Midterm: Malware Reverse Engineering

Due: 03/05/2021

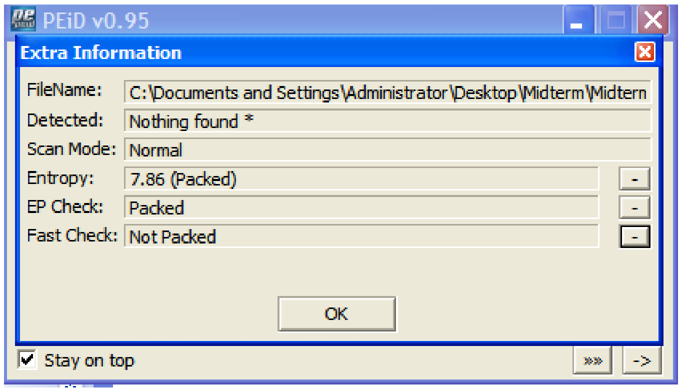
1. Answer the following questions based on your analysis of Problem 1.

Problem 1 SHA256: 212b29a8e36ccc8f65205122e33d84940a156a9a91329f747a713f988a157948

1) Are there any indications that this malware is packed? What are they? What is it packed with?

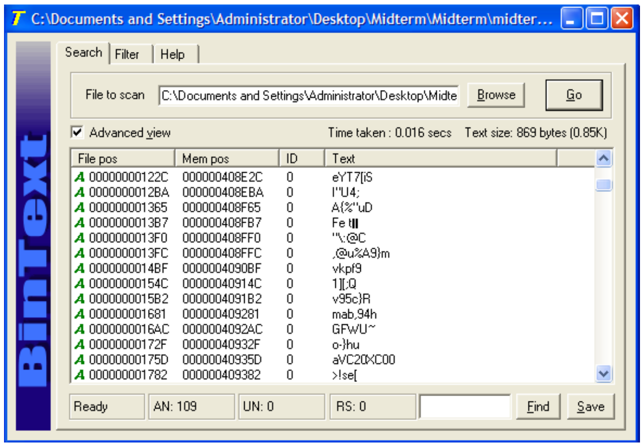
Yes, the malware is packed. Indications of being packed are:

- Depending on the Entropy value. If it is less than 7, file is not packed and if it is more than 7, file is packed. Here, Entropy of the malware is more than 7. Hence it indicates to be packed.



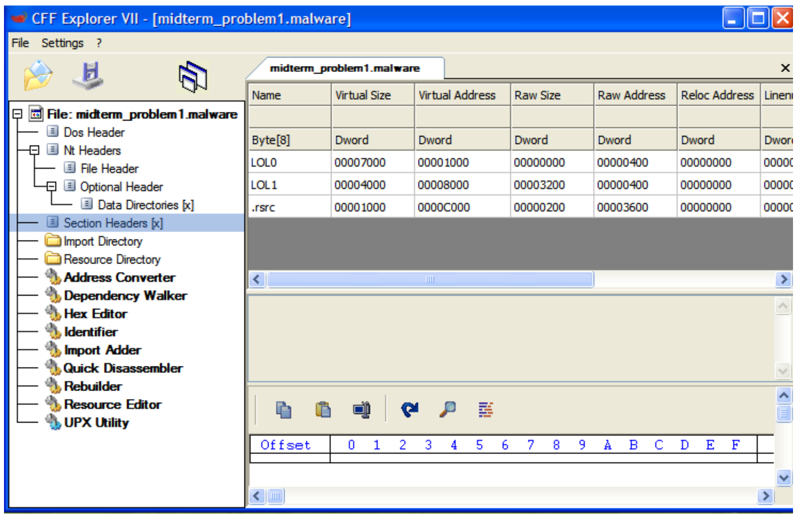
```
FileName:   C:\Documents and Settings\Administrator\Desktop\Midterm\Midtern
Detected:   Nothing found *
Scan Mode:  Normal
Entropy:    7.86 (Packed)
EP Check:   Packed
Fast Check: Not Packed
```

- Second indicator is that strings are obfuscated as seen below which is not the case for a plain file.



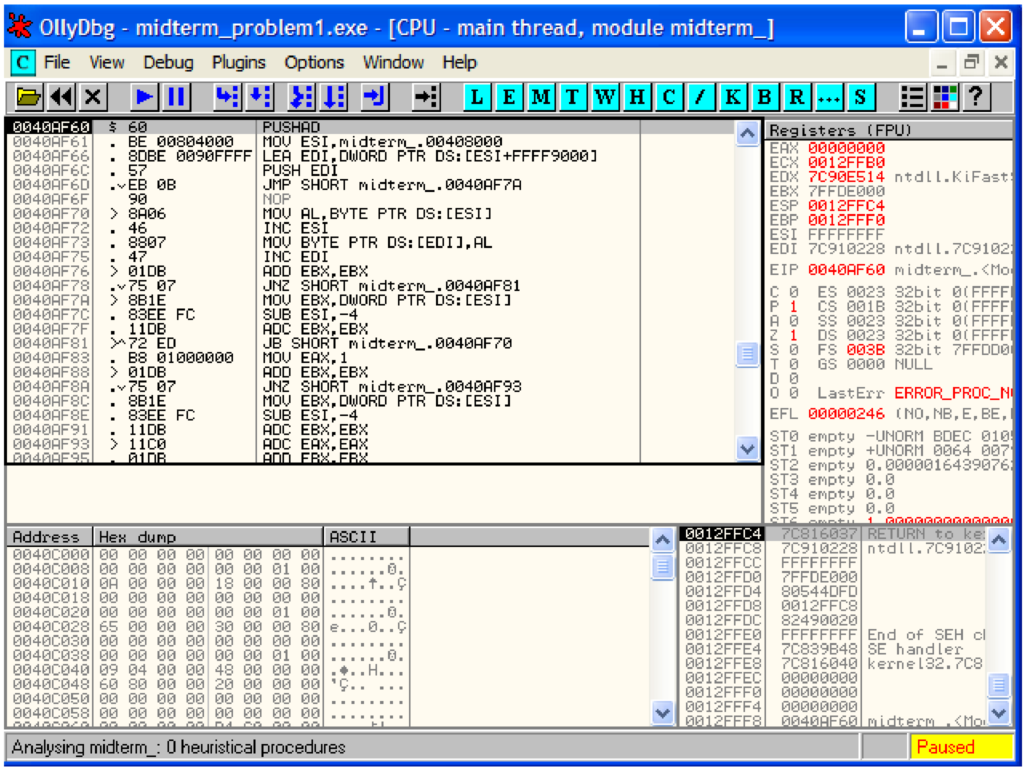
```
C:\Documents and Settings\Administrator\Desktop\Midterm\Midterm\midter...
File to scan  C:\Documents and Settings\Administrator\Desktop\Midte
[✓] Advanced view        Time taken : 0.016 secs   Text size: 869 bytes (0.85K)
File pos            Mem pos            ID   Text
A 00000000122C      000000408E2C       0    eYT7[iS
A 0000000012BA      000000408EBA       0    I"U4;
A 000000001365      000000408F65       0    A{%"uD
A 0000000013B7      000000408FB7       0    Fe tll
A 0000000013F0      000000408FF0       0    "\:@C
A 0000000013FC      000000408FFC       0    ,@u%A9}m
A 0000000014BF      0000004090BF       0    vkpf9
A 00000000154C      00000040914C       0    1][;Q
A 0000000015B2      0000004091B2       0    v95c}R
A 000000001681      000000409281       0    mab,94h
A 0000000016AC      0000004092AC       0    GFWU~
A 00000000172F      00000040932F       0    o-}hu
A 00000000175D      00000040935D       0    aVC20XC00
A 000000001782      000000409382       0    >!se[
Ready   AN: 109   UN: 0   RS: 0
```

- One more indicator is that the section names are obfuscated. (LOL0, LOL1). Which can be seen through CFF explorer below. The section names in a plain file are known section names like .text, .data, .rsrc etc.


CFF Explorer VII - [midterm_problem1.malware]

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenu
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword
LOL0	00007000	00001000	00000000	00000400	00000000	0000
LOL1	00004000	00008000	00003200	00000400	00000000	0000
.rsrc	00001000	0000C000	00000200	00003600	00000000	0000

The malware is a UPX packed file. One indicator for saying it's UPX packed is that when you load it into debugger. The instruction at the entry point is "PUSHAD" (As shown below). Here, all the registers are pushed into stack to perform unpacking of the file. The code found between "pushad" and "popad" from the entry point is the unpacking code.


OllyDbg - midterm_problem1.exe - [CPU - main thread, module midterm_]

```
0040AF60  $ 60              PUSHAD
0040AF61  . BE 00804000     MOV ESI,midterm_.00408000
0040AF66  . 8DBE 0090FFFF   LEA EDI,DWORD PTR DS:[ESI+FFFF9000]
0040AF6C  . 57              PUSH EDI
0040AF6D  .vEB 0B           JMP SHORT midterm_.0040AF7A
0040AF6F    90              NOP
0040AF70  > 8A06            MOV AL,BYTE PTR DS:[ESI]
0040AF72  . 46              INC ESI
0040AF73  . 8807            MOV BYTE PTR DS:[EDI],AL
0040AF75  . 47              INC EDI
0040AF76  > 01DB            ADD EBX,EBX
0040AF78  .v75 07           JNZ SHORT midterm_.0040AF81
0040AF7A  > 8B1E            MOV EBX,DWORD PTR DS:[ESI]
0040AF7C  . 83EE FC         SUB ESI,-4
0040AF7F  . 11DB            ADC EBX,EBX
0040AF81  >^72 ED           JB SHORT midterm_.0040AF70
0040AF83  . B8 01000000     MOV EAX,1
0040AF88  > 01DB            ADD EBX,EBX
0040AF8A  .v75 07           JNZ SHORT midterm_.0040AF93
0040AF8C  . 8B1E            MOV EBX,DWORD PTR DS:[ESI]
0040AF8E  . 83EE FC         SUB ESI,-4
0040AF91  . 11DB            ADC EBX,EBX
0040AF93  > 11C0            ADC EAX,EAX
0040AF95  . 01DB            ADD EBX,EBX
```

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFast
EBX 7FFDE000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C910228 ntdll.7C9102

EIP 0040AF60 midterm_.<Mo

C 0  ES 0023 32bit 0(FFFF
P 1  CS 001B 32bit 0(FFFF
A 0  SS 0023 32bit 0(FFFF
Z 1  DS 0023 32bit 0(FFFF
S 0  FS 003B 32bit 7FFDD0
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_PROC_N

EFL 00000246 (NO,NB,E,BE,

ST0 empty -UNORM BDEC 010
ST1 empty +UNORM 0064 007
ST2 empty 0.0000001643907
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.00000000000000
```

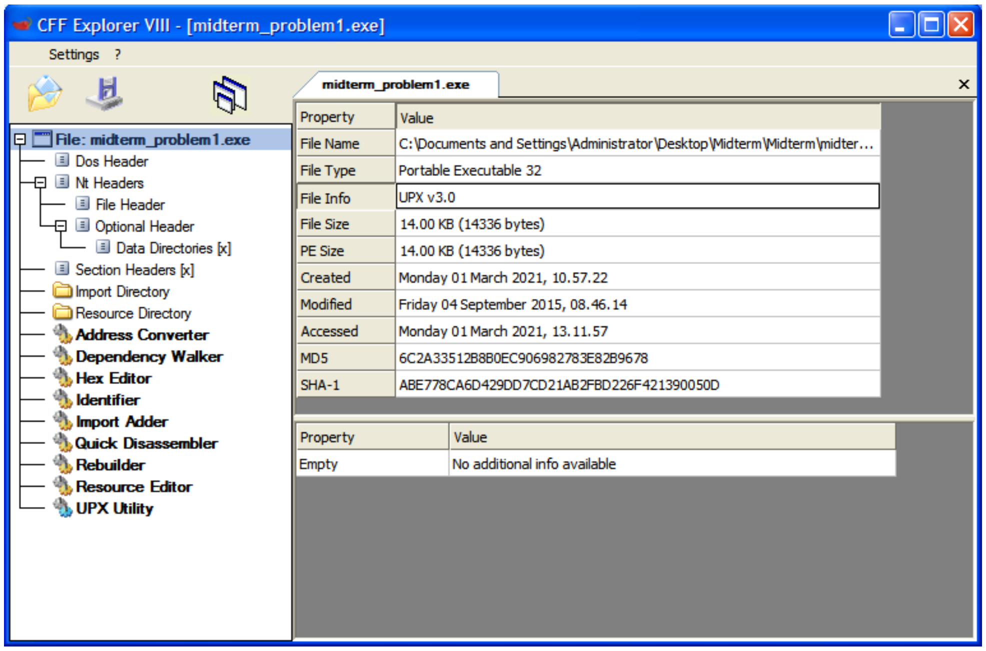
```
Address   Hex dump                 ASCII
0040C000  00 00 00 00 00 00 00 00  ........
0040C008  00 00 00 00 00 00 01 00  ......0.
0040C010  0A 00 00 00 18 00 00 80  ....↑..Ç
0040C018  00 00 00 00 00 00 00 00  ........
0040C020  00 00 00 00 00 00 01 00  ......0.
0040C028  65 00 00 00 30 00 00 80  e...0..Ç
0040C030  00 00 00 00 00 00 00 00  ........
0040C038  00 00 00 00 00 00 01 00  ......0.
0040C040  09 04 00 00 48 00 00 00  .♦..H...
0040C048  60 80 00 00 20 00 00 00  `Ç.. ...
0040C050  00 00 00 00 00 00 00 00  ........
0040C058  00 00 00 00 00 00 00 00  ........
```

```
0012FFC4  7C816037  RETURN to ke
0012FFC8  7C910228  ntdll.7C9102
0012FFCC  FFFFFFFF
0012FFD0  7FFDE000
0012FFD4  80544DFD
0012FFD8  0012FFC8
0012FFDC  82490020
0012FFE0  FFFFFFFF  End of SEH c
0012FFE4  7C839B48  SE handler
0012FFE8  7C816040  kernel32.7C8
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  0040AF60  midterm_.<Mo
```

Analysing midterm_: 0 heuristical procedures

Paused

One other way I could confirm it was UPX packed is through CFF explorer, under file info.



CFF Explorer VIII - [midterm_problem1.exe]

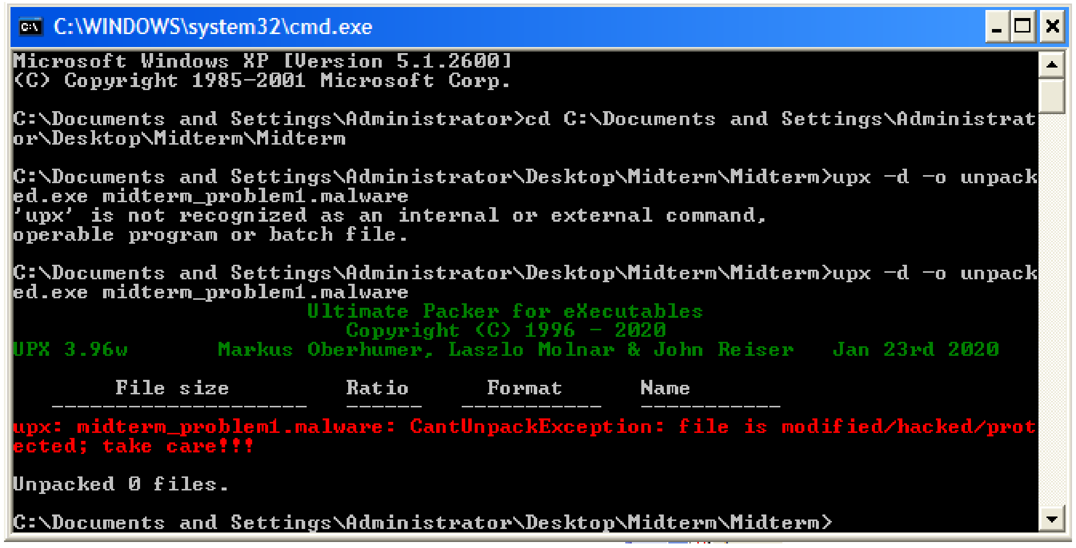
Property	Value
File Name	C:\Documents and Settings\Administrator\Desktop\Midterm\Midterm\midter...
File Type	Portable Executable 32
File Info	UPX v3.0
File Size	14.00 KB (14336 bytes)
PE Size	14.00 KB (14336 bytes)
Created	Monday 01 March 2021, 10.57.22
Modified	Friday 04 September 2015, 08.46.14
Accessed	Monday 01 March 2021, 13.11.57
MD5	6C2A33512B8B0EC906982783E82B9678
SHA-1	ABE778CA6D429DD7CD21AB2FBD226F421390050D

Property	Value
Empty	No additional info available

Generally, the section names of a UPX packed file are named .upx but in this case, looks like the malware obfuscated the section names in order to hide its packed nature.

2) Are you able to unpack it? Why or why not?

Initially I used the UPX tool available online that can be used to unpack a UPX packed file. But, in this case, the malware could not be unpacked and throwed exception of the file being modified, or protected. (as shown below, “CantUnpackException”). This can be due to malware being patched against “upx -d” since, UPX is an open source tool and very commonly used to unpack UPX files. This malware looks like it is protected against unpacking through this tool.

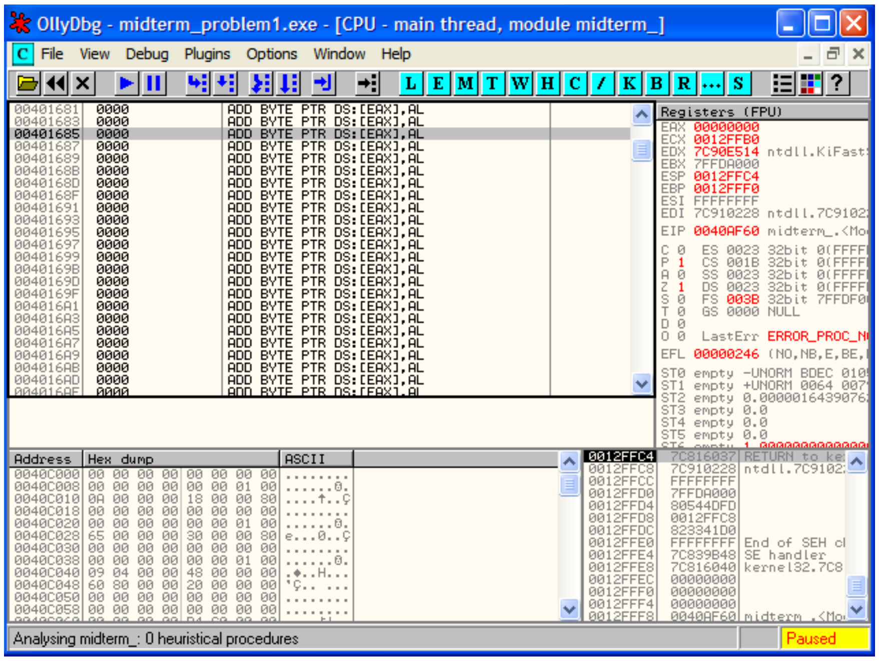


C:\WINDOWS\system32\cmd.exe

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>cd C:\Documents and Settings\Administrat
or\Desktop\Midterm\Midterm

C:\Documents and Settings\Administrator\Desktop\Midterm\Midterm>upx -d -o unpack
ed.exe midterm_problem1.malware
'upx' is not recognized as an internal or external command,
operable program or batch file.

C:\Documents and Settings\Administrator\Desktop\Midterm\Midterm>upx -d -o unpack
ed.exe midterm_problem1.malware
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2020
UPX 3.96w       Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
upx: midterm_problem1.malware: CantUnpackException: file is modified/hacked/prot
ected; take care!!!

Unpacked 0 files.

C:\Documents and Settings\Administrator\Desktop\Midterm\Midterm>
```

Then I tried unpacking it manually through ollydbg. First, I have to get to the original entry point of the file, such that, it can be dumped and we have the plain file. For getting to the original entry point. The technique I used is as follows:

- Once the file is loaded in Ollydbg, executed the file until popad. What it does is, the unpacking routine stored in between the pushad and popad is executed and the file at this point is unpacked and the control flow changes to the main() of the file. Which is the original entry point of the file. Now, where the control flow is changed? One way to look out was to look for a long jump after popad. In this case, it was the second jump that took me to the OEP (Original entry point). Address of OEP in this case is 00401681. Figures below show the OEP location before executing the Unpacking routine and after executing the unpacking routine. You can see, in the second image the instructions are like that of a plain file and the first image has no plain instructions.


OllyDbg - midterm_problem1.exe - [CPU - main thread, module midterm_]
File
View
Debug
Plugins
Options
Window
Help
L E M T W H C / K B R ... S

```
00401681  0000      ADD BYTE PTR DS:[EAX],AL
00401683  0000      ADD BYTE PTR DS:[EAX],AL
00401685  0000      ADD BYTE PTR DS:[EAX],AL
00401687  0000      ADD BYTE PTR DS:[EAX],AL
00401689  0000      ADD BYTE PTR DS:[EAX],AL
0040168B  0000      ADD BYTE PTR DS:[EAX],AL
0040168D  0000      ADD BYTE PTR DS:[EAX],AL
0040168F  0000      ADD BYTE PTR DS:[EAX],AL
00401691  0000      ADD BYTE PTR DS:[EAX],AL
00401693  0000      ADD BYTE PTR DS:[EAX],AL
00401695  0000      ADD BYTE PTR DS:[EAX],AL
00401697  0000      ADD BYTE PTR DS:[EAX],AL
00401699  0000      ADD BYTE PTR DS:[EAX],AL
0040169B  0000      ADD BYTE PTR DS:[EAX],AL
0040169D  0000      ADD BYTE PTR DS:[EAX],AL
0040169F  0000      ADD BYTE PTR DS:[EAX],AL
004016A1  0000      ADD BYTE PTR DS:[EAX],AL
004016A3  0000      ADD BYTE PTR DS:[EAX],AL
004016A5  0000      ADD BYTE PTR DS:[EAX],AL
004016A7  0000      ADD BYTE PTR DS:[EAX],AL
004016A9  0000      ADD BYTE PTR DS:[EAX],AL
004016AB  0000      ADD BYTE PTR DS:[EAX],AL
004016AD  0000      ADD BYTE PTR DS:[EAX],AL
004016AF  0000      ADD BYTE PTR DS:[EAX],AL
```

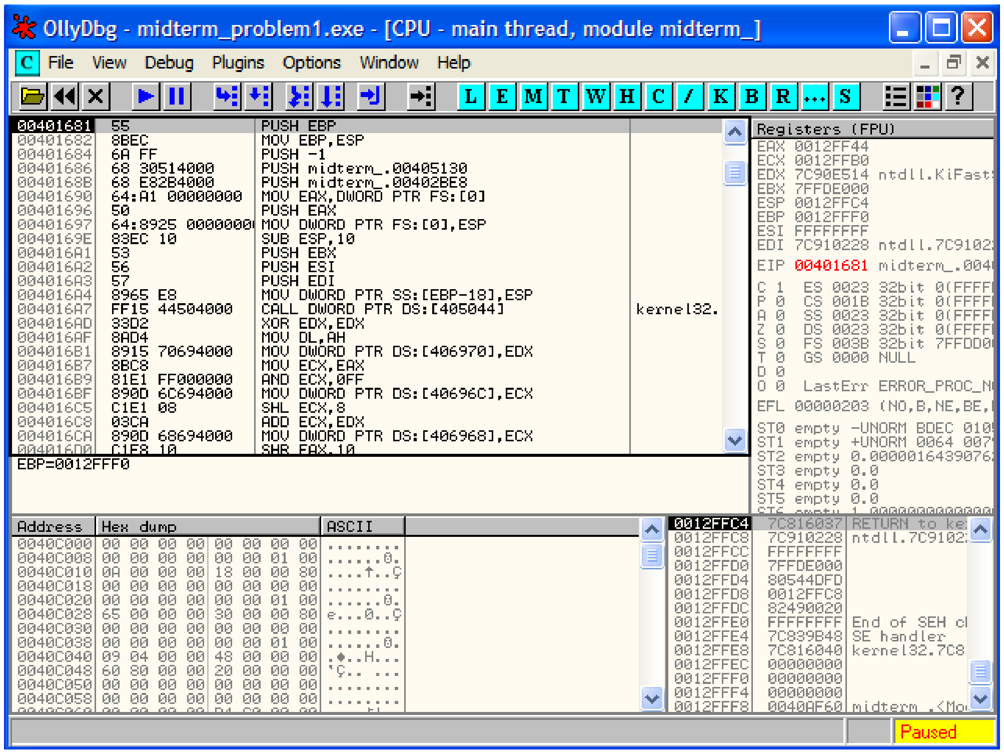
```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFast
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C910228 ntdll.7C9102

EIP 0040AF60 midterm_.<Mo

C 0  ES 0023 32bit 0(FFFF
P 1  CS 001B 32bit 0(FFFF
A 0  SS 0023 32bit 0(FFFF
Z 1  DS 0023 32bit 0(FFFF
S 0  FS 003B 32bit 7FFDF0
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_PROC_N

EFL 00000246 (NO,NB,E,BE,

ST0 empty -UNORM BDEC 010
ST1 empty +UNORM 0064 007
ST2 empty 0.0000016439076
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
```

```
Address  Hex dump                 ASCII
0040C000 00 00 00 00 00 00 00 00 ........
0040C008 00 00 00 00 00 00 01 00 ......0.
0040C010 0A 00 00 00 18 00 00 80 ....↑..Ç
0040C018 00 00 00 00 00 00 00 00 ........
0040C020 00 00 00 00 00 00 01 00 ......0.
0040C028 65 00 00 00 30 00 00 80 e...0..Ç
0040C030 00 00 00 00 00 00 00 00 ........
0040C038 00 00 00 00 00 00 01 00 ......0.
0040C040 09 04 00 00 48 00 00 00 .♦..H...
0040C048 60 80 00 00 20 00 00 00 `Ç.. ...
0040C050 00 00 00 00 00 00 00 00 ........
0040C058 00 00 00 00 00 00 00 00 ........
```

```
0012FFC4  7C816037 RETURN to ke
0012FFC8  7C910228 ntdll.7C9102
0012FFCC  FFFFFFFF
0012FFD0  7FFDA000
0012FFD4  80544DFD
0012FFD8  0012FFC8
0012FFDC  823341D0
0012FFE0  FFFFFFFF End of SEH c
0012FFE4  7C839B48 SE handler
0012FFE8  7C816040 kernel32.7C8
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  0040AF60 midterm_.<Mo
```

```
Analysing midterm_: 0 heuristical procedures          Paused
```

Figure: Before Unpacking the file


OllyDbg - midterm_problem1.exe - [CPU - main thread, module midterm_]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S

```
00401681  55               PUSH EBP
00401682  8BEC             MOV EBP,ESP
00401684  6A FF            PUSH -1
00401686  68 30514000      PUSH midterm_.00405130
0040168B  68 E82B4000      PUSH midterm_.00402BE8
00401690  64:A1 00000000   MOV EAX,DWORD PTR FS:[0]
00401696  50               PUSH EAX
00401697  64:8925 0000000  MOV DWORD PTR FS:[0],ESP
0040169E  83EC 10          SUB ESP,10
004016A1  53               PUSH EBX
004016A2  56               PUSH ESI
004016A3  57               PUSH EDI
004016A4  8965 E8          MOV DWORD PTR SS:[EBP-18],ESP
004016A7  FF15 44504000    CALL DWORD PTR DS:[405044]          kernel32.
004016AD  33D2             XOR EDX,EDX
004016AF  8AD4             MOV DL,AH
004016B1  8915 70694000    MOV DWORD PTR DS:[406970],EDX
004016B7  8BC8             MOV ECX,EAX
004016B9  81E1 FF000000    AND ECX,0FF
004016BF  890D 6C694000    MOV DWORD PTR DS:[40696C],ECX
004016C5  C1E1 08          SHL ECX,8
004016C8  03CA             ADD ECX,EDX
004016CA  890D 68694000    MOV DWORD PTR DS:[406968],ECX
004016D0  C1E8 10          SHR EAX,10
EBP=0012FFF0
```

```
Registers (FPU)
EAX 0012FF44
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFast
EBX 7FFDE000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C910228 ntdll.7C9102

EIP 00401681 midterm_.004

C 1  ES 0023 32bit 0(FFFF
P 0  CS 001B 32bit 0(FFFF
A 0  SS 0023 32bit 0(FFFF
Z 0  DS 0023 32bit 0(FFFF
S 0  FS 003B 32bit 7FFDD0
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_PROC_N

EFL 00000203 (NO,B,NE,BE,

ST0 empty -UNORM BDEC 010
ST1 empty +UNORM 0064 007
ST2 empty 0.0000016439076
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
```

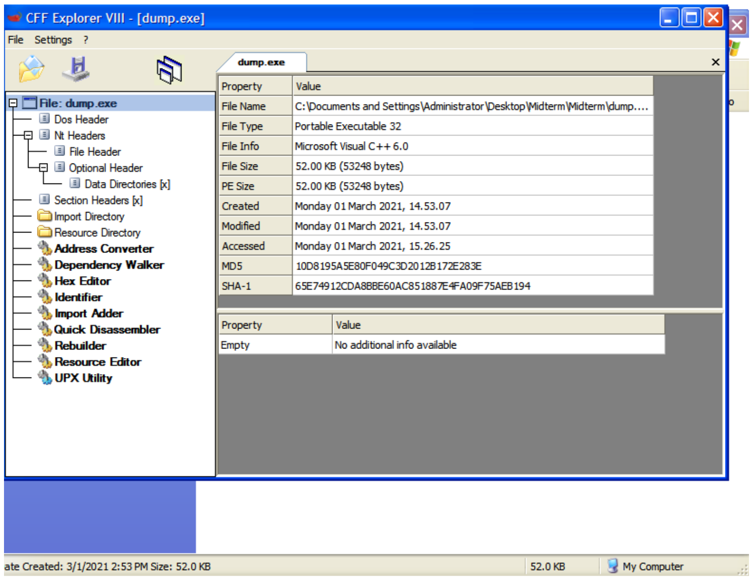
```
Address   Hex dump                 ASCII
0040C000  00 00 00 00 00 00 00 00  ........
0040C008  00 00 00 00 00 00 01 00  ......0.
0040C010  0A 00 00 00 18 00 00 80  ....†..Ç
0040C018  00 00 00 00 00 00 00 00  ........
0040C020  00 00 00 00 00 00 01 00  ......0.
0040C028  65 00 00 00 30 00 00 80  e...0..Ç
0040C030  00 00 00 00 00 00 00 00  ........
0040C038  00 00 00 00 00 00 01 00  ......0.
0040C040  09 04 00 00 48 00 00 00  .♦..H...
0040C048  60 80 00 00 20 00 00 00  `Ç.. ...
0040C050  00 00 00 00 00 00 00 00  ........
0040C058  00 00 00 00 00 00 00 00  ........
```

```
0012FFC4  7C816037  RETURN to ke
0012FFC8  7C910228  ntdll.7C9102
0012FFCC  FFFFFFFF
0012FFD0  7FFDE000
0012FFD4  80544DFD
0012FFD8  0012FFC8
0012FFDC  82490020
0012FFE0  FFFFFFFF  End of SEH c
0012FFE4  7C839B48  SE handler
0012FFE8  7C816040  kernel32.7C8
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  0040AF60  midterm_.<Mo
```

Paused

Figure: After unpacking the file.

- Once I reached the OEP. I dumped the process. You can see in the figure below, it is a plain PE executable which is VC++ compiled.



Property	Value
File Name	C:\Documents and Settings\Administrator\Desktop\Midterm\Midterm\dump....
File Type	Portable Executable 32
File Info	Microsoft Visual C++ 6.0
File Size	52.00 KB (53248 bytes)
PE Size	52.00 KB (53248 bytes)
Created	Monday 01 March 2021, 14.53.07
Modified	Monday 01 March 2021, 14.53.07
Accessed	Monday 01 March 2021, 15.26.25
MD5	10D8195A5E80F049C3D2012B172E283E
SHA-1	65E74912CDA8BBE60AC851887E4FA09F75AEB194

Property	Value
Empty	No additional info available

ate Created: 3/1/2021 2:53 PM Size: 52.0 KB   52.0 KB   My Computer

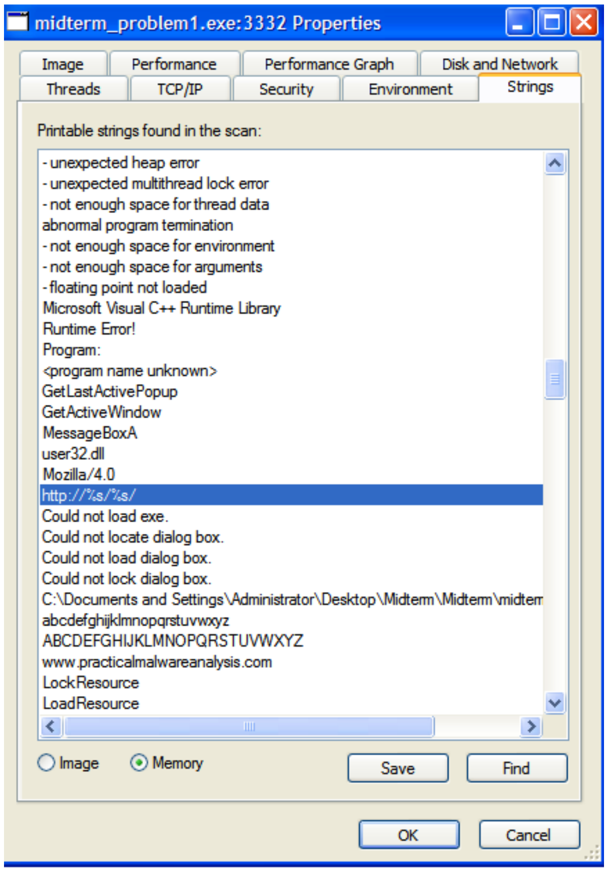
- The imports of the dumped file need to be reconstructed since the IAT is lost while dumping the process from ollydbg. “Import REconstructor” can be used to fix this. Once the imports are fixed. The dumped file which is unpacked can be analyzed. I couldn’t get the Import REconstructor tool online, so my rest of the analyses is done on the packed file.

3) What are a few strings that stick out to you and why?

Since the file is packed, strings are obfuscated. So the plain strings can be found in the memory (Shown in the figure below) when the malware is executed. Some of the strings that caught my attention are as follows:

- hxxp://%s/%s/: Looks like the malware is trying to connect to a webpage over internet. It could be a CnC server.
- www[.]practicalmalwareanalysis[.]com : Malware might be connecting to this domain to download malicious payload.
- Sleep: Malware might be using this to hide its execution from sandboxes. Since sandbox timeout after the limit that is set for them for each file execution.
- TerminateProcess: Malware might be performing anti-VM or anti-debugging checks or any other anti-reversing checks such that it's execution is terminated once the condition check satisfies.
- GetCurrentProcess: Malware might be using this to get the running process to perform process injection.
- GetVersion: Malware might be using the windows version check. Where it checks for the windows version it is running in. Most of the malwares use this check to see if the malware is being run in windows xp and terminate. Since most of the reverse engineering researchers use windows xp for analysis.
- WriteFile: Malware might be using this to over write a file. This in combination with "GetCurrentProcess", looks like handle to a running process is fetched to inject the code into it.
- GetOEMCP, GetACP, GetCPInfo: Again, malware might be using them to check the current original equipment manufacturer (OEM) code page identifier, ANSI code page identifier, available code page for the operating system. Such that, if it doesn't match an actual OS, the execution is terminated.
- InternetOpenUrlA: Another indicator that malware is connecting to internet, by opening a resource specified by http.

Looks like the malware is either trying to process inject or download payload from a compromised website. It has functions that most legitimate programs do not use.



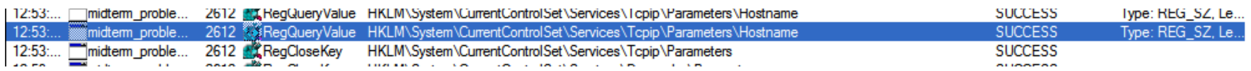
```
midterm_problem1.exe:3332 Properties

Printable strings found in the scan:
- unexpected heap error
- unexpected multithread lock error
- not enough space for thread data
abnormal program termination
- not enough space for environment
- not enough space for arguments
- floating point not loaded
Microsoft Visual C++ Runtime Library
Runtime Error!
Program:
<program name unknown>
GetLastActivePopup
GetActiveWindow
MessageBoxA
user32.dll
Mozilla/4.0
http://%s/%s/
Could not load exe.
Could not locate dialog box.
Could not load dialog box.
Could not lock dialog box.
C:\Documents and Settings\Administrator\Desktop\Midterm\Midterm\midterm
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
www.practicalmalwareanalysis.com
LockResource
LoadResource
```

Figure: Strings from memory

4) What happens when you run this malware? Is it what you expected and why?

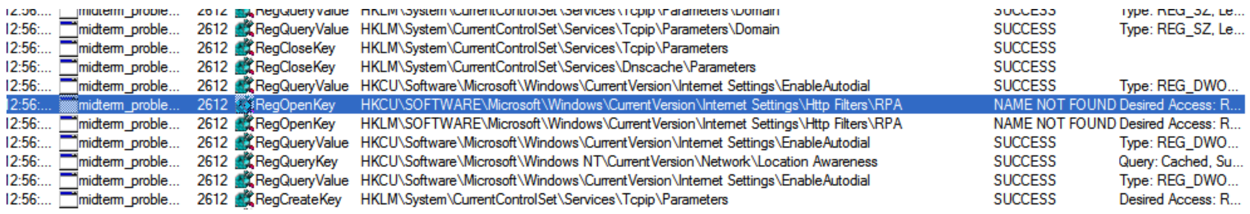
When the malware is executed, a console GUI is displayed. But in the background, it does the following:

- Host name is read using "RegQueryValue" at "HKLM\System\CurrentControlSet\Services\Tcpip\Parameters\Hostname". Which is an indicator that malware is trying to get the user details and check if the file is run in VM. Since, most of the times a virtual machine contains host name like "vmware". So, the malware is trying to check if the OS is of legitimate user or not.



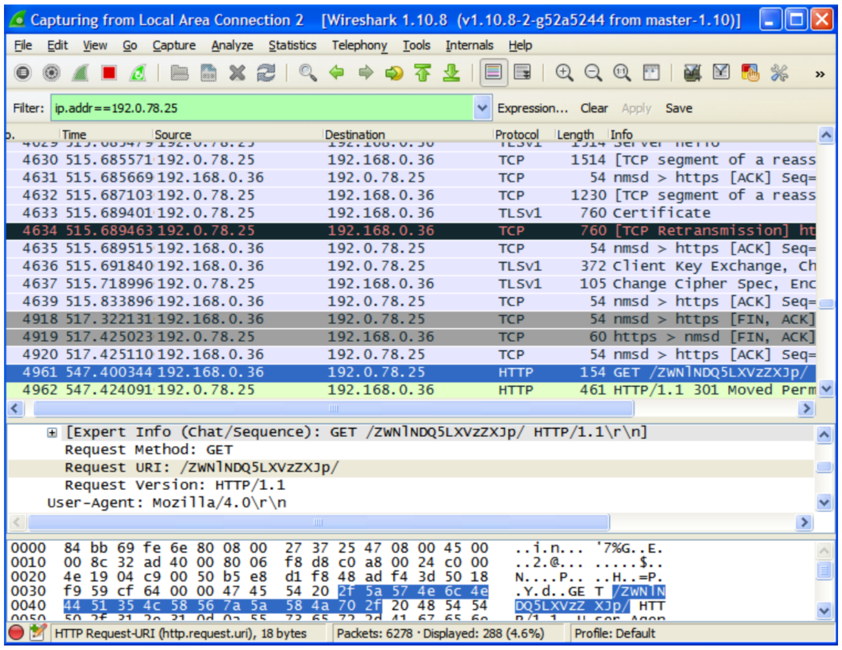
```
12:53:...  midterm_proble...  2612  RegQueryValue  HKLM\System\CurrentControlSet\Services\Tcpip\Parameters\Hostname  SUCCESS  Type: REG_SZ, Le...
12:53:...  midterm_proble...  2612  RegQueryValue  HKLM\System\CurrentControlSet\Services\Tcpip\Parameters\Hostname  SUCCESS  Type: REG_SZ, Le...
12:53:...  midterm_proble...  2612  RegCloseKey    HKLM\System\CurrentControlSet\Services\Tcpip\Parameters           SUCCESS
```

- "HKLM\System\CurrentControlSet\Services\Tcpip\Parameters\Domain" is queried, along with "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Http Filters\RPA" and "HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\EnableAutodial" thus indicating that the malware is indeed making connections through this domain over the internet.



```
12:56:... midterm_proble... 2612 RegQueryValue HKLM\System\CurrentControlSet\Services\Tcpip\Parameters\Domain                          SUCCESS          Type: REG_SZ, Le...
12:56:... midterm_proble... 2612 RegQueryValue HKLM\System\CurrentControlSet\Services\Tcpip\Parameters\Domain                          SUCCESS          Type: REG_SZ, Le...
12:56:... midterm_proble... 2612 RegCloseKey   HKLM\System\CurrentControlSet\Services\Tcpip\Parameters                                 SUCCESS
12:56:... midterm_proble... 2612 RegCloseKey   HKLM\System\CurrentControlSet\Services\Dnscache\Parameters                              SUCCESS
12:56:... midterm_proble... 2612 RegQueryValue HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\EnableAutodial          SUCCESS          Type: REG_DWO...
12:56:... midterm_proble... 2612 RegOpenKey    HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Http Filters\RPA       NAME NOT FOUND Desired Access: R...
12:56:... midterm_proble... 2612 RegOpenKey    HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Http Filters\RPA       NAME NOT FOUND Desired Access: R...
12:56:... midterm_proble... 2612 RegQueryValue HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\EnableAutodial          SUCCESS          Type: REG_DWO...
12:56:... midterm_proble... 2612 RegQueryKey   HKCU\Software\Microsoft\Windows NT\CurrentVersion\Network\Location Awareness            SUCCESS          Query: Cached, Su...
12:56:... midterm_proble... 2612 RegQueryValue HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\EnableAutodial          SUCCESS          Type: REG_DWO...
12:56:... midterm_proble... 2612 RegCreateKey  HKLM\System\CurrentControlSet\Services\Tcpip\Parameters                                 SUCCESS          Desired Access: R...
```

- HTTP request is made to "192.0.78.25", "192.0.78.25" to GET data from CnC. Attacker, now has access to the victims' machine through this IP's and can use this link to infiltrate and exfiltrate from the victims' machine in a later time.



Capturing from Local Area Connection 2 [Wireshark 1.10.8 (v1.10.8-2-g52a5244 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: ip.addr==192.0.78.25
Expression... Clear Apply Save

```
No.  Time       Source       Destination  Protocol Length Info
4630 515.685571 192.0.78.25  192.168.0.36 TCP      1514 [TCP segment of a reass
4631 515.685669 192.168.0.36 192.0.78.25  TCP        54 nmsd > https [ACK] Seq=
4632 515.687103 192.0.78.25  192.168.0.36 TCP      1230 [TCP segment of a reass
4633 515.689401 192.0.78.25  192.168.0.36 TLSv1     760 Certificate
4634 515.689463 192.0.78.25  192.168.0.36 TCP       760 [TCP Retransmission] ht
4635 515.689515 192.168.0.36 192.0.78.25  TCP        54 nmsd > https [ACK] Seq=
4636 515.691840 192.168.0.36 192.0.78.25  TLSv1     372 Client Key Exchange, Ch
4637 515.718996 192.0.78.25  192.168.0.36 TLSv1     105 Change Cipher Spec, Enc
4639 515.833896 192.168.0.36 192.0.78.25  TCP        54 nmsd > https [ACK] Seq=
4918 517.322131 192.168.0.36 192.0.78.25  TCP        54 nmsd > https [FIN, ACK]
4919 517.425023 192.0.78.25  192.168.0.36 TCP        60 https > nmsd [FIN, ACK]
4920 517.425110 192.168.0.36 192.0.78.25  TCP        54 nmsd > https [ACK] Seq=
4961 547.400344 192.168.0.36 192.0.78.25  HTTP      154 GET /ZWNlNDQ5LXVzZXJp/
4962 547.424091 192.0.78.25  192.168.0.36 HTTP      461 HTTP/1.1 301 Moved Perm

  ⊞ [Expert Info (Chat/Sequence): GET /ZWNlNDQ5LXVzZXJp/ HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /ZWNlNDQ5LXVzZXJp/
    Request Version: HTTP/1.1
  User-Agent: Mozilla/4.0\r\n

0000  84 bb 69 fe 6e 80 08 00  27 37 25 47 08 00 45 00   ..i.n... '7%G..E.
0010  00 8c 32 ad 40 00 80 06  f8 d8 c0 a8 00 24 c0 00   ..2.@... .....$..
0020  4e 19 04 c9 00 50 b5 e8  d1 f8 48 ad f4 3d 50 18   N....P.. ..H..=P.
0030  f9 59 cf 64 00 00 47 45  54 20 2f 5a 57 4e 6c 4e   .Y.d..GE T /ZWNlN
0040  44 51 35 4c 58 56 7a 5a  58 4a 70 2f 20 48 54 54   DQ5LXVzZ XJp/ HTT

HTTP Request-URI (http.request.uri), 18 bytes   Packets: 6278 · Displayed: 288 (4.6%)   Profile: Default
```

I initially thought it might be a trojan downloader because it showed some domain names in the strings. But on execution it doesn't show any traces of file being added on disk. What makes sense out of the information available is that, a lot of victim data is queried which is then sent over to CnC and the victims' machine listens on the port for attackers' further instructions.

5) Are there any host-based signatures? (Files, registry keys, processes or services, etc). If so, what are they?

Yes, Following can be used to write host-based signatures:

"HKLM\System\CurrentControlSet\Services\Tcpip\Parameters\Hostname" – Querying the hostname of victim machine. (Getting user information).

"C:\Users\admin\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\644B8874112055B5E195ECB0E8F243A4" – Malware adds binary value to this location.

"C:\Users\admin\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\E49827401028F7A0F97B5576C77A26CB_7CE95D8DCA26FE957E7BD7D76F353B08" – Malware adds binary value to this location.

These combined with network-based signatures will produce a strong signature.

6) Are there any network-based signatures? (URLs, packet contents. etc) If so, what are they?

Yes, following can be used to write network-based signature:

- "hxxp://%s/%s/" : This string indicates there's a network activity.
- "hxxp://www[.]practicalmalwareanalysis[.]com/yw5hbhlzddatmmqx/" : This is the URL in the packet content when the victims machine connects to "192.0.78.25"
- "192.0.78.25" : Malware attempts to connect to this IP which has a reputation of being malicious

While these attributes individually might not make a strong signature. Them combined with host-based attributes will give a strong signature.

7) Is there anything that impeded your analysis? How so? How might you overcome this?

Yes, the malware showed no network activity directly with the domain name "hxxp://www[.]practicalmalwareanalysis[.]com/yw5hbhlzddatmmqx/"which was found in the strings, on my virtual machine. I checked if the process was running with a different name to mask its activity, but turns out this was because the malware used IP addresses to connect to CnC. To get more network activity that might not be seen in my VM since the malware has a few anti-vm checks, I ran the malware in Cuckoo sandbox provided my VirusTotal and checked the network activity through "Relations" and "behavior" tab of it. From it I could infer that the malware is actually making connection to, "192.0.78.25" which have the reputation of being malicious.

8) What do you think is the purpose of this malware?

Based on the static and dynamic analysis, this malware has a generic behavior of trojan. Since it's connecting to malicious IP. The IP is a compromised (Virus total shows 2 anti-virus engines reporting it as malicious). I feel this malware shows generic behavior of being a trojan, since it opens a lot of registry values and exfiltrates information from the victim. Such as his hostname details, OS details to CnC server. The purpose of this malware might just be to give the control of the victim machine (like backdoor) to attacker such that it becomes a part of attacker's botnet. Which can be used by the attacker to perform various other DDOS attacks later.

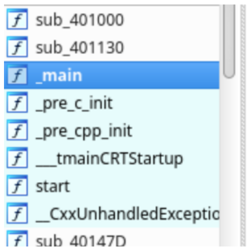
2. Answer the following questions based on your analysis of Problem 2.

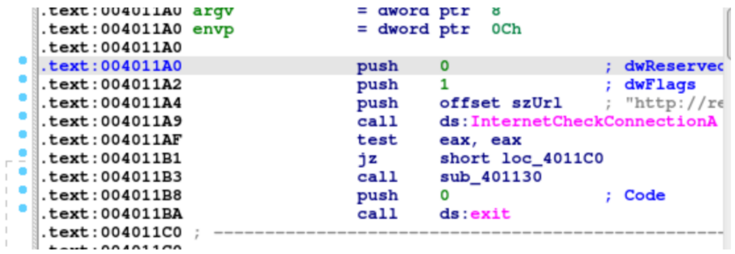
Problem 2 SHA256: 0d6ca46a1d62c6a7fcadb184aee9f06444e7f80fa6098826b750933e2af1b393

1) Main function:

a. What is the address of main?

When the file is loaded in IDApro. Main function on the left can be selected to get the address of main. Here, address of main is 004011A0





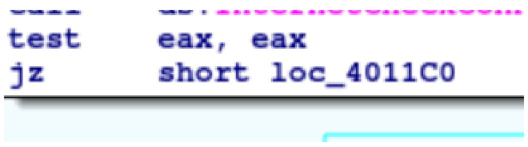
```
.text:004011A0 argv            = dword ptr  8
.text:004011A0 envp            = dword ptr  0Ch
.text:004011A0
.text:004011A0                 push    0               ; dwReservec
.text:004011A2                 push    1               ; dwFlags
.text:004011A4                 push    offset szUrl    ; "http://re
.text:004011A9                 call    ds:InternetCheckConnectionA
.text:004011AF                 test    eax, eax
.text:004011B1                 jz      short loc_4011C0
.text:004011B3                 call    sub_401130
.text:004011B8                 push    0               ; Code
.text:004011BA                 call    ds:exit
.text:004011C0 ; ---------------------------------------------------
```

b. What does this function do?

Here in the main function, dwflags, dwReserved, "hxxp://reversing[.]rocks/" are pushed as arguments to InternetCheckConnectionA, which is then used to check if the malware is having connection over internet with "hxxp://reversing[.]rocks/". Then jumps to loc_4011C0 if it is already connected. If not, sub_401130 is called.

i. What code constructs are used in this function?

Code construct used here is "if else" in high-level language. Since it corresponds to "test" and "jz" in assembly code of this function and only one "test" "jz" combination is found



```
test    eax, eax
jz      short loc_4011C0
```

ii. Are there any interesting strings? If so, what are they?

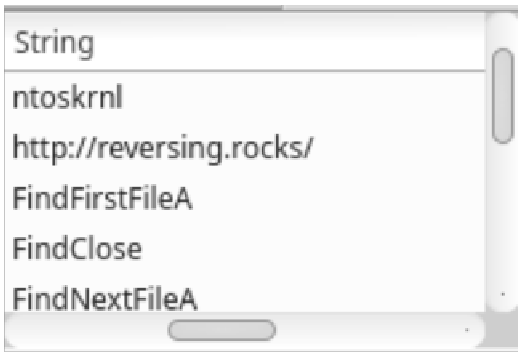
"hxxp://reversing[.]rocks/" : InternetCheckConnectionA is used to check if the malware is having connection over internet with "hxxp://reversing[.]rocks/" in order to establish a connection.

"_crtTerminateProcess": Malware might be terminating itself after execution or once it detects that the file is reverse engineered. To hide its actual purpose.

"IsDebuggerPresent": Checks if a debugger is present in the OS. Could be used as anti-reversing technique.

"HttpSendRequestExA": Looks like the malware is requesting data over the internet through a domain.

"FindFirstFileA", "FindNextFileA", "FindClose": Malware might be checking active processes for performing process injection.



```
String
ntoskrnl
http://reversing.rocks/
FindFirstFileA
FindClose
FindNextFileA
```

2) Looking at the subroutine a 0x00401130:

a. What are the arguments to InternetConnectA? What do they mean?

Argument passed to "InternetConnectA" are as follows:

0 – hardcoded value of 0 is passed

0 – dwFlag value is passed as 0. So, in this case service used is FTP.

3 – dwService value is passed as 3, it indicates the type of service to access which in this case is HTTP service (cause the argument passed is 3).

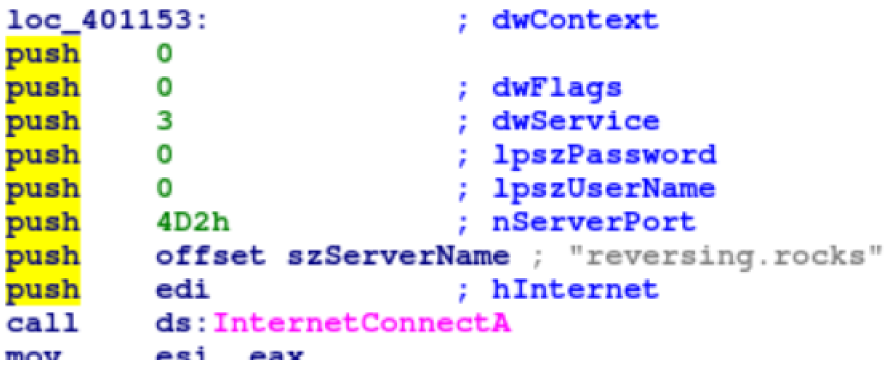
0 – lpszPassword value is passed as 0, which is pointer to a string containing the password to log-on. Since here, the value is NULL, it is possible that either the default password is used which is user's email name or a blank password is used.

0 – lpszUserName value is passed as 0. Since the value is NULL, the function uses the default password.

4D2h – Its decimal equivalent value is "1234". Which is the server port that's used to make the connection.

Offset szServerName – Memory location of the server name to which the connection needs to be made is passed. In this case it's "reversing.rocks"

EDI – Handle returned by InternetOpen is passed.



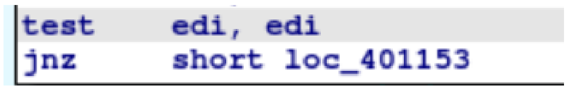
```
loc_401153:                 ; dwContext
push    0
push    0                   ; dwFlags
push    3                   ; dwService
push    0                   ; lpszPassword
push    0                   ; lpszUserName
push    4D2h                ; nServerPort
push    offset szServerName ; "reversing.rocks"
push    edi                 ; hInternet
call    ds:InternetConnectA
mov     esi, eax
```

b. What does this function do?

The purpose of this function is to open a HTTP session for "reversing.rocks" server site and connect to it once the session is successfully opened. If not, the control flow is shifted to the end of the program. Once the connection is established, all the handles used for the connection are closed.

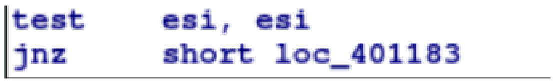
i. What code constructs are used in this function?

Code construct used here is "nested if" since, there are conditional jumps and the initial if loop contains another if loop inside it. Initial loop instruction is the first figure below and 2nd loop instruction is the 2nd figure below.



```
test    edi, edi
jnz     short loc_401153
```

Figure: First "if" condition



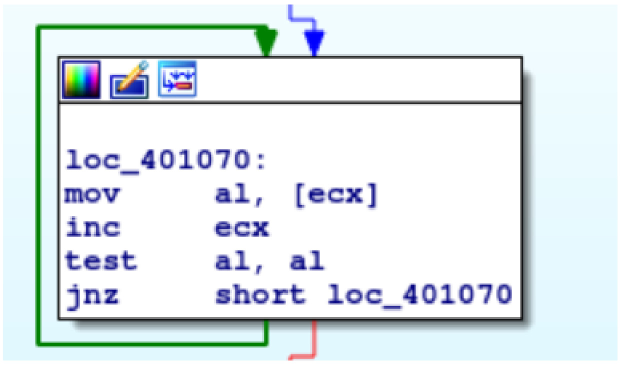
```
test    esi, esi
jnz     short loc_401183
```

Figure: Second "if" condition

3) Looking at the subroutine at 0x00401000:

a. What code constructs are used in this function?

Code constructs used here are "Nested if" and "Nested for loop". Since there are multiple conditional jumps and "test" conditions under the main "if" along with "Nested for loops" in between. One of the "for loop" is shown below.

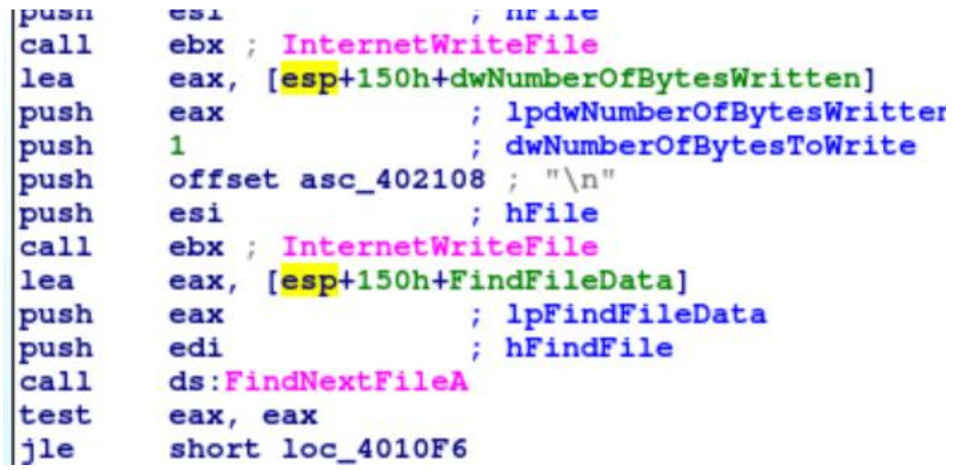

```
loc_401070:
mov     al, [ecx]
inc     ecx
test    al, al
jnz     short loc_401070
```

b. What imported functions are called?

Following import functions are called:

“FindFirstFileA”, “HttpOpenRequestA”, “HttpSendRequestExA”, “InternetWriteFile”, “FindNextFileA”, “InternetWriteFile” ,”FindNextFileA”, “HttpEndRequestA”, “InternetCloseHandle”, “FindClose”.

A few are shown below in the figure:


```
call    ebx ; InternetWriteFile
lea     eax, [esp+150h+dwNumberOfBytesWritten]
push    eax                 ; lpdwNumberOfBytesWritter
push    1                   ; dwNumberOfBytesToWrite
push    offset asc_402108 ; "\n"
push    esi                 ; hFile
call    ebx ; InternetWriteFile
lea     eax, [esp+150h+FindFileData]
push    eax                 ; lpFindFileData
push    edi                 ; hFindFile
call    ds:FindNextFileA
test    eax, eax
jle     short loc_4010F6
```

c. What does this subroutine do?

In this subroutine, it initially searches for a directory or subdirectory with a specific name. If found the control flow reaches the end of the program. Here, the sample is checking if the payload already exists in victim machine. If not found, it opens a handle to http request, then sends “POST” request to server through which it writes to internet opened file. Once done, it continues to search for a directory or subdirectory with a specific name. If found, handle to the internet is closed. If not found, it keeps repeating the whole processes until the file is found.

4) What does this malware do?

Apart from some suspicious imports and one URL. The sample doesn’t show any strong host and network artifacts. The URL itself doesn’t have any malicious reputation nor the sample does. In my opinion, the sample is not a malware.