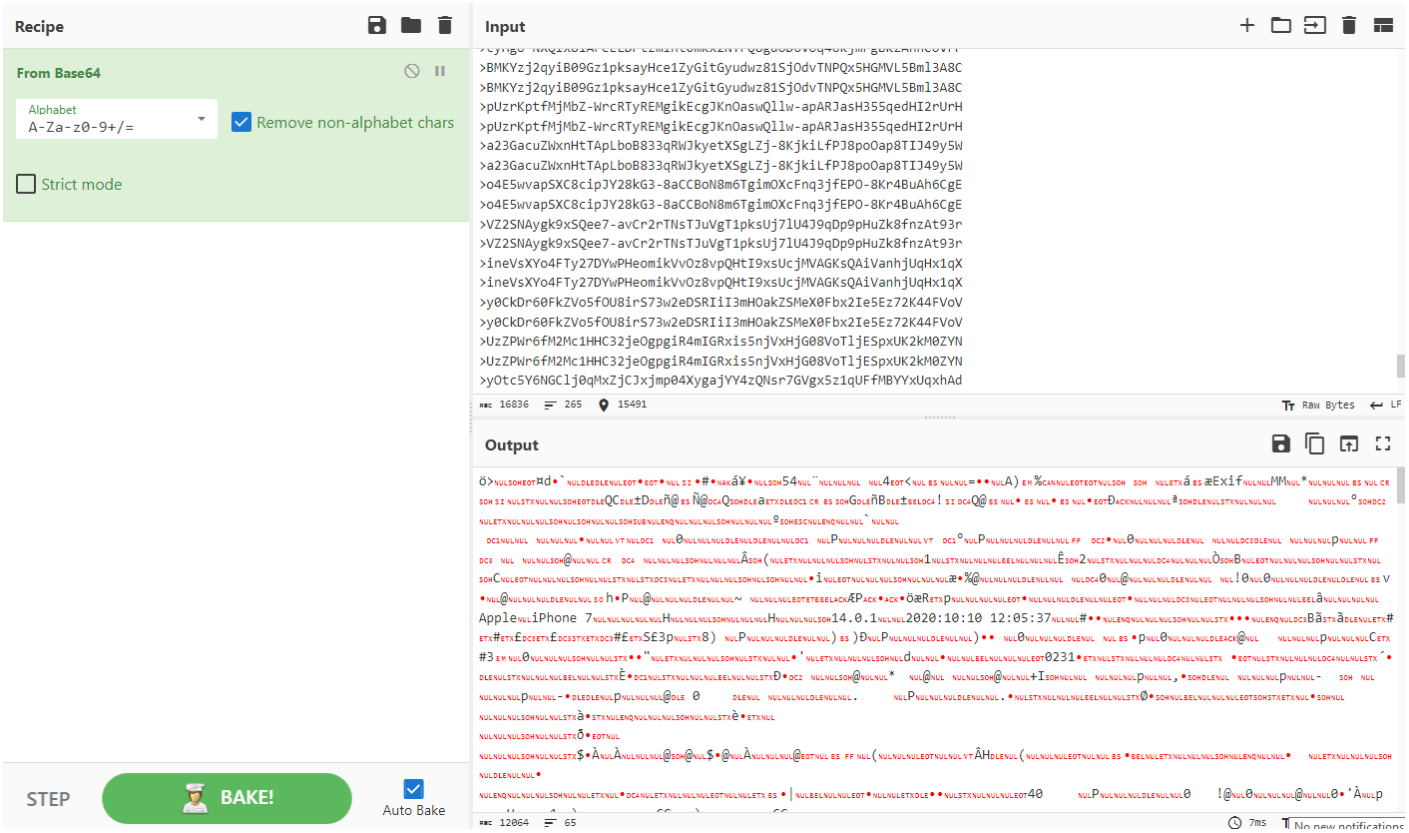
Also, we can see its was captured using an iPhone 7.

These details will be used as we approach to our next step of the analysis.

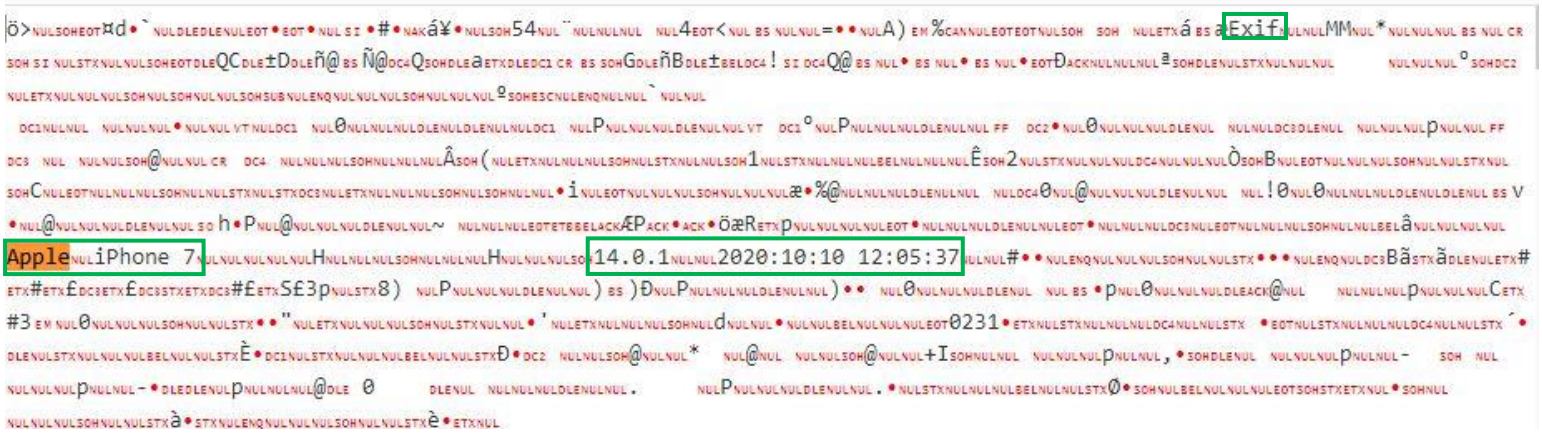
The next thing I did is to take the PCAP file from Wireshark, and using the command line I executed the "floss" command along with the "grep" command in order to retrieve the base64 strings that lay inside the file.

```
floss "traffic wireshark.pcapng" --format sc32 | grep -Pv "GET" | grep -Pv "cosmosfurbootsemporium" | grep -Pv "320801014033Z001" | grep -Pv "POST" | grep
-P '[A-Za-z0-9+/]{16,}'
INFO: floss: extracting static strings
finding decoding function features: 100%| | 1/1 [00:00<?, ? functions/s, skipped 0 library functions]
INFO: floss.stackstrings: extracting stackstrings from 1 functions
extracting stackstrings: 100%| | 1/1 [00:00<?, ? functions/s]
INFO: floss.tightstrings: extracting tightstrings from 0 functions...
extracting tightstrings: 0 functions [00:00, ? functions/s]
INFO: floss.string_decoder: decoding strings
decoding strings: 100%| | 1/1 [00:00<?, ? functions/s]
INFO: floss: finished execution after 4.61 seconds
INFO: floss: rendering results
>_9j_4AAQSkZJRgABAQAASABIAAD_4QjmRXhpZgAATU0AKgAAAAgADQEPAAIAAA
>_9j_4AAQSkZJRgABAQAASABIAAD_4QjmRXhpZgAATU0AKgAAAAgADQEPAAIAAA
 EEEFFDELFEEPFACNFAFFEBEGEDEBENCA
 FHEPFCELEHFCEPFFFACACACACACACABN
>AGAAAAqgEQAAIAAAAJAAAAsAESAAMAAAABAAEAAAEaAAUAAAABAAAAugEbAAUA
>AGAAAAqgEQAAIAAAAJAAAAsAESAAMAAAABAAEAAAEaAAUAAAABAAAAugEbAAUA
>AAABAAAAwgEoAAMAAAABAAIAAAExAAIAAAAHAAAAygEyAAIAAAAUAAAA0gFCAA
>AAABAAAAwgEoAAMAAAABAAIAAAExAAIAAAAHAAAAygEyAAIAAAAUAAAA0gFCAA
>QAAAABAAACAAFDAAQAAAABAAACAAITAAMAAAABAAEAAIdpAAQAAAABAAAA5ogl
>QAAAABAAACAAFDAAQAAAABAAACAAITAAMAAAABAAEAAIdpAAQAAAABAAAA5ogl
>AAQAAAABAAAH4gAAAABBcHBsZQBpUGhvbmUgNwAAAAASAAAAEAAABIAAAAAT
>AAQAAAABAAAH4gAAAABBcHBsZQBpUGhvbmUgNwAAAAASAAAAEAAABIAAAAAT
>E0LjAuMQAAMjAyMDoxMDoxMCAxMjowNTozNwAAI4KaAAUAAAABAAACkIKdAAUA
>E0LjAuMQAAMjAyMDoxMDoxMCAxMjowNTozNwAAI4KaAAUAAAABAAACkIKdAAUA
>AAABAAACmIgiAAMAAAABAAIAAIgnAAMAAAABAGQAAJAAAAcAAAAEMDIzMZADAA
>AAABAAACmIgiAAMAAAABAAIAAIgnAAMAAAABAGQAAJAAAAcAAAAEMDIzMZADAA
>IAAAAUAAACoJAEAAIAAAAUAAACtJAQAAIAAAAHAAACyJARAAIAAAAHAAAC0JAS
>IAAAAUAAACoJAEAAIAAAAUAAACtJAQAAIAAAAHAAACyJARAAIAAAAHAAAC0JAS
>AAIAAAAHAAAC2JEBAAcAAAAEAOIDAJIBAAoAAAABAAAC4JICAAUAAAABAAAC6J
```

Then I moved this output to "CyberChef" in order to decode that data.



As you see may see from the screenshot below, many details in here are unreadable, but since we suspect the malware exfiltrates the "cosmo.jpeg" file data, I searched for keywords such as "Apple".



As we can see, we found a string that contains the "Apple" vendor name, the phone 's version (iPhone 7), the kind of camera that has been used, the word "Exif" that relate to Image file format, and also the time frame mentioned in the file details.

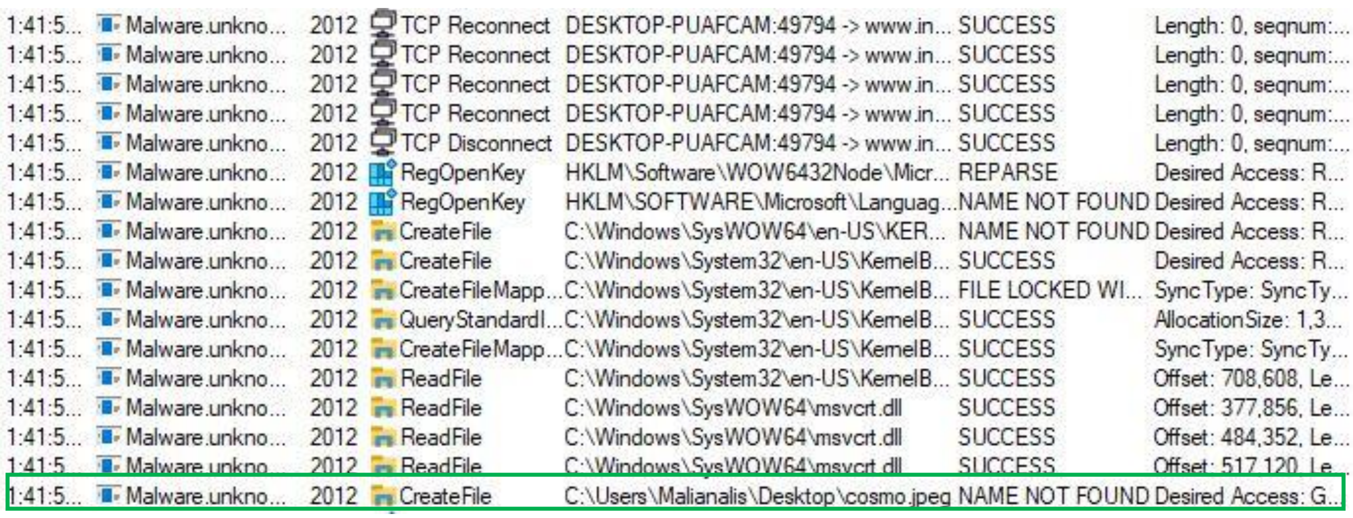
At this point, we have stronger indicators that this malware is taking the data saved in our cosmo.jpeg file.

The next step I wanted to do is to try and trick the malware: first deleting the file from Its place, and in my second check, just change its name.

Staring first with deleting the file:

Since we know this malware looks for "cosmo.jpeg" immediately after unavailable internet connection, I decided to delete "cosmo.jpeg" just before turning off the internet.

As we can see from the screenshot below, the malware works as intended and searches for the file immediately. but since this file doesn't exist, it fails, and we get the error "NAME NOT FOUND".



1:41:5...	Malware.unkno...	2012	TCP Reconnect	DESKTOP-PUAFCAM:49794 -> www.in...	SUCCESS	Length: 0, seqnum:...
1:41:5...	Malware.unkno...	2012	TCP Reconnect	DESKTOP-PUAFCAM:49794 -> www.in...	SUCCESS	Length: 0, seqnum:...
1:41:5...	Malware.unkno...	2012	TCP Reconnect	DESKTOP-PUAFCAM:49794 -> www.in...	SUCCESS	Length: 0, seqnum:...
1:41:5...	Malware.unkno...	2012	TCP Reconnect	DESKTOP-PUAFCAM:49794 -> www.in...	SUCCESS	Length: 0, seqnum:...
1:41:5...	Malware.unkno...	2012	TCP Disconnect	DESKTOP-PUAFCAM:49794 -> www.in...	SUCCESS	Length: 0, seqnum:...
1:41:5...	Malware.unkno...	2012	RegOpenKey	HKLM\Software\WOW6432Node\Micr...	REPARSE	Desired Access: R...
1:41:5...	Malware.unkno...	2012	RegOpenKey	HKLM\SOFTWARE\Microsoft\Languag...	NAME NOT FOUND	Desired Access: R...
1:41:5...	Malware.unkno...	2012	CreateFile	C:\Windows\SysWOW64\en-US\KER...	NAME NOT FOUND	Desired Access: R...
1:41:5...	Malware.unkno...	2012	CreateFile	C:\Windows\System32\en-US\KernelB...	SUCCESS	Desired Access: R...
1:41:5...	Malware.unkno...	2012	CreateFileMapp...	C:\Windows\System32\en-US\KernelB...	FILE LOCKED WI...	SyncType: SyncTy...
1:41:5...	Malware.unkno...	2012	QueryStandardI...	C:\Windows\System32\en-US\KernelB...	SUCCESS	AllocationSize: 1,3...
1:41:5...	Malware.unkno...	2012	CreateFileMapp...	C:\Windows\System32\en-US\KernelB...	SUCCESS	SyncType: SyncTy...
1:41:5...	Malware.unkno...	2012	ReadFile	C:\Windows\System32\en-US\KernelB...	SUCCESS	Offset: 708,608, Le...
1:41:5...	Malware.unkno...	2012	ReadFile	C:\Windows\SysWOW64\msvcrt.dll	SUCCESS	Offset: 377,856, Le...
1:41:5...	Malware.unkno...	2012	ReadFile	C:\Windows\SysWOW64\msvcrt.dll	SUCCESS	Offset: 484,352, Le...
1:41:5...	Malware.unkno...	2012	ReadFile	C:\Windows\SysWOW64\msvcrt.dll	SUCCESS	Offset: 517,120, Le...
1:41:5...	Malware.unkno...	2012	CreateFile	C:\Users\Malianalis\Desktop\cosmo.jpeg	NAME NOT FOUND	Desired Access: G...

When retrieving back the internet after that failure, no further actions made by the malware seems to have taken place, suggesting deleting the file will prevent further execution by the malware.

Our next examination is in a case scenario in which the file comes under a different name.

In this case, I changed the file name from "cosmo.jpeg" to "Cosmo.jpeg".

As we can see from the example below, changing the name in that way didn't interfere the malware process and the execution worked as expected.

2:50:2...	Malware.unkno...	1028	CreateFile	C:\Users\Malianalis\Desktop\Cosmo.jpeg	SUCCESS	Desired Access: G...
2:50:2...	Malware.unkno...	1028	QueryStandardI...	C:\Users\Malianalis\Desktop\Cosmo.jpeg	SUCCESS	AllocationSize: 1,7...
2:50:2...	Malware.unkno...	1028	ReadFile	C:\Users\Malianalis\Desktop\Cosmo.jpeg	SUCCESS	Offset: 0, Length: 1...
2:50:2...	Malware.unkno...	1028	ReadFile	C:\Users\Malianalis\Desktop\Cosmo.jpeg	SUCCESS	Offset: 4,096, Leng...
2:50:2...	Malware.unkno...	1028	ReadFile	C:\Users\Malianalis\Desktop\Cosmo.jpeg	SUCCESS	Offset: 1,753,088, ...
2:50:2...	Malware.unkno...	1028	ReadFile	C:\Users\Malianalis\Desktop\Cosmo.jpeg	END OF FILE	Offset: 1,754,626, ...
2:50:2...	Malware.unkno...	1028	ReadFile	C:\Windows\SysWOW64\msvcrt.dll	SUCCESS	Offset: 267,264, Le...
2:50:2...	Malware.unkno...	1028	CloseFile	C:\Users\Malianalis\Desktop\Cosmo.jpeg	SUCCESS	

2:58:0...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54093 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:1...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54094 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:1...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54094 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:1...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54095 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:1...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54095 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:1...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54096 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:1...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54096 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:2...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54097 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:2...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54097 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:2...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54098 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:2...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54098 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:2...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54099 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:2...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54099 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:3...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54100 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:3...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54100 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:3...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54101 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:3...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54101 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:3...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54102 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:3...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54102 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:3...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54103 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:3...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54103 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:4...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54104 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:4...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54104 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:4...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54105 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:4...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54105 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:4...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54106 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:4...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54106 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:5...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54107 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:5...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54107 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:5...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54108 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:5...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54108 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:5...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54109 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:5...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54109 -> www.in...	SUCCESS	Length: 142, seqn...
2:59:0...	Malware.unkno...	1028	UDP Send	DESKTOP-PUAFCAM:54110 -> www.in...	SUCCESS	Length: 109, seqn...
2:59:0...	Malware.unkno...	1028	UDP Receive	DESKTOP-PUAFCAM:54110 -> www.in...	SUCCESS	Length: 142, seqn...

```
2776 2043.801888   10.0.0.4   10.0.0.3   DNS   184 Standard query response 0x1738 TXT KSo1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoKDhIWGh4.cosmosfurbootsemporium.local TXT
2777 2046.809406   10.0.0.3   10.0.0.4   DNS   151 Standard query 0xcb9f TXT iJipKTlJWwl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ.cosmosfurbootsemporium.local
2778 2046.817098   10.0.0.4   10.0.0.3   DNS   184 Standard query response 0xcb9f TXT iJipKTlJWwl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ.cosmosfurbootsemporium.local TXT
2779 2049.825539   10.0.0.3   10.0.0.4   DNS   151 Standard query 0x2e62 TXT 2uLj5OXm5-jp6vLz9PX29_j5-v_bAEMAAQEBAQEBAgEBAgMCAgIDBAMDAwMEBg.cosmosfurbootsemporium.local
2780 2049.833052   10.0.0.4   10.0.0.3   DNS   151 Standard query response 0x2e62 No such name TXT 2uLj5OXm5-jp6vLz9PX29_j5-v_bAEMAAQEBAQEBAgEBAgMCAgIDBAMDAwMEBg.cosmosfurbootsemporium.local
2783 2052.859182   10.0.0.3   10.0.0.4   DNS   151 Standard query 0xefe7 TXT QEBAQEBgcGBgYGBgYHBwcHBwcHBwgICAgICAkJCQkJCwsLCwsLCwsLC__bAEMB.cosmosfurbootsemporium.local
2784 2052.864925   10.0.0.4   10.0.0.3   DNS   151 Standard query response 0xefe7 No such name TXT QEBAQEBgcGBgYGBgYHBwcHBwcHBwgICAgICAkJCQkJCwsLCwsLCwsLC__bAEMB.cosmosfurbootsemporium.local
2785 2055.872227   10.0.0.3   10.0.0.4   DNS   151 Standard query 0xc482 TXT AgICAwMDBQMDBQsIBggLCwsLCwsLCwsLCwsLCwsLCwsLCwsLCwsLCw.cosmosfurbootsemporium.local
2786 2055.878938   10.0.0.4   10.0.0.3   DNS   184 Standard query response 0xc482 TXT AgICAwMDBQMDBQsIBggLCwsLCwsLCwsLCwsLCwsLCwsLCwsLCwsLCw.cosmosfurbootsemporium.local TXT
2789 2058.888627   10.0.0.3   10.0.0.4   DNS   151 Standard query 0x620d TXT sLCwsLCwsLCwsLCwsLC__dAAQAvf_aAAwDAQACEQMRAD8A_rnQqi_MOapm.cosmosfurbootsemporium.local
2790 2058.895343   10.0.0.4   10.0.0.3   DNS   151 Standard query response 0x620d No such name TXT sLCwsLCwsLCwsLCwsLCwsLC__dAAQAvf_aAAwDAQACEQMRAD8A_rnQqi_MOapm.cosmosfurbootsemporium.local
2791 2061.919896   10.0.0.3   10.0.0.4   DNS   151 Standard query 0x97e7 TXT GYNvXoam8qUQ75aZGzmIljwK-ZXwxPXivdQ2VVdQO4qYOsGHK7gKit0j2mcnNS.cosmosfurbootsemporium.local
2792 2061.928035   10.0.0.4   10.0.0.3   DNS   184 Standard query response 0x97e7 TXT GYNvXoam8qUQ75aZGzmIljwK-ZXwxPXivdQ2VVdQO4qYOsGHK7gKit0j2mcnNS.cosmosfurbootsemporium.local TXT
2793 2064.934133   10.0.0.3   10.0.0.4   DNS   151 Standard query 0xe031 TXT -UzxE5wDUmzRFdTrOyyIKYcRybk6VYhigkjZXPzDpUAj2JmQ4zQIaGVzuk6U_C.cosmosfurbootsemporium.local
2794 2064.937381   10.0.0.4   10.0.0.3   DNS   151 Standard query response 0xe031 No such name TXT -UzxE5wDUmzRFdTrOyyIKYcRybk6VYhigkjZXPzDpUAj2JmQ4zQIaGVzuk6U_C.cosmosfurbootsemporium.local
2795 2067.951056   10.0.0.3   10.0.0.4   DNS   151 Standard query 0xc09f TXT RyAxr1puYlOwNnNKjCR_LmbAHpQASlkUhlzmktPLJ2sNuakkQFwFbiopoDA4cn.cosmosfurbootsemporium.local
2796 2067.956282   10.0.0.4   10.0.0.3   DNS   151 Standard query response 0xc09f No such name TXT RyAxr1puYlOwNnNKjCR_LmbAHpQASlkUhlzmktPLJ2sNuakkQFwFbiopoDA4cn.cosmosfurbootsemporium.local
2797 2070.966225   10.0.0.3   10.0.0.4   DNS   151 Standard query 0xcc9a TXT NACTwjz8x01S4fGeaW3TdIXZqiKP5xdTuFAETmUvmQYAppeMSBkIz3pk4dweOB.cosmosfurbootsemporium.local
2798 2070.972892   10.0.0.4   10.0.0.3   DNS   184 Standard query response 0xcc9a TXT NACTwjz8x01S4fGeaW3TdIXZqiKP5xdTuFAETmUvmQYAppeMSBkIz3pk4dweOB.cosmosfurbootsemporium.local TXT
2799 2073.996992   10.0.0.3   10.0.0.4   DNS   151 Standard query 0x7901 TXT TYootoZQQaAIri8jVjEQcnvUaIExIzYBp9zaruGOtNWe0gHSqAJWkUZUHOahXy.cosmosfurbootsemporium.local
2800 2074.003282   10.0.0.4   10.0.0.3   DNS   184 Standard query response 0x7901 TXT TYootoZQQaAIri8jVjEQcnvUaIExIzYBp9zaruGOtNWe0gHSqAJWkUZUHOahXy.cosmosfurbootsemporium.local TXT
2801 2077.014215   10.0.0.3   10.0.0.4   DNS   151 Standard query 0x49f7 TXT 4nyg5pRLGzMqrgimxMZGJkFA-pLJAGfzyw3VRlS6Lgo_PpU80WXDA4FSNZucTo.cosmosfurbootsemporium.local
2802 2077.020162   10.0.0.4   10.0.0.3   DNS   151 Standard query response 0x49f7 No such name TXT 4nyg5pRLGzMqrgimxMZGJkFA-pLJAGfzyw3VRlS6Lgo_PpU80WXDA4FSNZucTo.cosmosfurbootsemporium.local
2803 2080.028304   10.0.0.3   10.0.0.4   DNS   151 Standard query 0x1ffd TXT 3AoAVfNf5LttpFVcwqSGNW5AWwzHP1qr9gtnuAWc_SgGNaDCeaneoYoX3cnINa.cosmosfurbootsemporium.local
2804 2080.034445   10.0.0.4   10.0.0.3   DNS   151 Standard query response 0x1ffd No such name TXT 3AoAVfNf5LttpFVcwqSGNW5AWwzHP1qr9gtnuAWc_SgGNaDCeaneoYoX3cnINa.cosmosfurbootsemporium.local

▸ Frame 2776: 184 bytes on wire (1472 bits), 184 bytes captured (1472 bits) on interface \Device\NPF_{DDB894CC-77CB-4325-AEB7-8C012DF0CC2B}, id 0
▸ Ethernet II, Src: PCSSystemtec_a4:d5:15 (08:00:27:a4:d5:15), Dst: PCSSystemtec_03:ba:c0 (08:00:27:03:ba:c0)
▸ Internet Protocol Version 4, Src: 10.0.0.4, Dst: 10.0.0.3
▸ User Datagram Protocol, Src Port: 53, Dst Port: 54051
▾ Domain Name System (response)
    Transaction ID: 0x1738
  ▸ Flags: 0x8500 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  ▾ Queries
    ▾ KSo1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoKDhIWGh4.cosmosfurbootsemporium.local: type TXT, class IN
        Name: KSo1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoKDhIWGh4.cosmosfurbootsemporium.local
        [Name Length: 91]
        [Label Count: 3]
        Type: TXT (16) (Text strings)
        Class: IN (0x0001)
  ▸ Answers
    [Request In: 2775]
    [Time: 0.006883000 seconds]
```

Then, I checked what happens If the file name is totally different.

I used the name "cosmoCat as an example, and as seen from the screenshot below, we got the same error we were seeing before that the file has not been found.

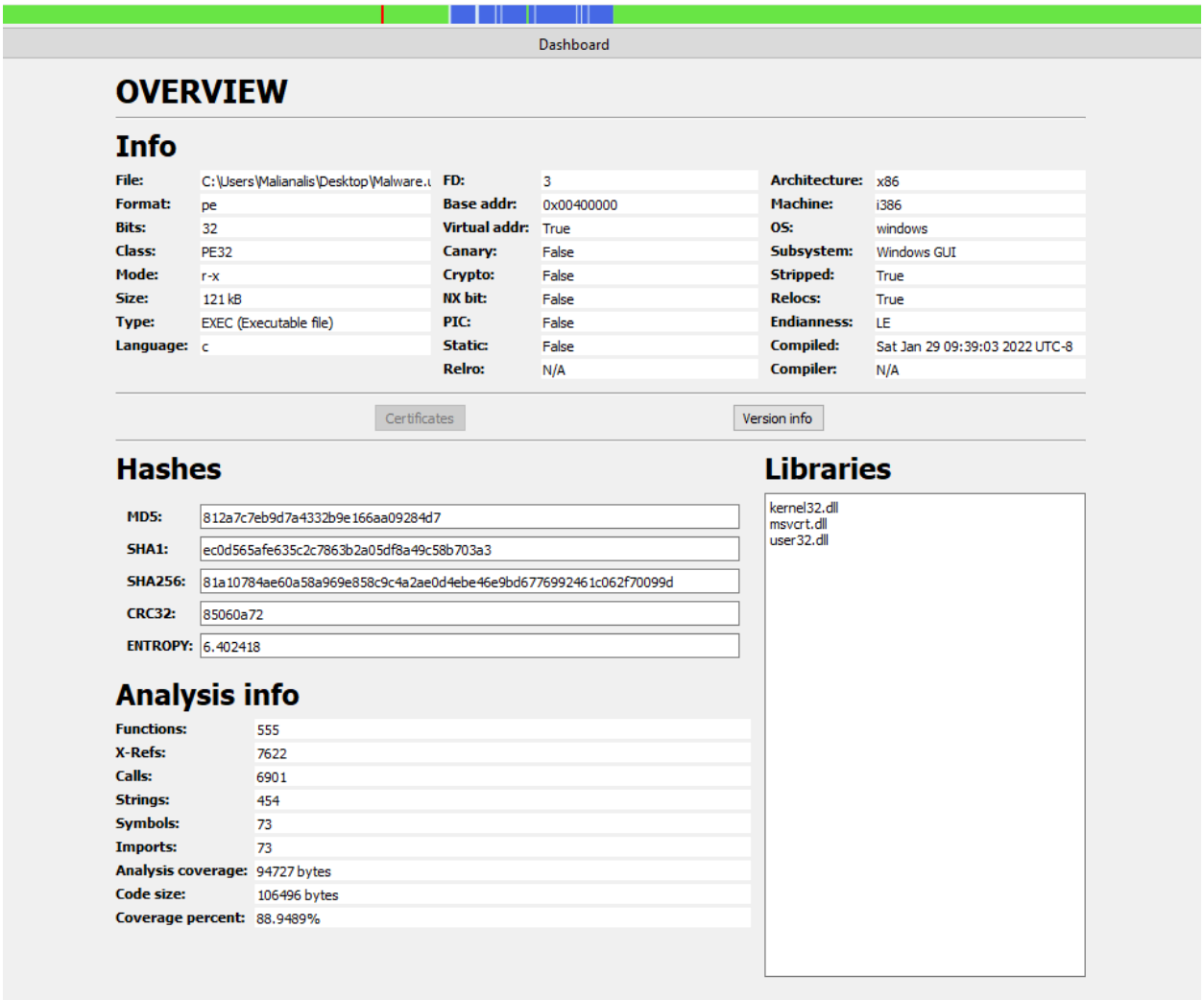
Based on these indicators, we can assume that the malware has few terms in which it launches its further actions. if the file is changed by its capital letters, the malware execution process will work as intended. But if the file will change its name to a totally different one, or either be deleted before the query has been made, the next steps of the malware processes will stop, and the malware will not launch its further exfiltration process.

In our next step of analysis, we are going to investigate the malware program at the assembly level and try to find more details regarding the malware logical flow and maybe to reveal other hidden stuffs the malware is doing while running.

Advanced Static Analysis

In this section we will try to dive deeper into the malware internal processes.

As we approach into this section, we are going to use "Cutter" as our advanced static analysis tool. opening the main page of the program, we get the details of the sample, some of which are already known to us from the first stages of analysis.



From there, we will move forward to the "main" function, because this is the actual place in which the malware detonates its malicious program.

In our case scenario, although some details are available to us, there is not much to be said regarding what we see so far, except the known pattern in which the function works, finally result in a call to unknown function named "fcn.00419440".

```
int main(int argc, char **argv, char **envp);
; var int32_t var_1ch @ stack - 0x1c
; var int32_t var_18h @ stack - 0x18
; var int32_t var_14h @ stack - 0x14
; var int32_t var_ch @ stack - 0xc
; arg char **argv @ stack + 0x4
0x0041aaa0      lea     ecx, [argv]
0x0041aaa4      and     esp, -16
0x0041aaa7      push    dword [ecx - 4]
0x0041aaaa      push    ebp
0x0041aaab      mov     ebp, esp
0x0041aaad      push    ecx
0x0041aaae      sub     esp, 20
0x0041aab1      call    fcn.00419440 ; fcn.00419440
0x0041aab6      mov     eax, dword section..data ; 0x41b000
0x0041aabb      mov     dword [var_1ch], 0
0x0041aac3      mov     dword [var_14h], eax
0x0041aac7      mov     eax, dword [0x422318]
0x0041aacc      mov     dword [var_18h], eax
0x0041aad0      mov     eax, dword [0x42231c]
0x0041aad5      mov     dword [esp], eax
0x0041aad8      call    fcn.00419128 ; fcn.00419128
0x0041aadd      mov     ecx, dword [var_ch]
0x0041aae0      sub     esp, 0x10
0x0041aae3      leave
0x0041aae4      lea     esp, [ecx - 4]
0x0041aae7      ret
0x0041aae8      nop
0x0041aae9      nop
0x0041aaea      nop
0x0041aaeb      nop
0x0041aaec      nop
0x0041aaed      nop
0x0041aaee      nop
0x0041aaef      nop
0x0041aaf0      jmp     0x401500
0x0041aaf5      nop
0x0041aaf6      nop
0x0041aaf7      nop
0x0041aaf8      nop
0x0041aaf9      nop
0x0041aafa      nop
0x0041aafb      nop
0x0041aafc      nop
0x0041aafd      nop
0x0041aafe      nop
0x0041aaff      nop
```

Another thing that could be said regarding the "main" function is the fact it takes 3 arguments (int argc. char **argv, char **envp) and then pushes into the stack few variables, while one of them is the actual argument "char **argv".

```
int main(int argc, char **argv, char **envp);
; var int32_t var_1ch @ stack - 0x1c
; var int32_t var_18h @ stack - 0x18
; var int32_t var_14h @ stack - 0x14
; var int32_t var_ch @ stack - 0xc
; arg char **argv @ stack + 0x4
```

Moving on in this function, what could be a bit suspicious is the presence of "section..data" that is moved into the "eax". Also, the mentioned variables above. are being used in order to call another unknown function- "fcn.00419128".

```
0x0041aab6      mov     eax, dword section..data ; 0x41b000
0x0041aabb      mov     dword [var_1ch], 0
0x0041aac3      mov     dword [var_14h], eax
0x0041aac7      mov     eax, dword [0x422318]
0x0041aacc      mov     dword [var_18h], eax
0x0041aad0      mov     eax, dword [0x42231c]
0x0041aad5      mov     dword [esp], eax
0x0041aad8      call    fcn.00419128 ; fcn.00419128
```

Let's review now the "inside" of the "section..data" instruction.

While we don't know for sure what its purpose we could say the following:

1. This instruction comes in a repetitive way- so we are dealing with an ongoing process.
2. It uses "–rw" (read & write permissions) to deal with the data, what could match the malware behavior when querying our "cosmo.jpeg' file: first reading its content, and then writing this content in a base64 form.

```
;-- section..data:
0x0041b000      or      al, byte [eax] ; [01] -rw- section size 4096 named .data
0x0041b002      add     byte [eax], al
0x0041b004      add     byte [eax], al
0x0041b006      add     byte [eax], al
0x0041b008      add     byte [eax], al
0x0041b00a      add     byte [eax], al
0x0041b00c      add     byte [eax], al
0x0041b00e      add     byte [eax], al
0x0041b010      add     byte [eax], al
0x0041b012      add     byte [eax], al
0x0041b014      add     byte [eax], al
0x0041b016      add     byte [eax], al
0x0041b018      add     byte [eax], al
0x0041b01a      add     byte [eax], al
0x0041b01c      add     byte [eax], al
0x0041b01e      add     byte [eax], al
;-- data.0041b020:
0x0041b020      add     al, 0
0x0041b022      add     byte [eax], al
0x0041b024      add     byte [eax], al
0x0041b026      add     byte [eax], al
0x0041b028      sbb     al, 2
0x0041b02a      add     byte [eax], al
0x0041b02c      add     byte [eax], al
0x0041b02e      add     byte [eax], al
0x0041b030      add     byte [eax], al
0x0041b032      add     byte [eax], al
0x0041b034      add     byte [eax], al
0x0041b036      add     byte [eax], al
0x0041b038      add     byte [eax], al
0x0041b03a      add     byte [eax], al
0x0041b03c      add     byte [eax], al
0x0041b03e      add     byte [eax], al
;-- data.0041b040:
0x0041b040      bnd     jge 0x41b079
0x0041b043      mov     ch, 0xac    ; 172
0x0041b045      retf
0x0041b046      iretd
0x0041b047      adc     dword [ebp + 0x5f8000ca], edx
0x0041b04d      dec     eax
0x0041b04e      mov     eax, dword [0x367df192]
;-- data.0041b050:
0x0041b050      int1
0x0041b051      jge     0x41b089
0x0041b053      mov     ch, 0xac    ; 172
0x0041b055      retf
```

From there, the next step I did is to move through the code subroutines and found out one subroutine that calls "fwrite" function. as seen from the screenshot below, this subroutine uses the msvcrt.dll library that deals with I/O string manipulation, memory allocation and more. this information adds another layer of knowledge for us since we know the strings of "cosmo.jpeg" are being used in order to exfiltrate the data from the file.

```
size_t sub.msvcrt.dll_fwrite(const void *ptr, size_t size, size_t nitems, FILE *stream);
```

Looking at the subroutine parameters, we can see other few indicators such as "size_t size"- that refers to a size of data to be written, "size_t nt items" that mentions the number of data to be written, and finally the "FILE *stream" parameter, that represent an abstracted form to a sequence of bytes associated with a file.

So all in all, we may suspect that this subroutine is part of how the exfiltration process looks like from the point of view of the program.

In our next step, we will move into the "Strings" section in "Cutter", to look for other important details.

When looking into the "Strings" section, I tried to find any suspicious strings that may be related to remote connectivity.

In the basic static analysis phase, we become aware of a few suspicious ones, so now it's the time to examine some of them. I first started with a string named "socket".

Address	String
0x0041cc1b	socket

Moving to the place in memory this string is invoked, I can see it invoked under different name: "data.0041cc1b", and not by its formal name, probably in order to evade detection.

```
0x0040b439      mov     ecx, dword [0x420508]
0x0040b43f      mov     edx, data.0041cc1b ; 0x41cc1b
0x0040b444      mov     dword [0x42b140], eax
0x0040b449      call    fcn.00406161 ; fcn.00406161
```

Also, just after this string is invoked, another string comes by: "str.closesocket",what may be related to the same evasion technique.

```
0x0040b44e      mov     ecx, dword [0x420508]
0x0040b454      mov     edx, str.closesocket ; 0x41cc22
0x0040b459      mov     dword [0x42b13c], eax
0x0040b45e      call    fcn.00406161 ; fcn.00406161
```

Moving on with this, I managed to find another suspicious string: "connect".

Checking upon the place it invoked in memory retrieve not only the string itself, but also another interesting strings such as "str.FindFirstFileW", "str.FindClose", "str.send".
These strings confirm what we found out in the early stages of analysis: The Malware first search the file, and when it finds that file, and internet connection is active, it sends its data immediately to a remote server.

.

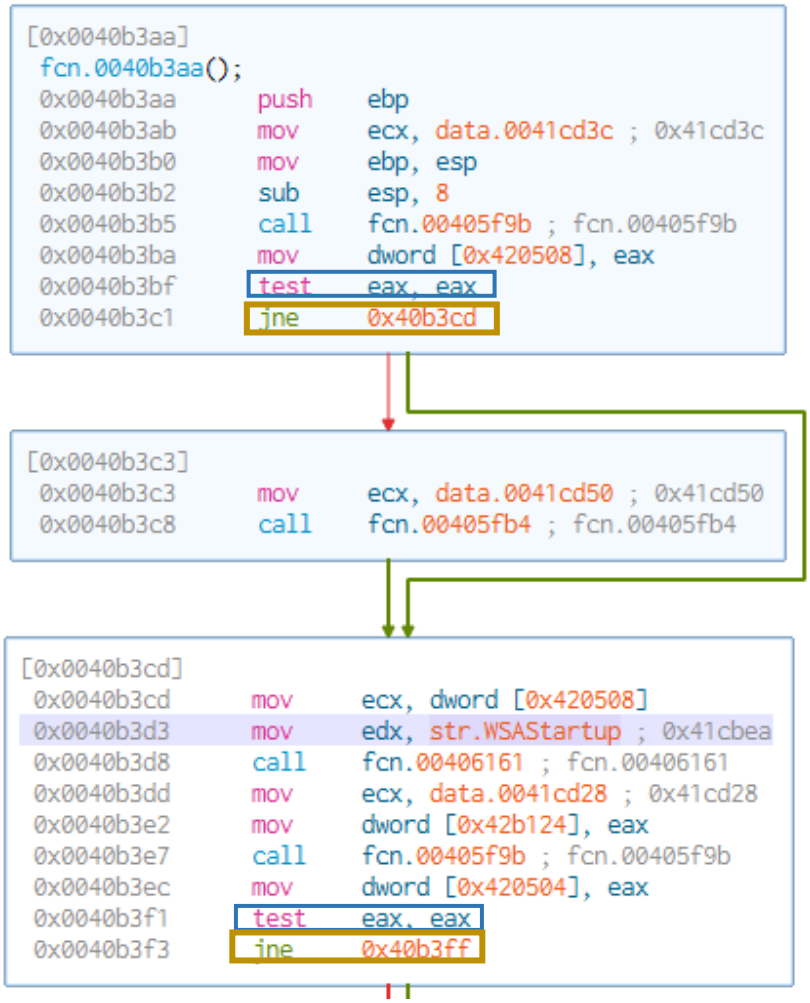
```
0x0040b4a8      mov     edx, str.connect ; 0x41cc50
0x0040b4ad      mov     dword [0x42b144], eax
0x0040b4b2      call    fcn.00406161 ; fcn.00406161
0x0040b4b7      mov     ecx, dword [0x420504]
0x0040b4bd      mov     edx, str.FindFirstFileW ; 0x41cc58
0x0040b4c2      mov     dword [0x42b14c], eax
0x0040b4c7      call    fcn.00406161 ; fcn.00406161
0x0040b4cc      mov     ecx, dword [0x420504]
0x0040b4d2      mov     edx, str.FindClose ; 0x41cc67
0x0040b4d7      mov     dword [0x42b114], eax
0x0040b4dc      call    fcn.00406161 ; fcn.00406161
0x0040b4e1      mov     ecx, dword [0x420508]
0x0040b4e7      mov     edx, str.send ; 0x41cc71
0x0040b4ec      mov     dword [0x42b11c], eax
0x0040b4f1      call    fcn.00406161 ; fcn.00406161
```

Finally, it seems like that all of these strings comes from a "parent" function process named "fcn.0040b3aa", that is responsible for handling both errors and successful execution processes.

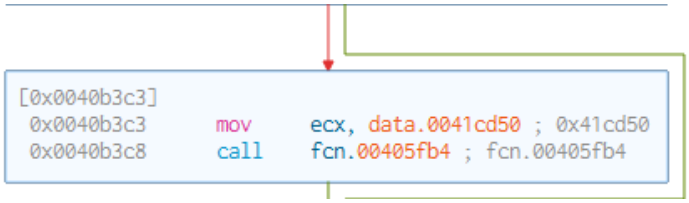
As seen from the screenshot below, there are two points of "jne" (Jump not equal) instructions, that checks the test results of the "test" instruction, and based on these results acts in two different manners.

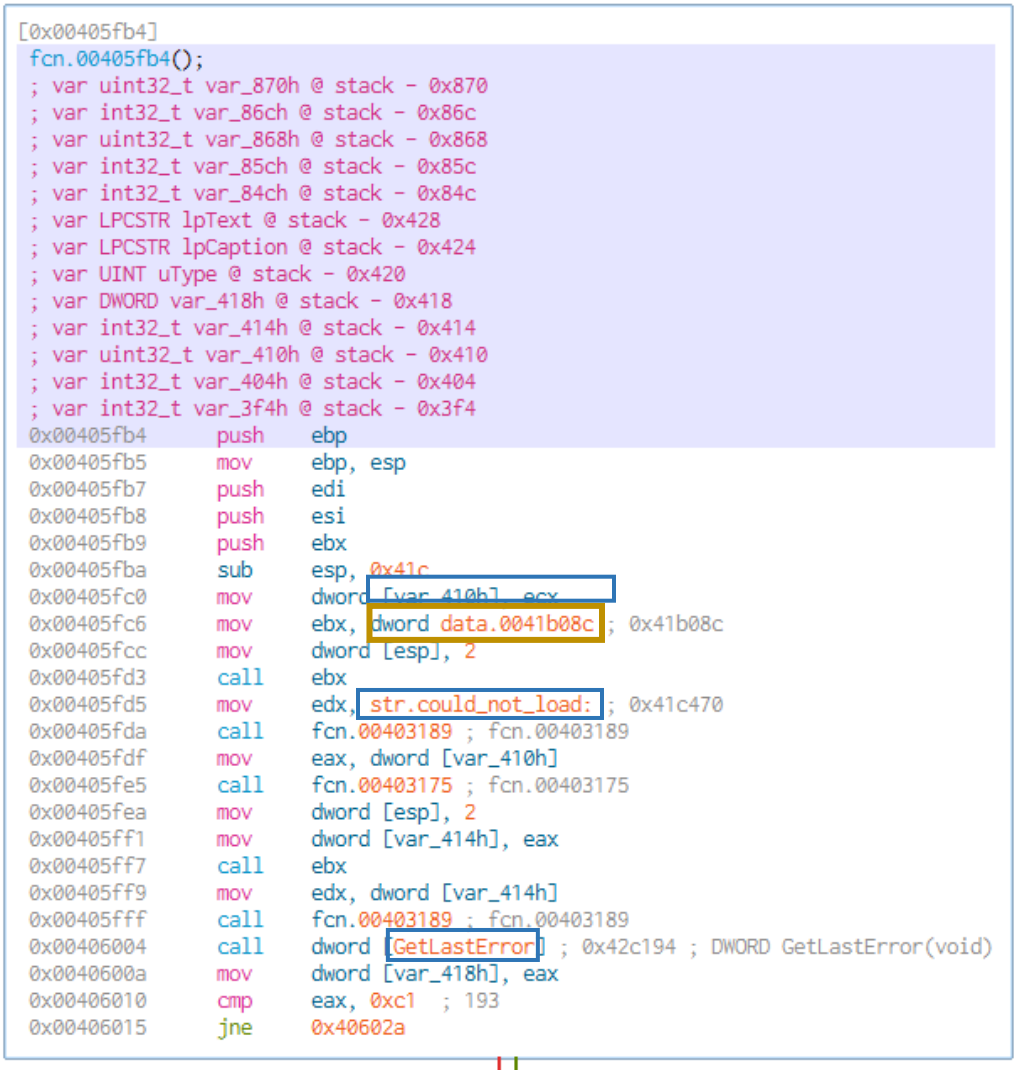
If the values stored on the "eax" will not be equal to each other (ZF Set to 0), the malware will keep running its execution process as intended.

On the contrary, if the "eax" values will be equal to each other (ZF Set to 1) the jump will not be taken, and we'll move to some other place in memory to handle failure execution scenario.



Examining what could happened if the jump is not taken is shown in here: strings such as "str.could_not_load", and "GetLastError" API call are invoked, giving us a clue of what could happen.





As seen before, in case the program works as expected, we can see strings such as "str.connect", "str.FindFirstFileW", "Str.FindClose", and "str.send" are invoked, what matches our previous assumptions regarding the malware actions on our targeted file.

Lastly, we found out the place in memory in which the "str.dnsclient.nim" is being called, followed by another call to "str,sendQuery",suggesting that it could be the place in which the malware queries our first domain: "hxxp:/hey.youup.local".

```
0x00411fe6      mov     dword [var_f0h], 0x31 ; '1' ; 49 ; int32_t arg_ch
0x00411fee      mov     edx, str.TimeoutError ; 0x41dc80
0x00411ff3      mov     ecx, ebx
0x00411ff5      mov     dword [var_f4h], str.dnsclient.nim ; 0x41dc8d ; const char *arg_8h
0x00411ffd      mov     dword [esp], str.sendQuery ; 0x41dc9b ; int32_t arg_4h
0x00412004      call    fcn.004057e9 ; fcn.004057e9
```

So to summarize this stage of analysis, we found out great deal of information regarding the preparation of the malware before execution, and also regarding what could happen in a case of a failure. Finally, we became aware of the kind of process that the malware takes when everything works as expected.

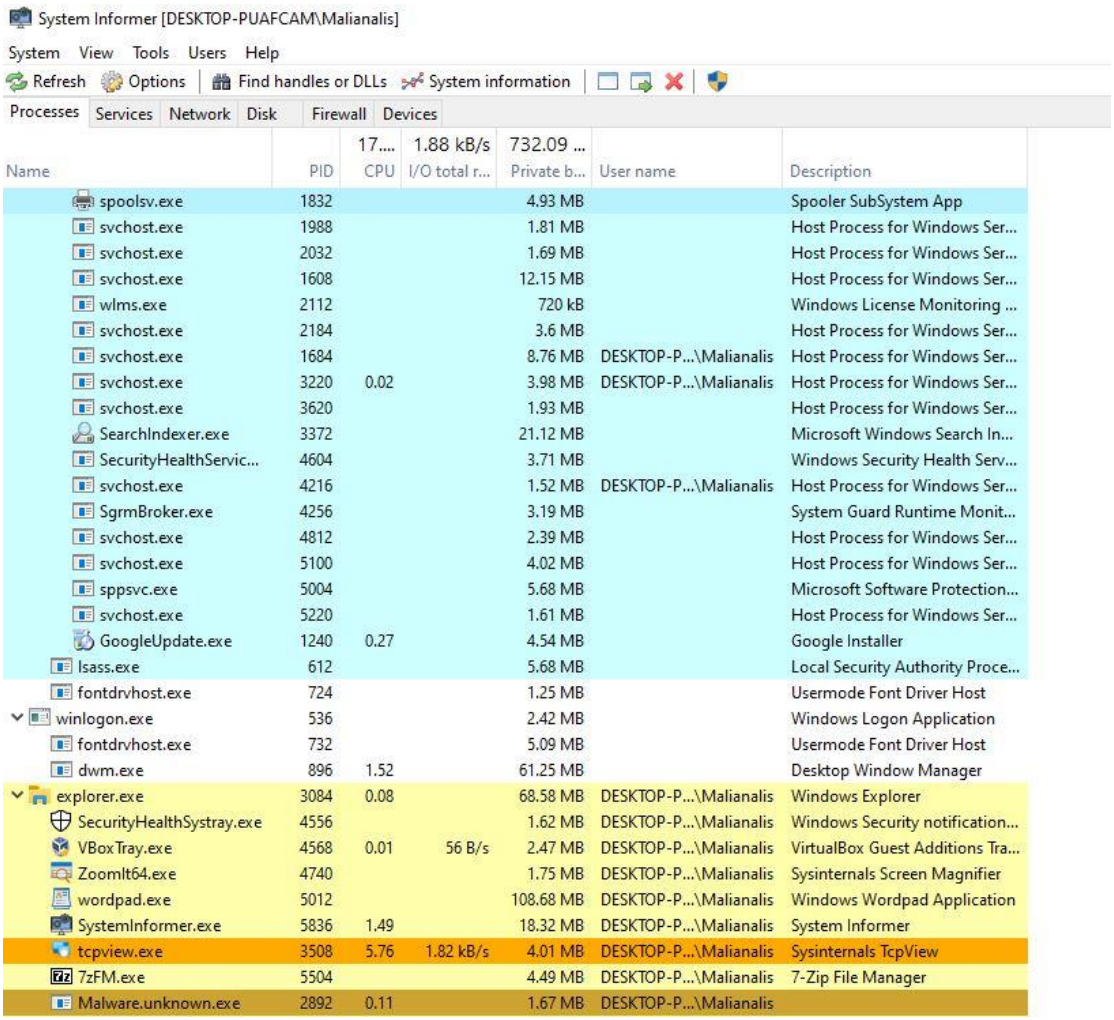
In the next section of analysis, we'll open up a debugger to look if what we found in the advanced static analysis phase matches the overflow of the program.

Advanced Dynamic Analysis

In the start of this section, we will leave for a bit our assembly investigation, and go back to some more basic information regarding the malware.

What we already know at this time is that in ideal situation, from the attacker point of view, the malware will successfully exfiltrate the data from our compromised file to the data exfiltration domain. While we know this is what actually being done, we are still not sure how this process has exactly took place, and how the attacker had gained those privileges to take these actions over the compromised endpoint.

To investigate that, I ran the file again, and at this time, I opened up "Systeminformer" (also known as "Process Hacker") to have another view of the malware processes actions.



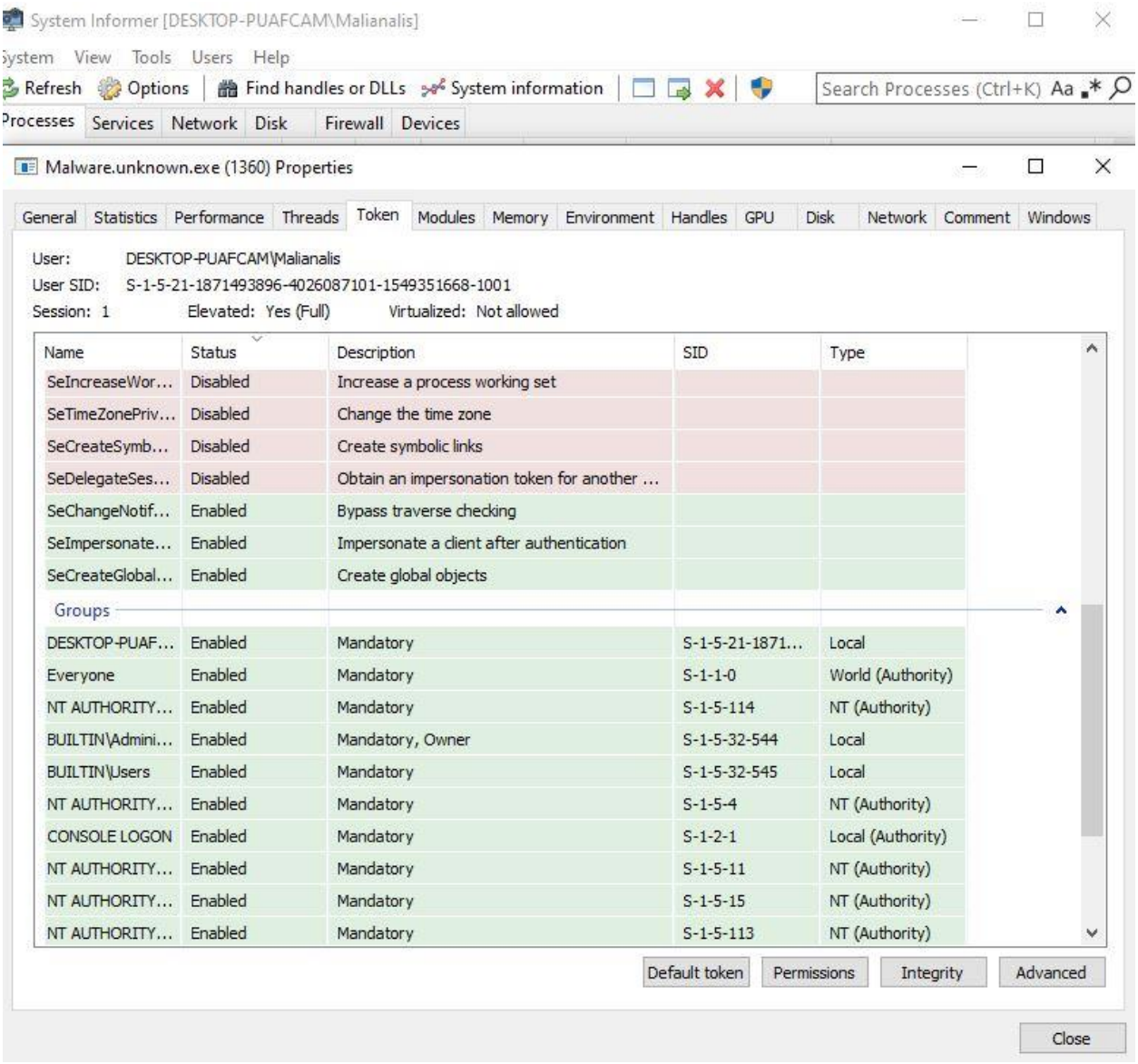
Moving to the Properties section, I came across the Modules the malware use. as we can see, there a more dll libraries that are being used then what we thought, resembling the way in which system changes are being made to prepare for successful execution process.

Name	Base address	Size	Description
Malware.unkn...	**0x400000**	**188 kB**	
apphelp.dll	0x73b50000	636 kB	Application Compatibility Client Library
bcrypt.dll	0x76fe0000	100 kB	Windows Cryptographic Primitives Library (Wow64)
bcryptprimitives...	0x76340000	380 kB	Windows Cryptographic Primitives Library
dnsapi.dll	0x72d50000	576 kB	DNS Client API DLL
FWPUCLNT.DLL	0x72ce0000	356 kB	FWP/IPsec User-Mode API
gdi32.dll	0x76d30000	140 kB	GDI Client DLL
gdi32full.dll	0x76950000	884 kB	GDI Client DLL
imm32.dll	0x76b40000	148 kB	Multi-User Windows IMM32 API Client DLL
IPHLPAPI.DLL	0x74750000	200 kB	IP Helper API
kernel32.dll	0x76720000	960 kB	Windows NT BASE API Client DLL
KernelBase.dll	0x758f0000	2.1 MB	Windows NT BASE API Client DLL
locale.nls	0x630000	804 kB	
msvcp_win.dll	0x752b0000	492 kB	Microsoft® C Runtime Library
msvcrt.dll	0x76280000	764 kB	Windows NT CRT DLL
mswsock.dll	0x72de0000	328 kB	Microsoft Windows Sockets 2.0 Service Provider
nsi.dll	0x76d20000	28 kB	NSI User-mode interface DLL
ntdll.dll	0x77100000	1.64 MB	NT Layer DLL
ntdll.dll	0x7ffe59e7...	1.97 MB	NT Layer DLL
rasadhlp.dll	0x72d40000	32 kB	Remote Access AutoDial Helper
rpcrt4.dll	0x76a30000	760 kB	Remote Procedure Call Runtime
SortDefault.nls	0x2250000	3.22 MB	
ucrtbase.dll	0x763a0000	1.12 MB	Microsoft® C Runtime Library
user32.dll	0x74ff0000	1.61 MB	Multi-User Windows USER API Client DLL
win32u.dll	0x76c00000	96 kB	Win32u
wow64.dll	0x7ffe58c0...	356 kB	Win32 Emulation on NT64
wow64cpu.dll	0x770f0000	40 kB	AMD64 Wow64 CPU
wow64win.dll	0x7ffe5848...	524 kB	Wow64 Console and Win32 API Logging
ws2_32.dll	0x76880000	396 kB	Windows Socket 2.0 32-Bit DLL

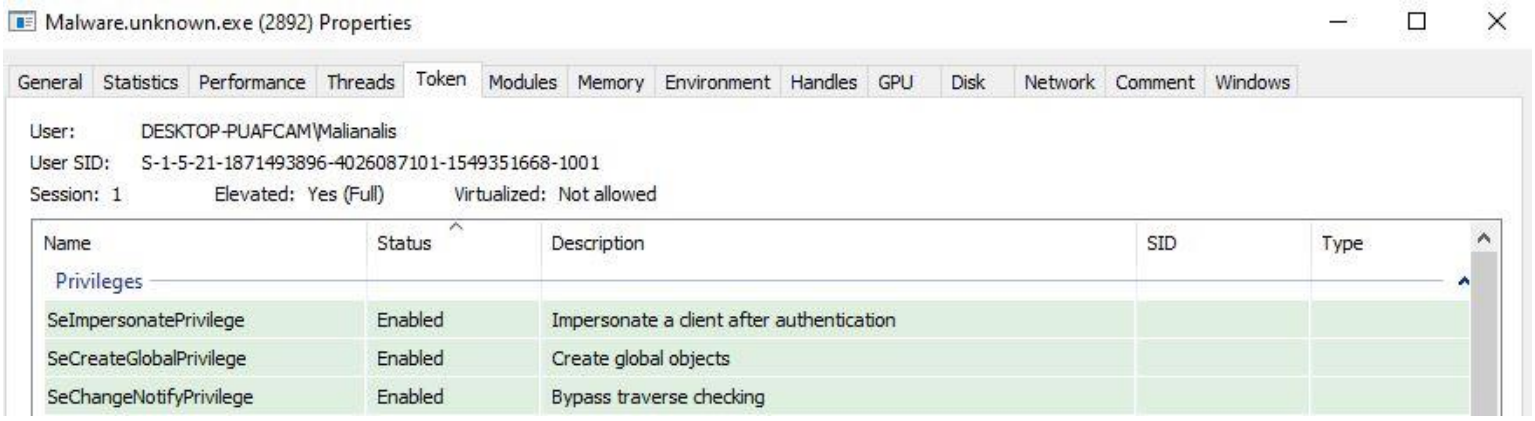
Moving on to the "Token" section, I was able to identify the malware process privileges.

I became aware that there are differences in the malware privileges when running in elevated mode (Run as admin) versus running the same file without admin privileges.

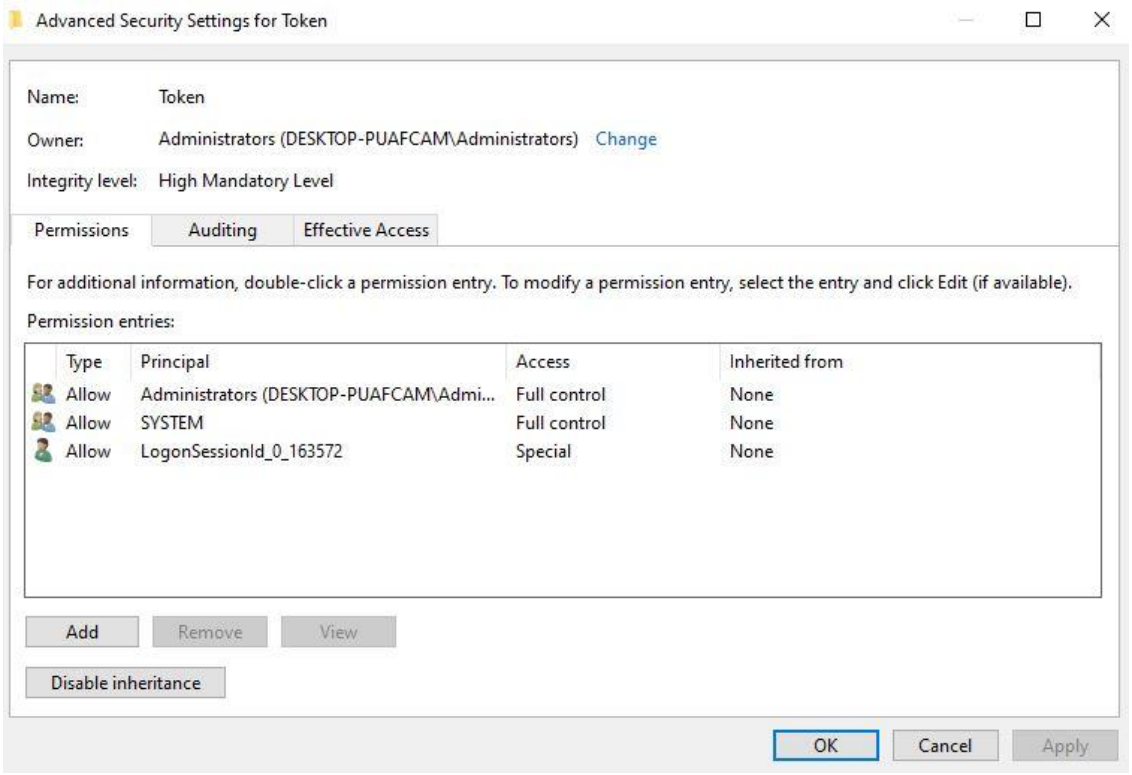
In an elevated mode, we can see the malware process has a lot of privileges, including those that are enabled only for admins, such as "NT Authority" privileges.



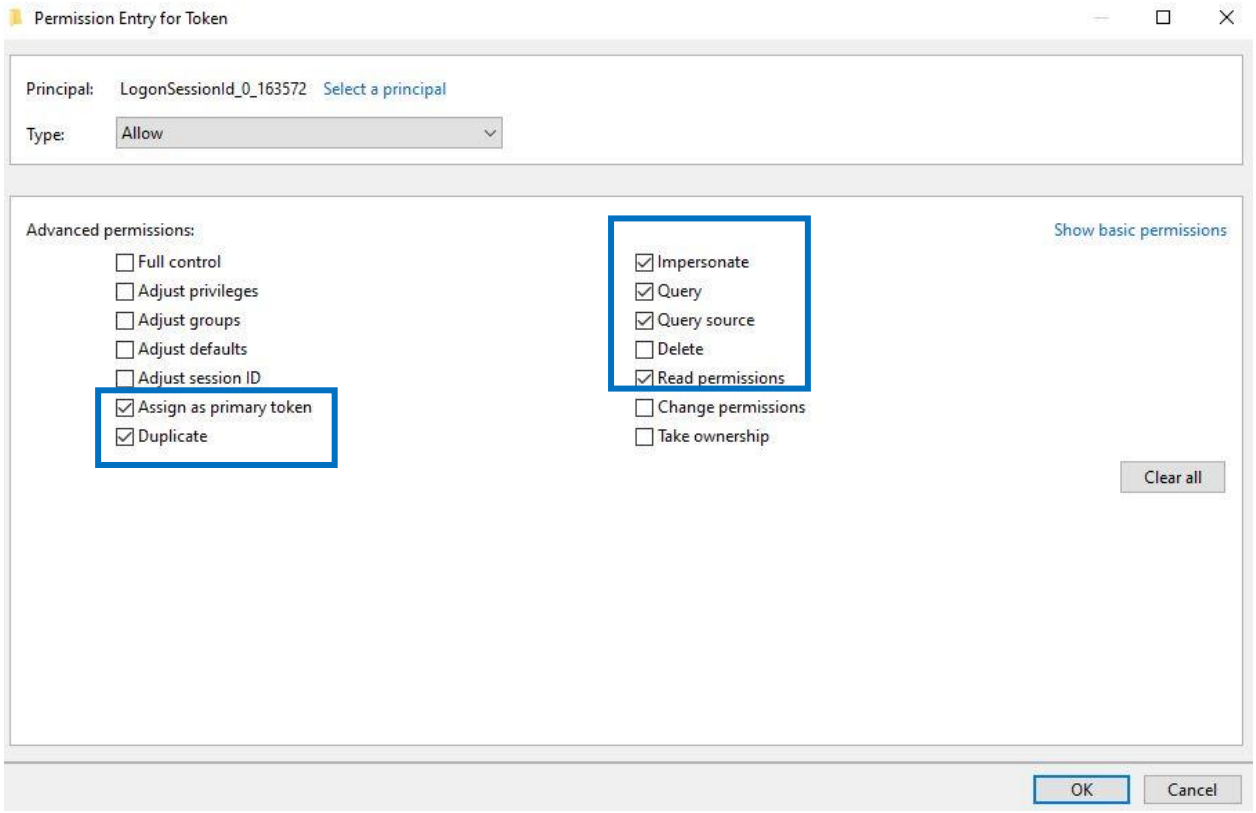
One Particular privilege that I found interesting in that regard is the "Impersonate a client after authentication" permission. this allows the user to gain unauthorized access while impersonating to the endpoint's client.



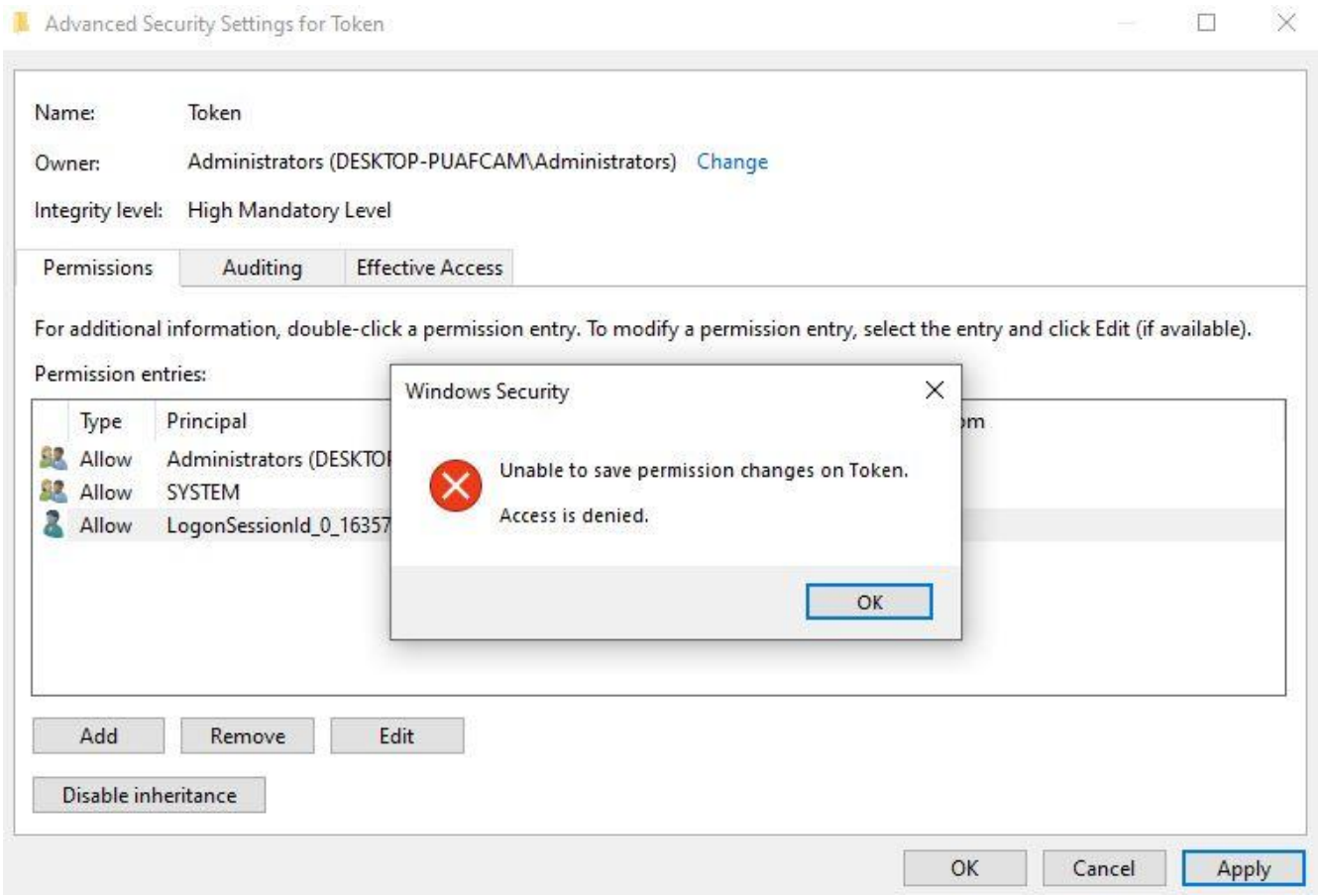
When opening the "Permissions" tab we found out this is actually being done through the creation of a user named "LogonSessionid_0_163572", that has "Special" Access permissions.



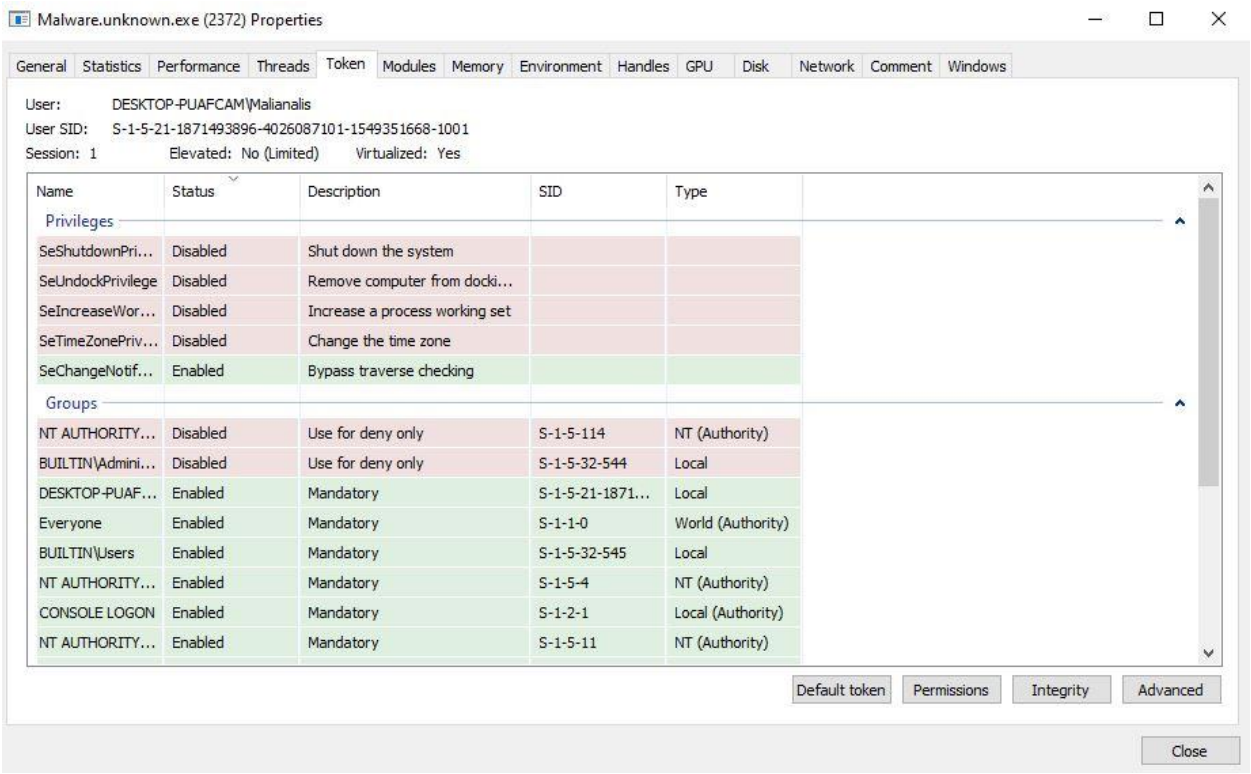
Opening the user "Special" permissions, reveals suspicious checkboxes that are active, such as "Query" and "Impersonate", what may lead us to think that this user creation has set the tone for the malware later execution processes.



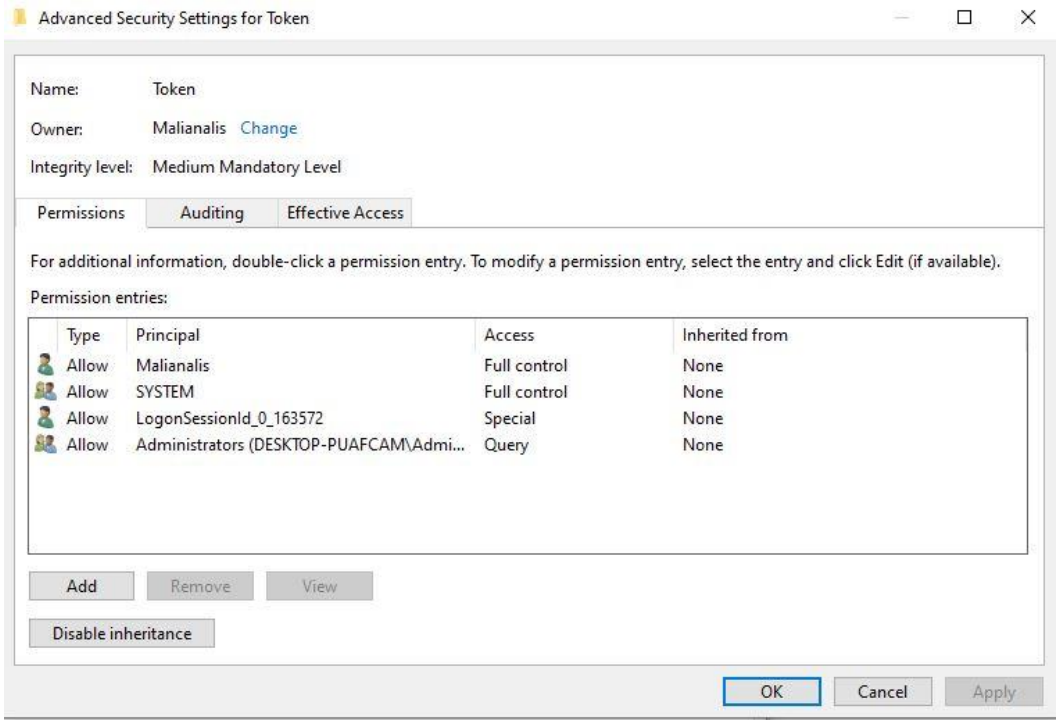
When trying to remove some of those permissions, we get an error message says we don't have the right privileges to do so. This makes this logon session user invulnerable for deletion or permission changes, and from the attacker point of view, allows him to retain sustainable connection between the compromised endpoint to his remote servers.

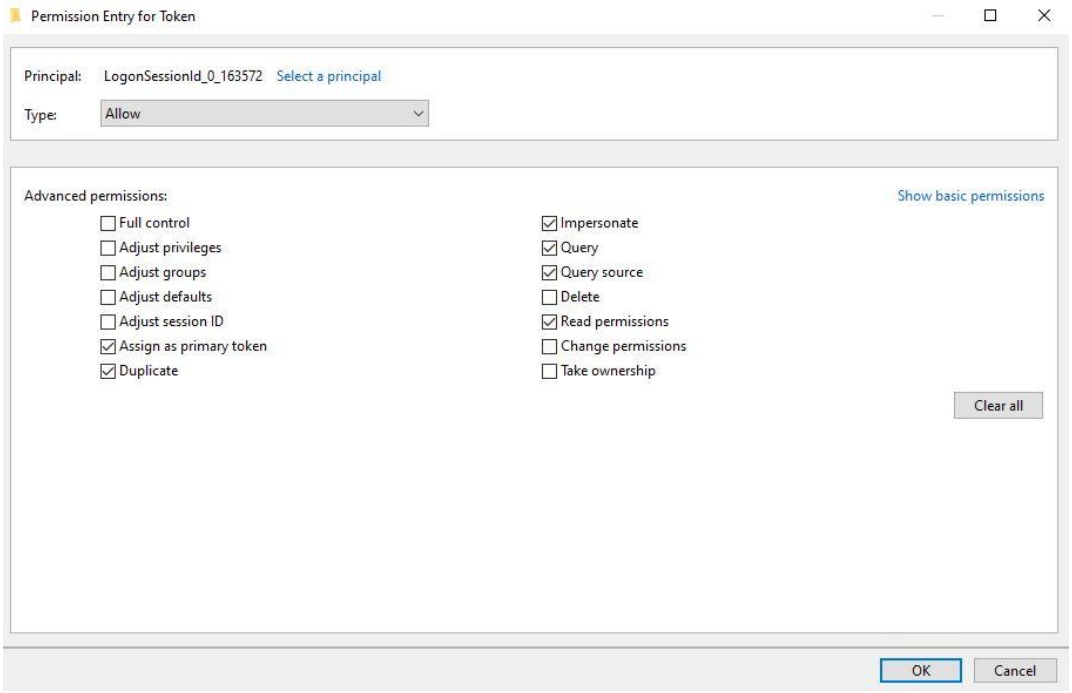


When running the sample without elevated privileges we can see fewer privileges that are enabled by the malware.

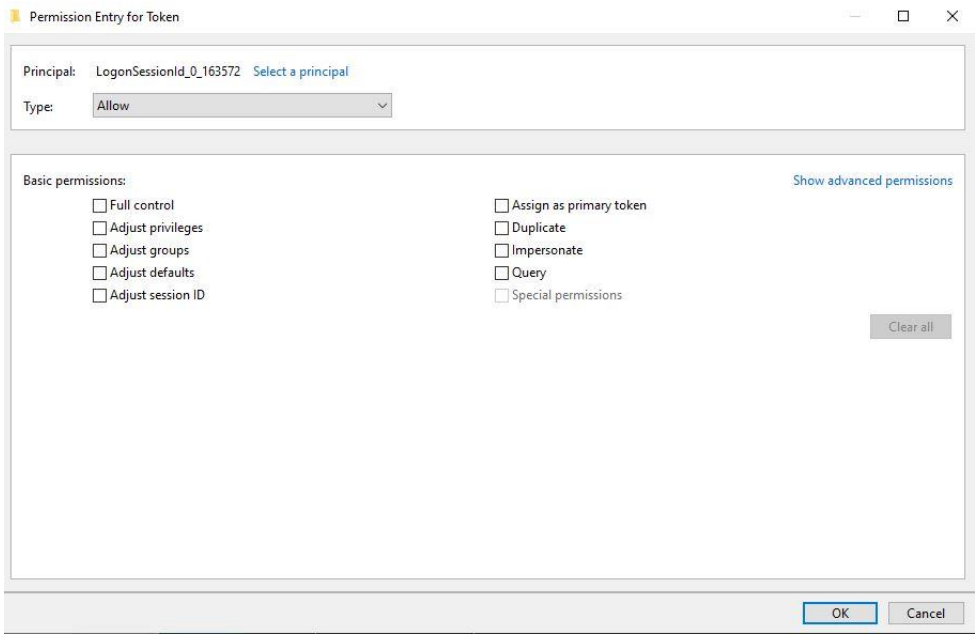


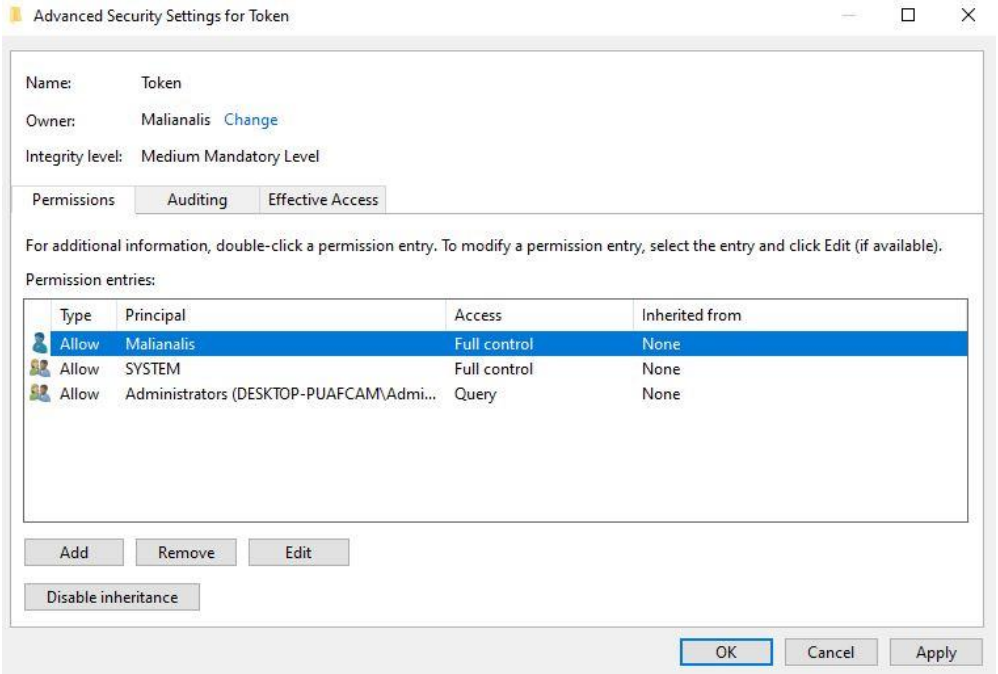
looking at the permission tab, we can see the same user creation, and also see his permissions has remained exactly the same as they were in the elevated mode.



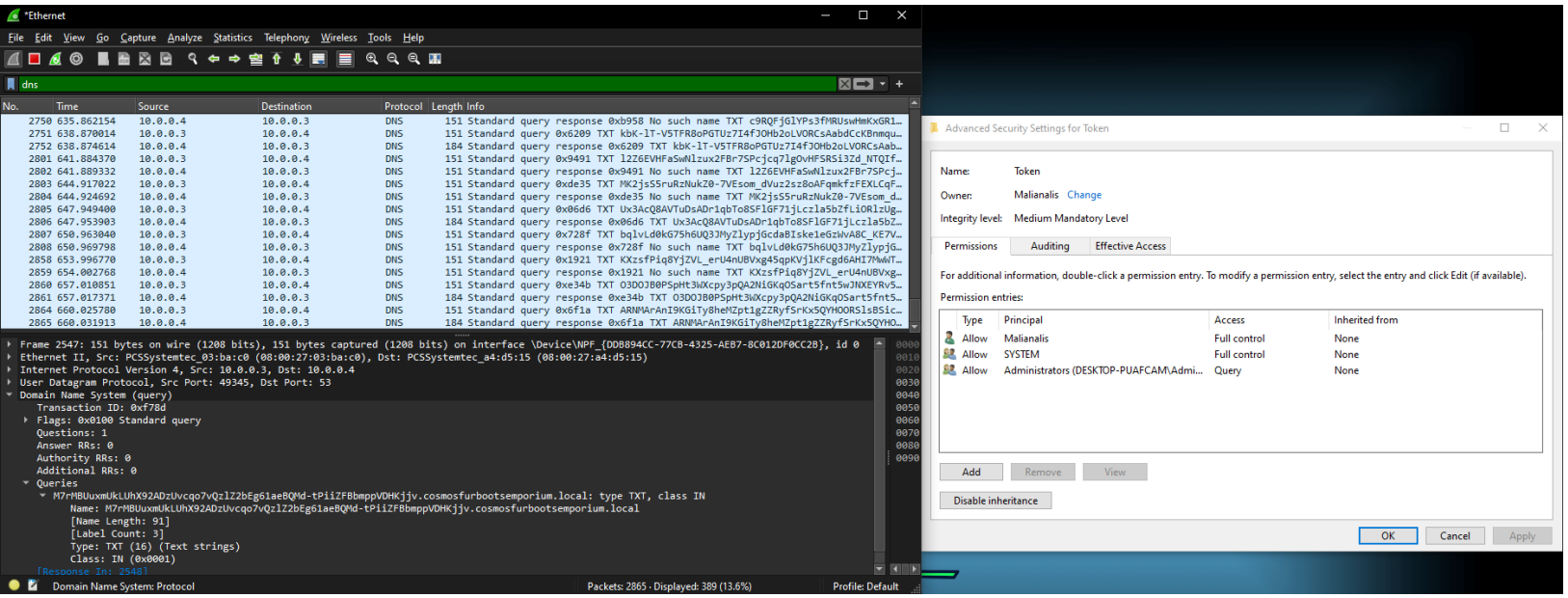


The only difference is that now we can remove or unchecked the current permissions, and even deleting the user from the users list.

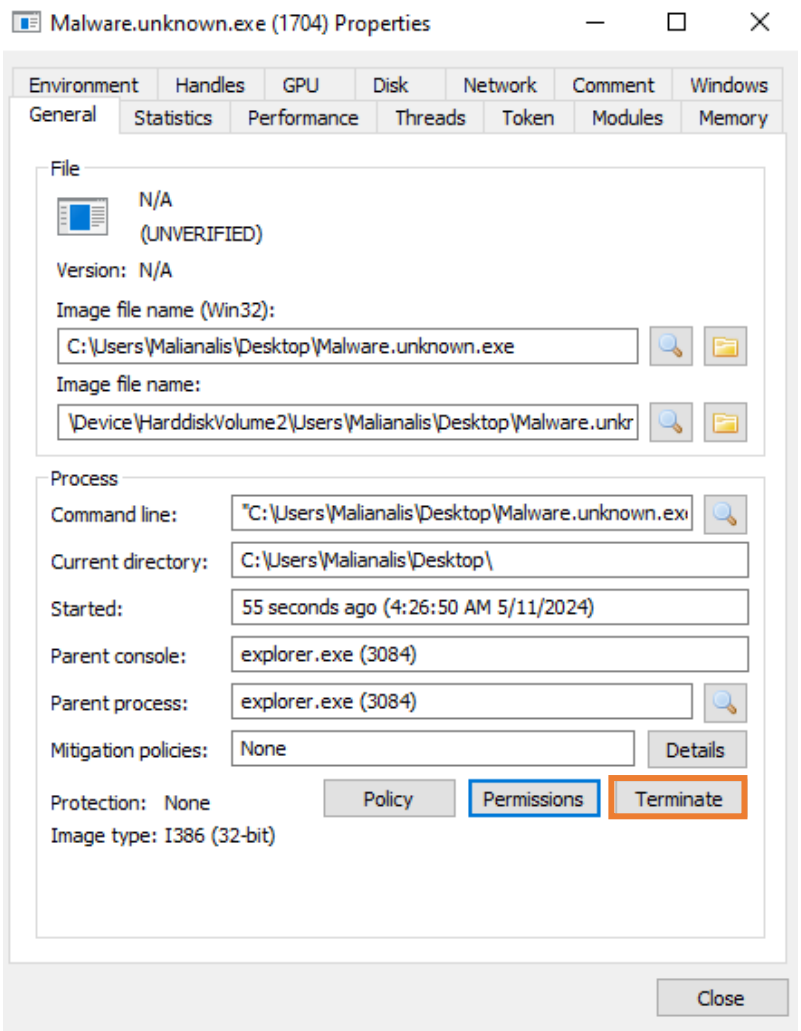




But what I came up with, Is that even doing so will not cause the malware to stop its processes. In fact, even without those permissions the malware kept doing what it is intended to do, as seen from the screenshot below.



These findings suggest that some communication information is still stored on the system, making sure that even if the user has been exposed during analysis, the malware will still keep its ongoing process active. at this point, it seems like there are only two options that can be done against the malware further execution process: either to terminate the process manually, or elsewhere delete the targeted file.



In that regard it is important to note that the stage in which we delete the file is crucial.

As we saw before, if the file is deleted before first queried, no other actions will be taken by the malware since it didn't find its targeted file. But if we try to delete the file while the exfiltration process has already been started, nothing will prevent the malware from further execute.

So in this short check, we found out that the first connection with the remote server using the permanent user is the key point for the malware in order to launch its further processes. Once connection is observed, the only action that can be taken against the malware execution process is to terminate the process completely or delete the targeted file before the exfiltration process starts. unless doing so, the malware will keep it execution flow as intended.

Let's move on now to examine the file in the "x32dbg" and see if the details we came up with so far matches what we'll find in the debugger.

As discussed in the Advanced static analysis phase, there are few key points in the malware execution flow, in which the malware has to decide, based upon the Zero flag values, either to launch its further execution process or stop.

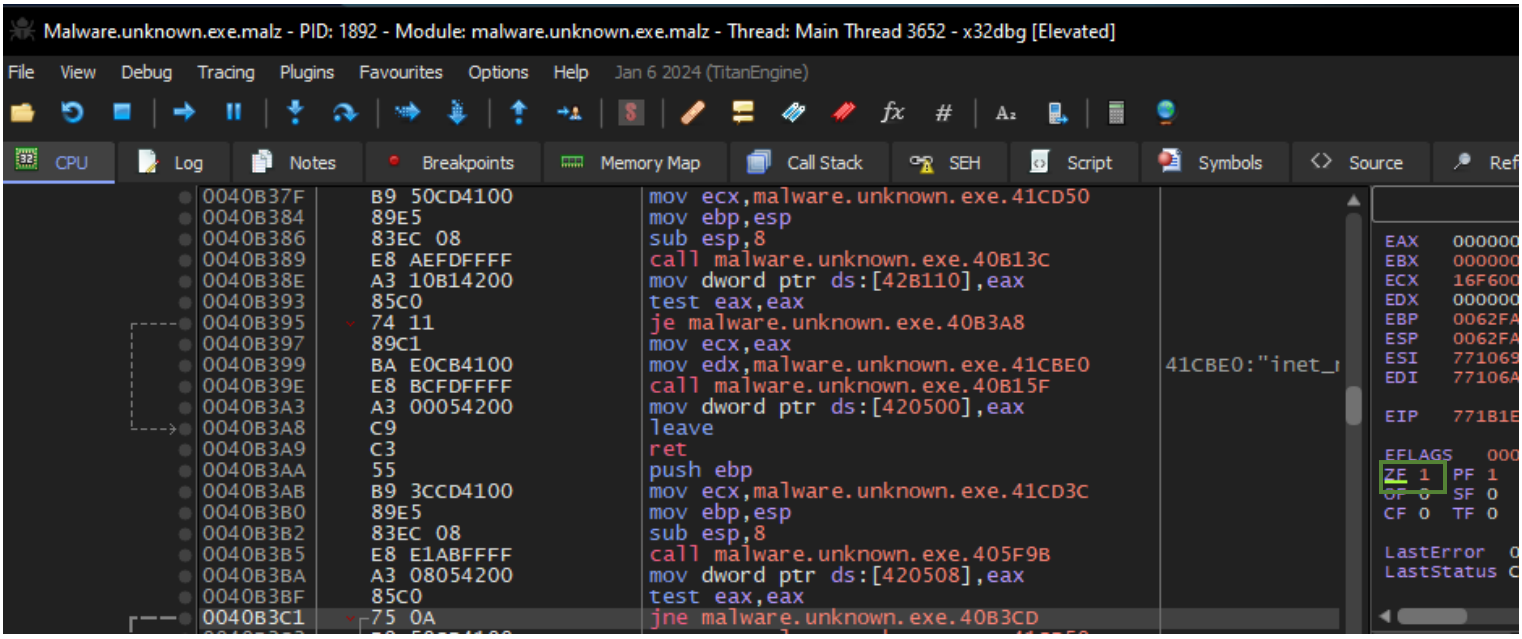
In that regard, we found the place in the debugger in which the first "jne" is invoked.

As we look through this both in Cutter and in the debugger, we can see the values that are being moved before the "jne" instruction is actually the ones that are required for the call of "malware.unknown.exe" function, while in Cutter It comes under the obfuscated name of "fcn.00405f9b."

```
[0x0040b3aa]
fcn.0040b3aa();
0x0040b3aa      push    ebp
0x0040b3ab      mov     ecx, data.0041cd3c ; 0x41cd3c
0x0040b3b0      mov     ebp, esp
0x0040b3b2      sub     esp, 8
0x0040b3b5      call    fcn.00405f9b ; fcn.00405f9b
0x0040b3ba      mov     dword [0x420508], eax
0x0040b3bf      test    eax, eax
0x0040b3c1      jne     0x40b3cd
```

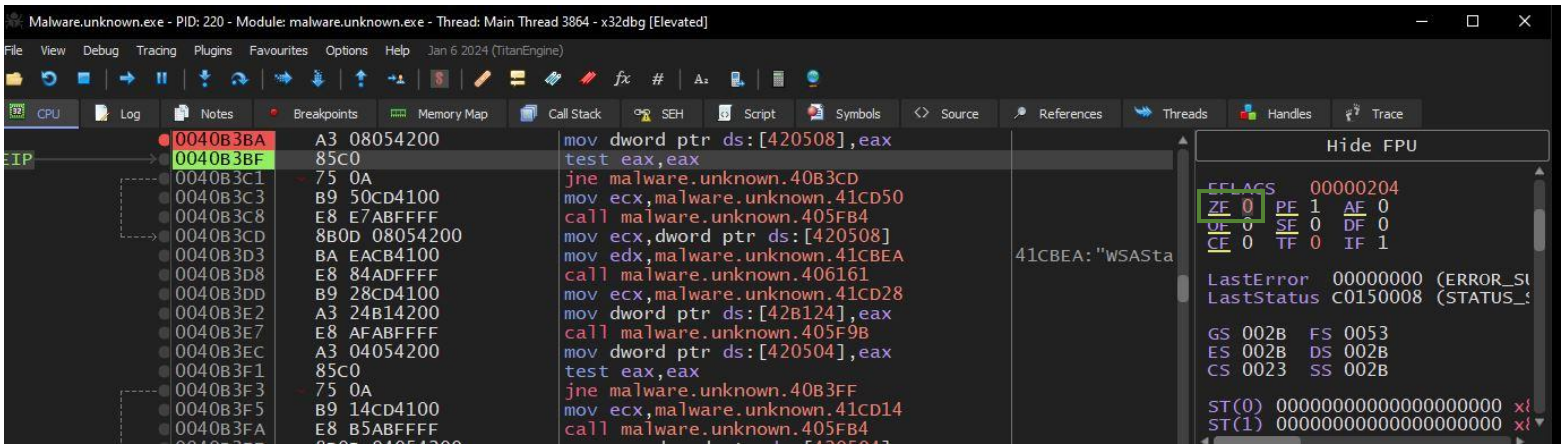
```
0B3AA    55              push ebp
0B3AB    B9 3CCD4100     mov ecx,malware.unknown.exe.41CD3C
0B3B0    89E5            mov ebp,esp
0B3B2    83EC 08         sub esp,8
0B3B5    E8 E1ABFFFF     call malware.unknown.exe.405F9B
0B3BA    A3 08054200     mov dword ptr ds:[420508],eax
0B3BF    85C0            test eax,eax
0B3C1    75 0A           jne malware.unknown.exe.40B3CD
```

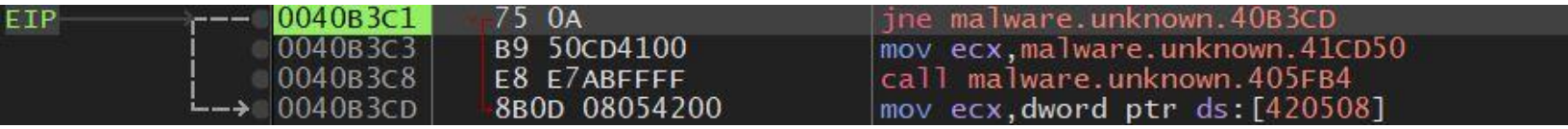
From the right-side menu of the debugger we can also see that the ZF is set to 1, meaning currently the values are equal, hence a jump will not be made.

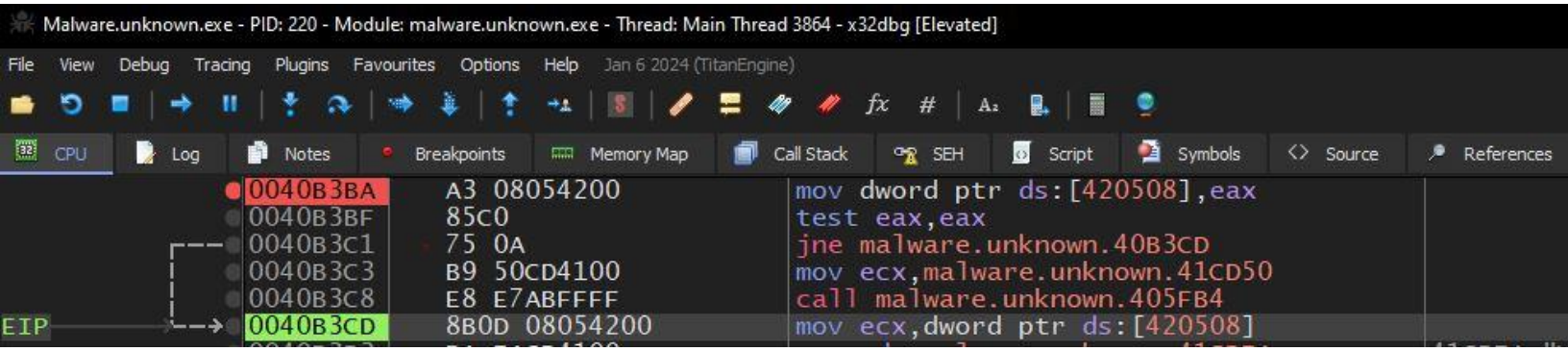


Since we want to examine the execution flow of the program more carefully, we are going to trick the malware and change the ZF as if the values of "eax" are not equal (ZF Set to 0) resulting in a further launch of the malware processes.

So what we are going to do is to change the ZF values of the "test,eax,eax" instruction to 0 and we will follow up with the process:



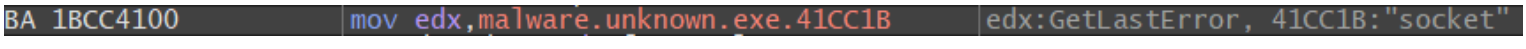




As you can see, the malware reached exactly where we though it will, moving as if the jump has been made, and now we can examine more carefully what happens in the malware later execution process.

Moving on with this, in the advanced static analysis phase we were able to identify a string named "socket", what potentially gave us a clue regarding the malware remote communication.

While obfuscated in "Cutter", in the debugger things look different, and we were able to find the place in which this string first appears.



What we were also able to identify is the next following behavior of the malware, first closing the socket, and then reloads it to the "eax" in order to call the malware function again:

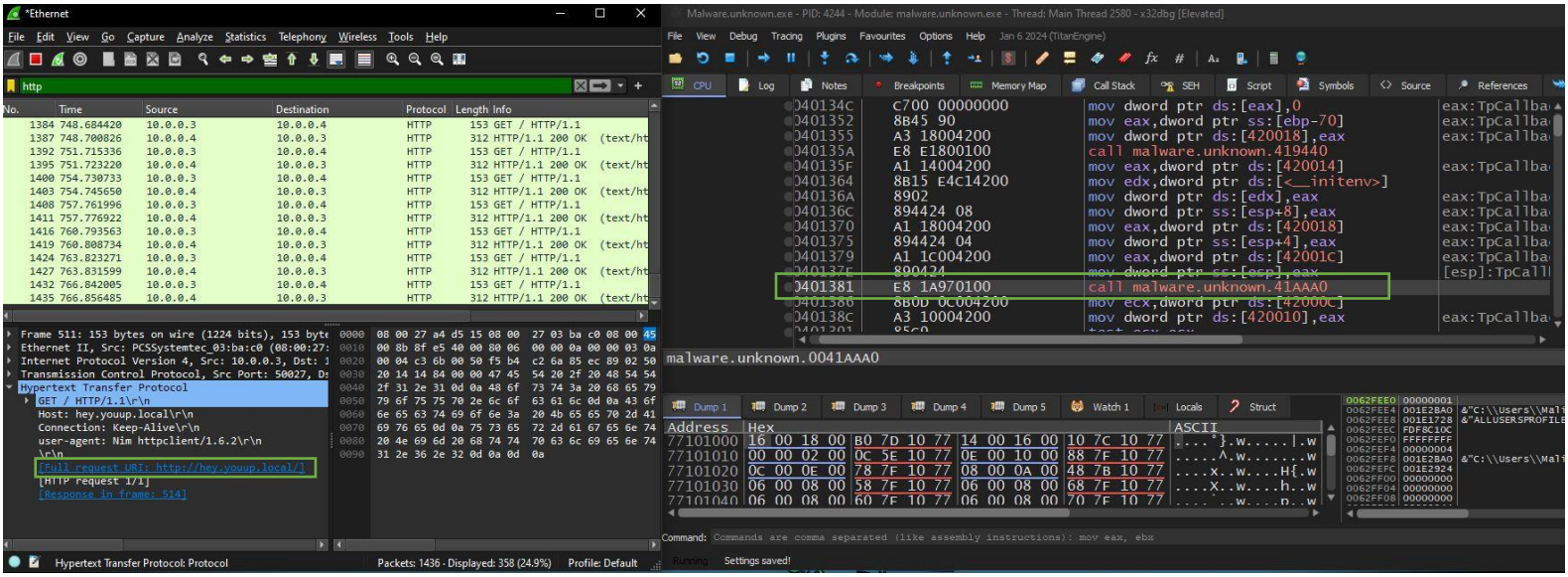
```
BA 22CC4100    mov edx,malware.unknown.41CC22          edx:"closesocket",
A3 3CB14200    mov dword ptr ds:[42B13C],eax           eax:socket
E8 FEACFFFF    call malware.unknown.406161
```

If we look at the same time in the code in "Cutter", no indication that this is what actually happens, what may seem like a good way to disguise the malware processes.

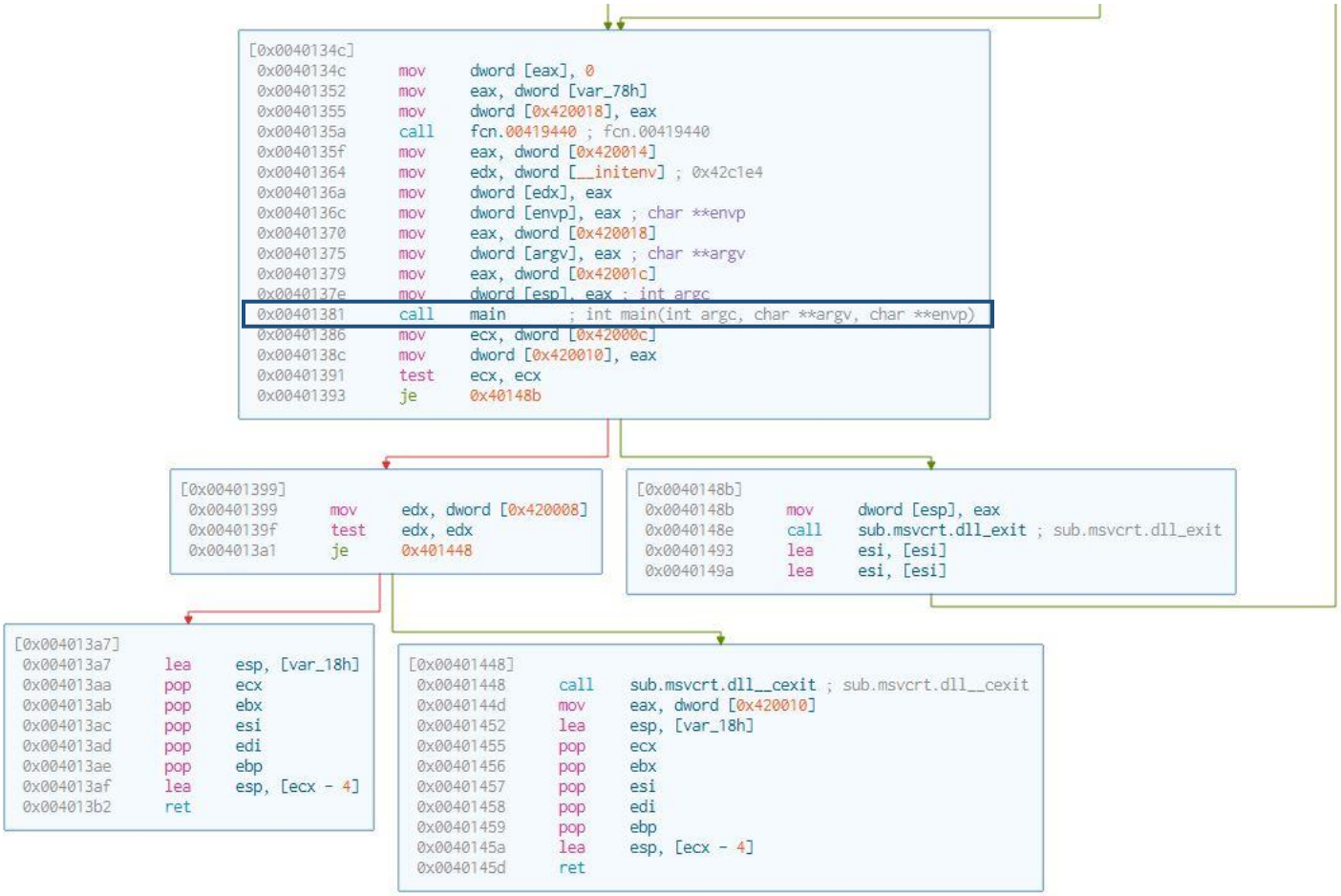
```
0x0040b454      mov     edx, str.closesocket ; 0x41cc22
0x0040b459      mov     dword [0x42b13c], eax
0x0040b45e      call    fcn.00406161 ; fcn.00406161
```

Moving on with the code instructions, we tried to find the exact point in which the malware communicate with the remote server.

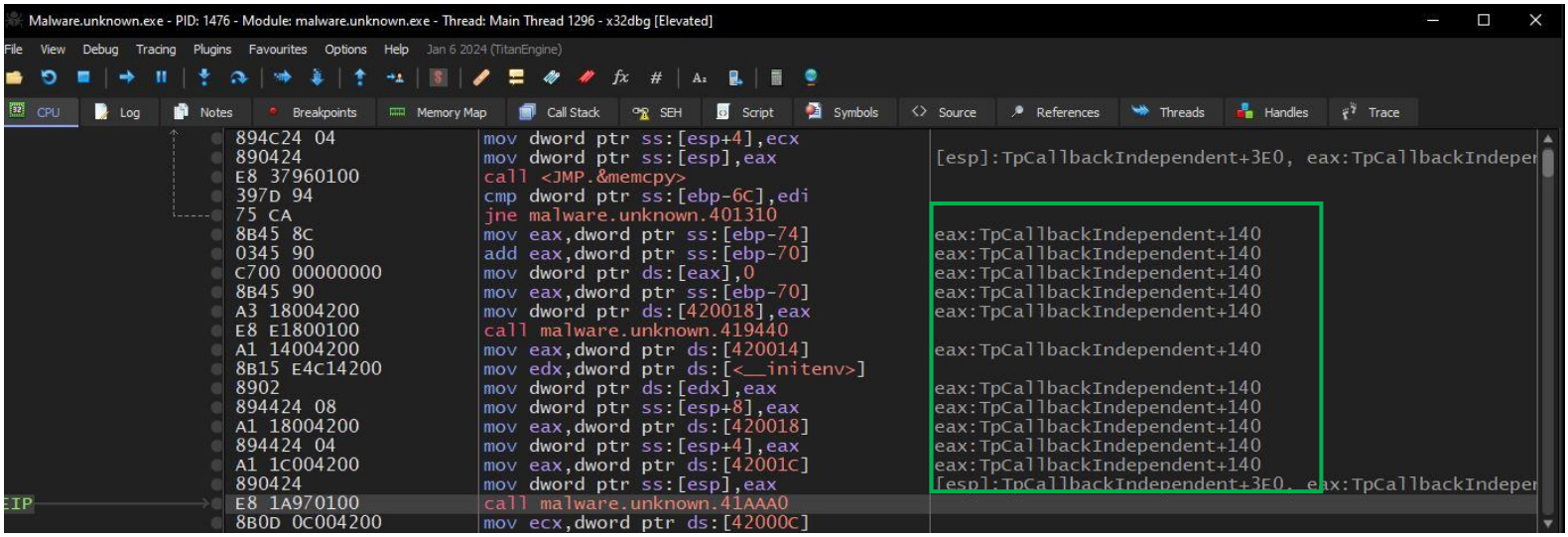
As seen from the Image below, after moving some variables into the eax, the call is ready to send the HTTP GET requests, as seen in Wireshark.



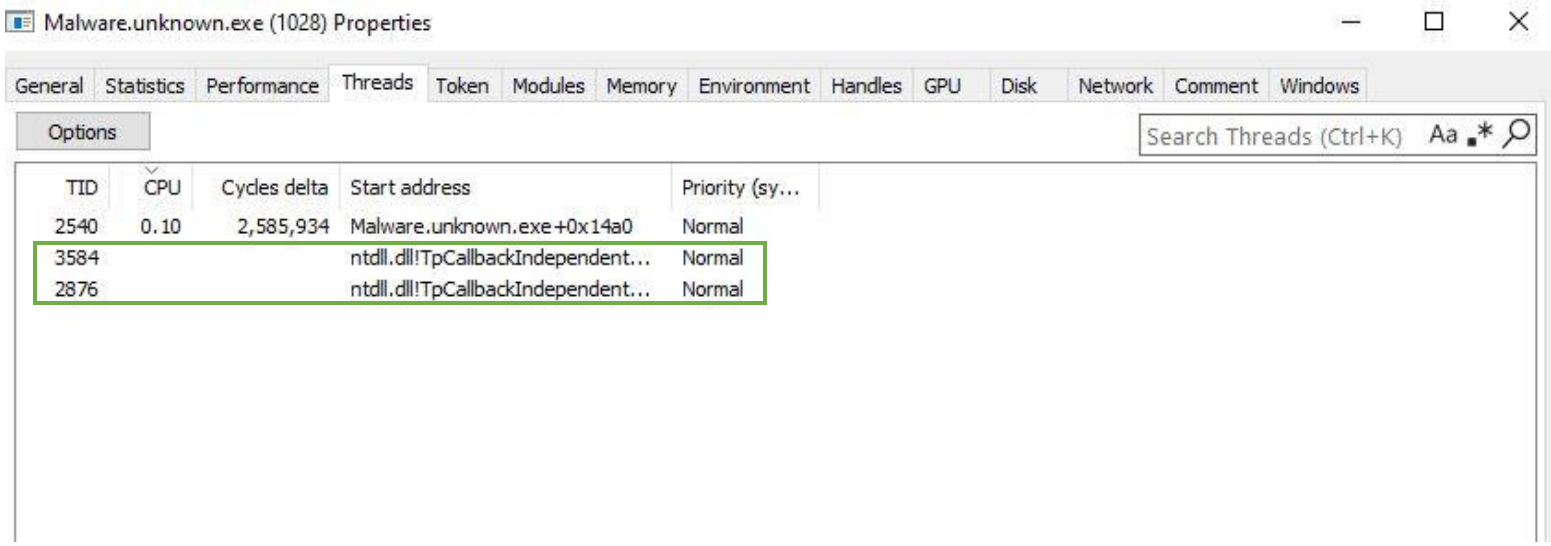
When examining the same point it in "Cutter", we can see a call is made for the "main" function, that takes three arguments. still we are not sure what those arguments are since they only refer to a place in memory.



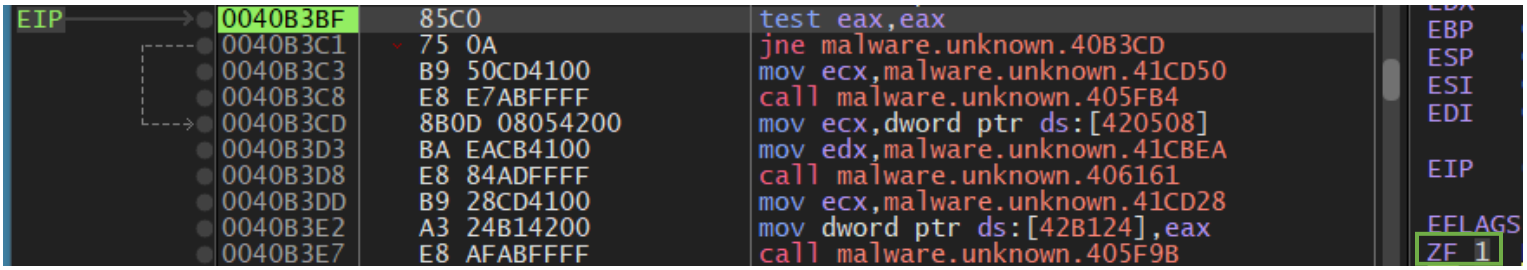
Examining again the code in the debugger, reveled that these variables are actually what generates the Domain callback:



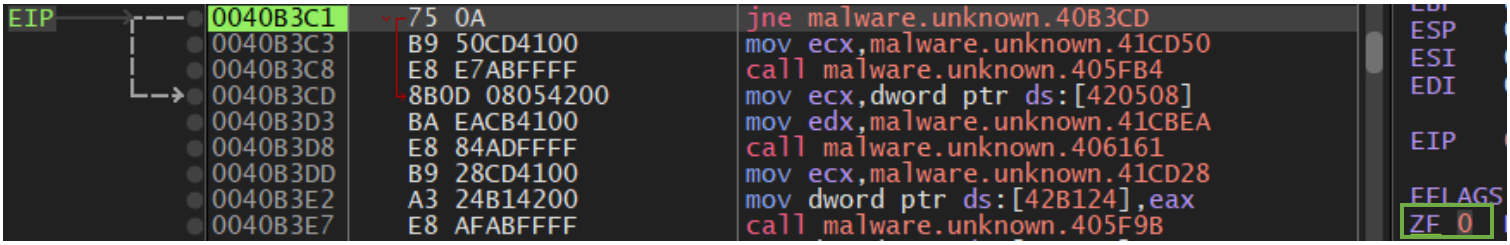
Opening process hacker while running the malware confirms that assumption. As seen in here, there are three open threads, while two of them are the ones that has been seen in the debugger, used by the ntdll.dll library.



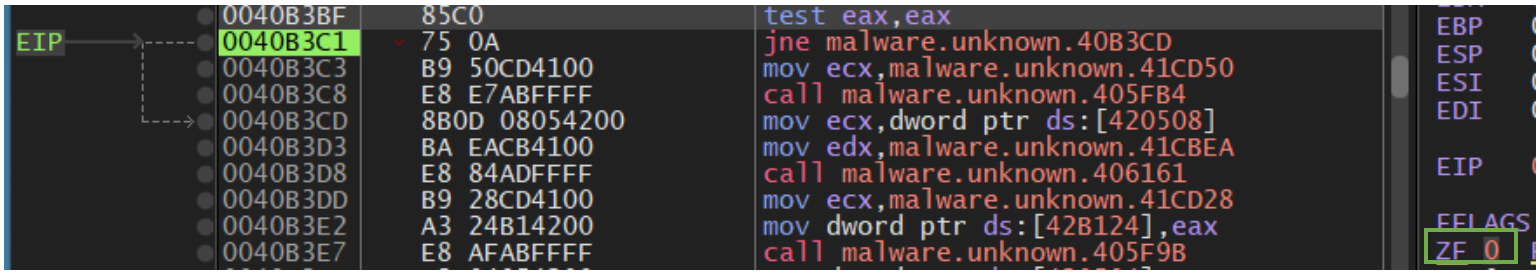
Lastly, I came across very interesting behavior of the malware, that may suggest it uses some kind of anti-analysis technique.
I became aware that while manually setting the zero flag to 1 in order to examine what happen in case the jump is not taken, it is suddenly changed my ZF values automatically to "0".
As you can see we first setting the ZF to 1, because it might tell us what the malware is doing when the jump is not taken.



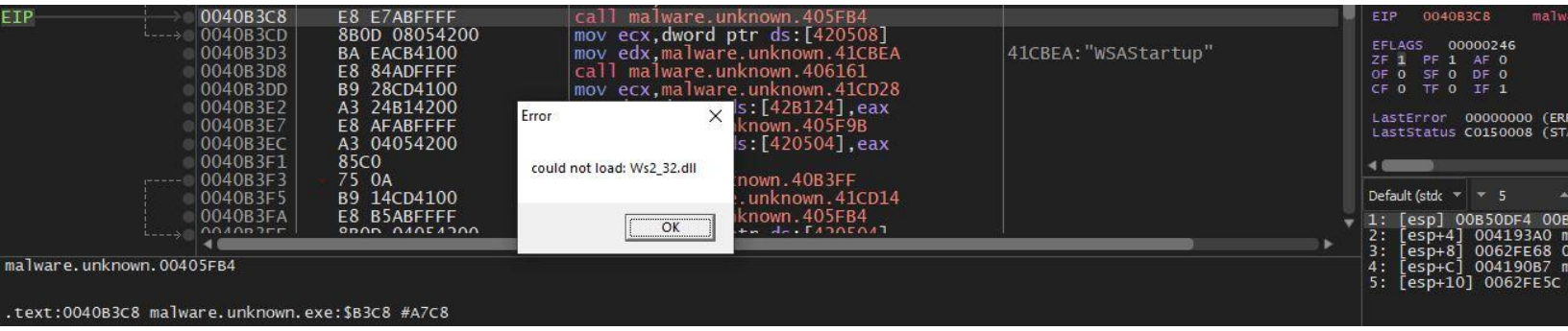
Suddenly, moving on to the next "jne" instruction changed the ZF back to "0".



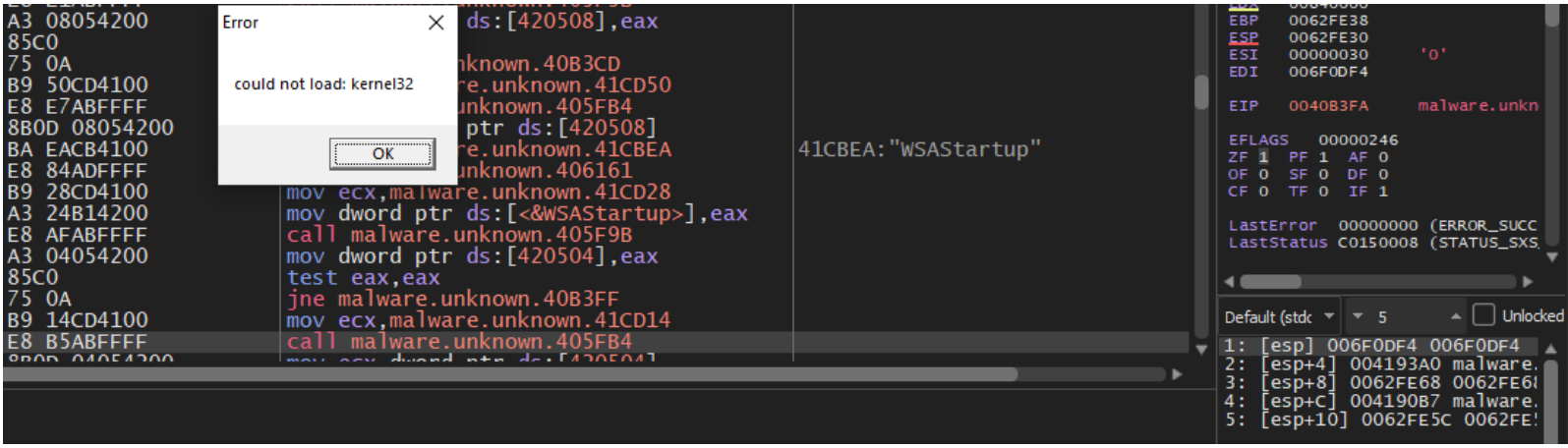
I also came back to "test" instruction to double-check and as you can see, the ZF value has returned to "0" without any intervention by me.



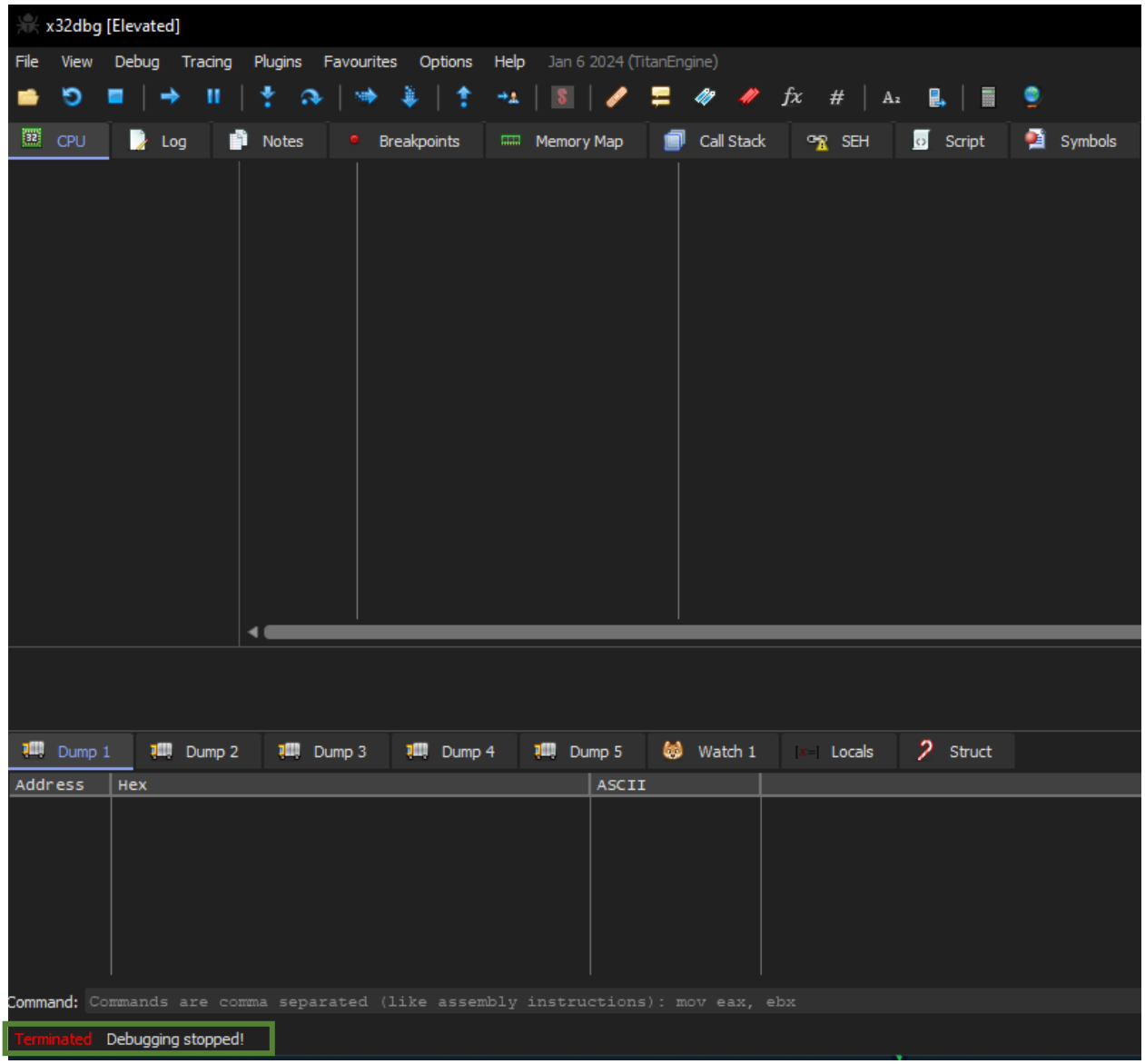
Since I became aware of it, I decided to manually set both values to 1: in the "test" instruction and in the "jne" instruction. When doing so, the jump has not been taken and I got the following error instead:



In case it passes that first check (zero flagged set to 0 either on "test" instruction or in "jne" instruction) the jump will be taken and eventually we'll move to our second "test" instruction in which we again, need to set both values manually to 1. When doing so, we got a different error, this time regrading kernel32 dll:



Nevertheless, it doesn't matter which approach do we take, this error messages will eventually terminate our debug process as shown in here:



When looking into this in "Cutter", we can see that the errors we faced are actually refers to the place in the function that takes care of the procedures when the jump in not taken.
As we saw in the advanced static analysis phase, the strings "str.could_not_load" and also the API call "GetLastError", are probably the ones that generated the popped-up error messages.

```
0x00405fb4      push    ebp
0x00405fb5      mov     ebp, esp
0x00405fb7      push    edi
0x00405fb8      push    esi
0x00405fb9      push    ebx
0x00405fba      sub     esp, 0x41c
0x00405fc0      mov     dword [var_410h], ecx
0x00405fc6      mov     ebx, dword data.0041b08c ; 0x41b08c
0x00405fcc      mov     dword [esp], 2
0x00405fd3      call    ebx
0x00405fd5      mov     edx, str.could_not_load: ; 0x41c470
0x00405fda      call    fcn.00403189 ; fcn.00403189
0x00405fdf      mov     eax, dword [var_410h]
0x00405fe5      call    fcn.00403175 ; fcn.00403175
0x00405fea      mov     dword [esp], 2
0x00405ff1      mov     dword [var_414h], eax
0x00405ff7      call    ebx
0x00405ff9      mov     edx, dword [var_414h]
0x00405fff      call    fcn.00403189 ; fcn.00403189
0x00406004      call    dword [GetLastError] ; 0x42c194 ; DWORD GetLastError(void)
```

So to summarize this part of analysis, through looking into the debugger we found out the main points in the malware execution process that eventually leads to the data exfiltration process.
In case the program fails to launch during debugging, an alert pop-up messages will appear, and the debugging process will terminate itself, probably in order to evade follow up detection.
In our next phase of analysis, we'll summarize the main findings of the analysis in the IOCs section.

PRACTICAL MALWARE ANALYSIS & TRIAGE

Indicators of Compromise

Network Indicators

hxxp://hey.youup.local

hxxp://auth.ns.local

hxxp://cosmosfurootsemporium.local

Host-based Indicators

Malware.unknown.exe

cosmo.jpeg

LogonSessionId_0_163572

Rules & Signatures

A full set of YARA rules is included in Appendix A.

{Information on specific signatures, i.e. strings, URLs, etc}

Appendices

A. Yara Rules

```
rule Yara_Data_Exfiltration_Mal {

    meta:
        last_updated = "2024-05-27"
        author = "TMCA"
        description = "A sample Yara rule for Data Exfiltration Malware"

    strings:
        $string1 = "cosmo.jpeg" ascii
        $string2 = "hwtwtwpw:w/w/whwewyw.wywowuwuwpw.wlwowcwawlw" ascii
        $string3 = "axuxtxhx.xnxsx.xlxoxcxaxlx" ascii
        $string4 = "BcBoBsBmBoBsBfBuBrBbBoBoBtBsBeBmBpBoBrBiBuBmB.BlBoBcBaBlB"
ascii
        $string5 = "nim"
        $PE_magic_byte = "MZ"


    condition:
        $PE_magic_byte at 0 and
            $string1 and $string2 or $string3 or $string4 or $string5
}
```

B. Callback URLs

Domain	Port
hxxp://hey.youup.local	80
hxxp://auth.ns.local	80
hxxp://cosmosfurootsemporium.local	80

C. Yara Detections

```
C:\Users\Malianalis\Desktop
λ yara64 YR.yara Malware.unknown.exe
Yara_Data_Exfiltration_Mal Malware.unknown.exe
```

Fig 1: Yara scan found out the rule has been triggered due to "Malware.unknown.exe".

```
C:\Users\Malianalis\Desktop
λ yara64 YR.yara Malware.unknown.exe -s -w -p 32
Yara_Data_Exfiltration_Mal Malware.unknown.exe
0x1ccf0:$string1: cosmo.jpeg
0x1cec8:$string2: hwtwtwpw:w/w/whwewyw.wywowuwuwpw.wlwowcwawlw
0x1cf08:$string3: axuxtxhx.xnxsx.xlxoxcxaxlx
0x1cf49:$string4: BcBoBsBmBoBsBfBuBrBbBoBoBtBsBeBmBpBoBrBiBuBmB.BlBoBcBaBlB
0x1a2f6:$string5: nim
0x1a32c:$string5: nim
0x1a4da:$string5: nim
0x1abb6:$string5: nim
0x1ac35:$string5: nim
0x1acd0:$string5: nim
0x1b04e:$string5: nim
0x1b11e:$string5: nim
0x1b272:$string5: nim
0x1b36d:$string5: nim
0x1b44c:$string5: nim
0x1b4ac:$string5: nim
0x1b50c:$string5: nim
0x1b56c:$string5: nim
0x1b5cc:$string5: nim
0x1b60c:$string5: nim
0x1b6ac:$string5: nim
0x1b76c:$string5: nim
0x1b80c:$string5: nim
0x1b953:$string5: nim
0x1bc4f:$string5: nim
0x1bcd4:$string5: nim
0x1bd91:$string5: nim
0x1be97:$string5: nim
0x1c110:$string5: nim
0x1c1f3:$string5: nim
0x1c2dc:$string5: nim
0x1c80f:$string5: nim
0x1c8b2:$string5: nim
0x1ca52:$string5: nim
0x1cbd2:$string5: nim
0x1cd53:$string5: nim
0x1cdd3:$string5: nim
0x0:$PE_magic_byte: MZ
```

Fig 2: Yara scan detects the places in memory that were triggered due to their strings.