Faculty of Business, IT, and Management
HACK2200 Hacking and Exploits
Lab 2: Malware and Viruses[1]

WARNING

The binaries in this lab contain malicious code and live malware, you should NOT install or run these programs without first setting up a safe environment. To do so make sure to remove the network card from the Windows 10 Virtual Machine (WIN10VM) when you are instructed to do so below.

Once this lab is completed, destroy the WIN10VM instance, we will not be using this instance again.

Instructions

- This lab should be completed individually.
- This lab is designed for the purpose of education and training, but not for any illegal activities including hacking. Beware to only use these exploits on hosts that you have permission to hack.
- When a question asks for screenshots, your screenshots **must**:
    - Include the full window (the application window, or the terminal window, etc.),
    - have the PROMPT setup as per the instructions, including the date and time in the same format provided in the instructions. Screenshots without the prompt setup will receive zero credit,
    - be clearly readable,
    - include all the information required by the question, and
    - not include extra commands, failed attempts, and/or error messages.
- Failure to follow submission instructions will result in marks deduction. There will be marks deduction for including more than what is required in the instructions. Do not provide any screenshots that are not required in the instructions.
- The below instructions are guidelines, you are expected to troubleshoot any errors you run into.
- Read and complete the assignment instructions below and finish all the tasks. Provide screenshots and answer the questions in the provided answer file.

Lab Setup

- Note if you already have a Windows 10 VM that you don't need anymore, you may use it. However, we will dispose of the Windows 10 VM at the end of this lab.
- If you don't have a Windows 10 VM that you can dispose of, then build a new one, by following these steps:

[1] This lab is designed based on 1- Hands-On Ethical Hacking and Network Defense, by Michael Simpson, Nicholas Antill, 3rd Edition, 2016. And 2- Practical Malware Analysis By: Michael Sikorski and Andrew Honig.

1. Download Windows 10 https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/
2. Select MSEdge on Win10 (x64) Stable 1809 and the platform you prefer.
3. Start the virtual machines you just downloaded. We will refer to this machine as Win10VM during this lab.

In this lab, we will explore:

Part 1 – Viruses

Part 2 – Malware Analysis

Part 1 – Viruses

In this part, we will write a simple virus. A virus is a program that attaches itself to a file or another program, often sent via e-mail. The keyword is "attaches." A virus doesn't stand on its own, so it can't replicate itself or operate without the presence of a host. A virus attaches itself to a host file or program (such as Microsoft Word).

A fork bomb is an attack where a process continuously repeats itself and consumes the system's resources. A fork bomb does not harm any files on the system. However, it slows down or crashes the system. You can create a fork bomb using a batch file and execute it. You can create batch files to perform malicious tasks such as deleting system files, creating backdoors, and so on.

Consider an example of a batch file that will delete all the files in the Windows operating system's System32 directory. The given code on execution can result in damage to your system and it may require extensive time and skill to fix the system.

```
@echo off
Del c:\windows\system32\*.*
Del c:\windows\*.*
```

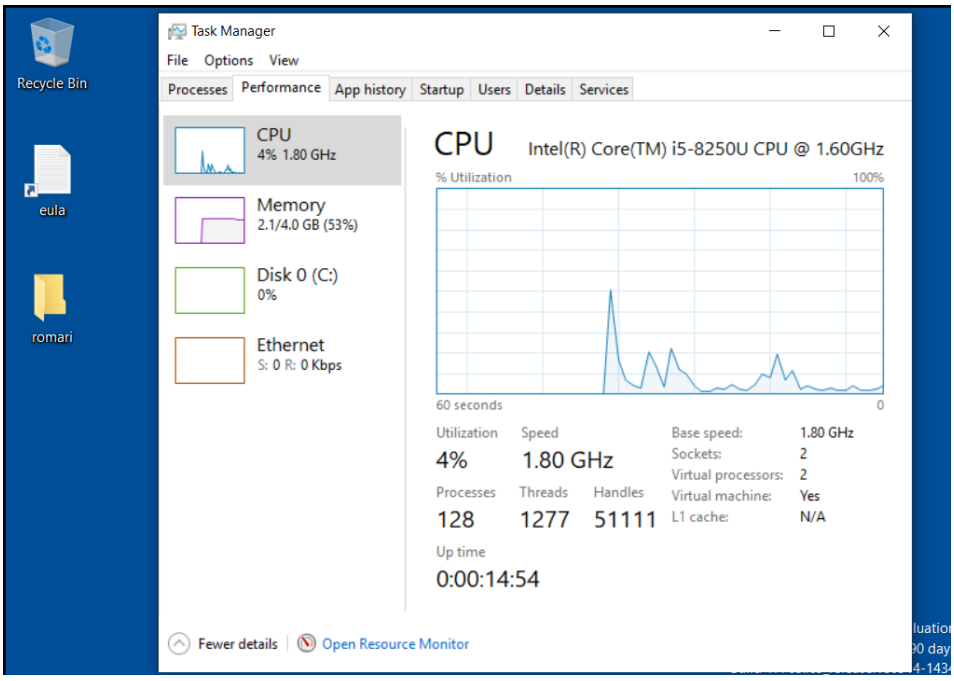
The @echo off command will disable the Command Prompt from being shown and will execute the batch file in the back end.

In this exercise, you will learn how to create a fork bomb and execute it.

To create a fork bomb using a batch file and execute it, follow the steps below:

Step 1 - Power on your Windows 10 VM (Win10VM), create a new folder on the desktop and call it *yourfirstname* (replace *yourfirstname* with your first name), similar to the screen sample below.

Step 2 - Launch the **Task Manager** on your Win10VM, and browse to the **Performance** tab, so you can observe the working of the fork bomb execution.



Step 3- In **Notepad**, create a new batch file and type the following fork bomb code:

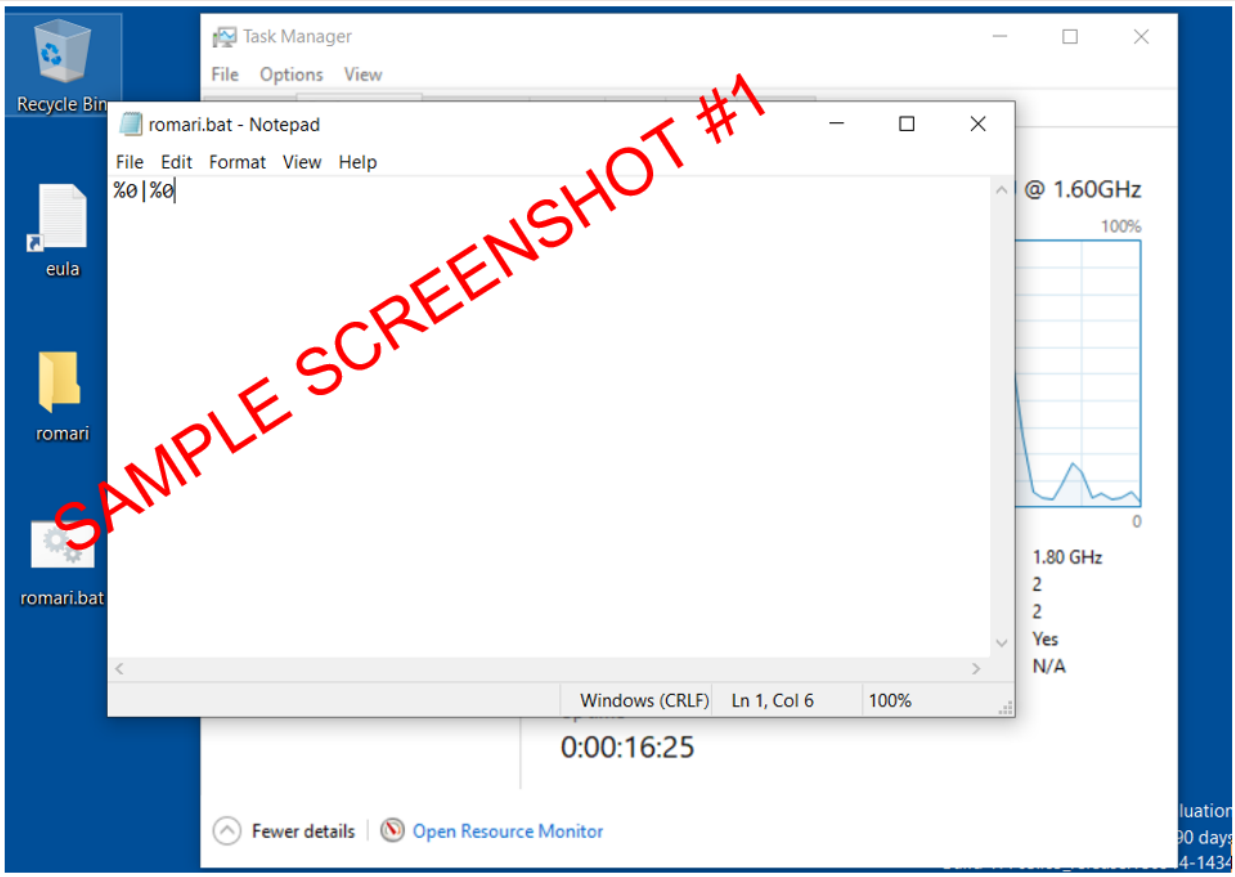
```
%0|%0
```

A batch file contains instructions to be executed in sequence. In this batch file, %0 is the name of the currently executing batch file. This batch file is going to recursively execute itself forever. It quickly creates many processes and slows down the system.

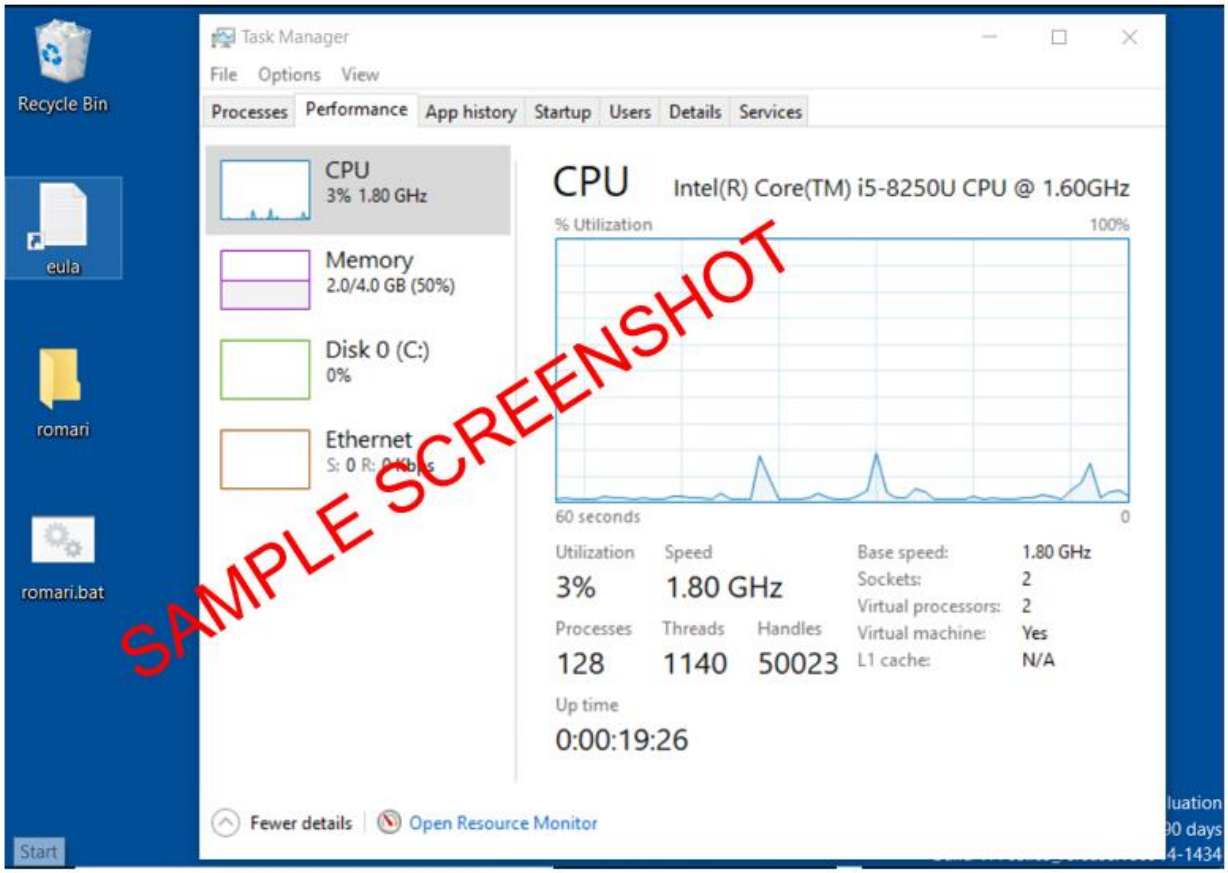
Save the notepad file to the Desktop, and name it as

yourfirstname.bat

Take a screenshot similar to the one below, and place it in the answer file under Screenshot#1.

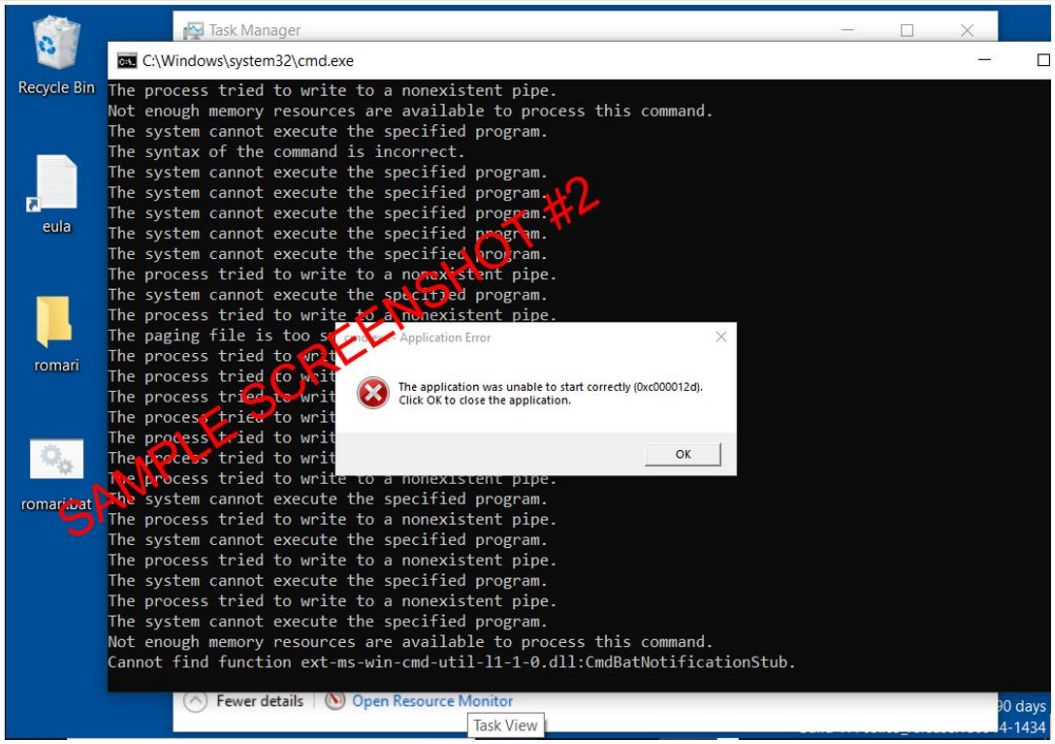


Step 4 - Before you execute the batch file, observe the CPU usage under the performance tab in the **Task Manager**. The screenshot below shows CPU utilization at 3% (your system might be slightly different).

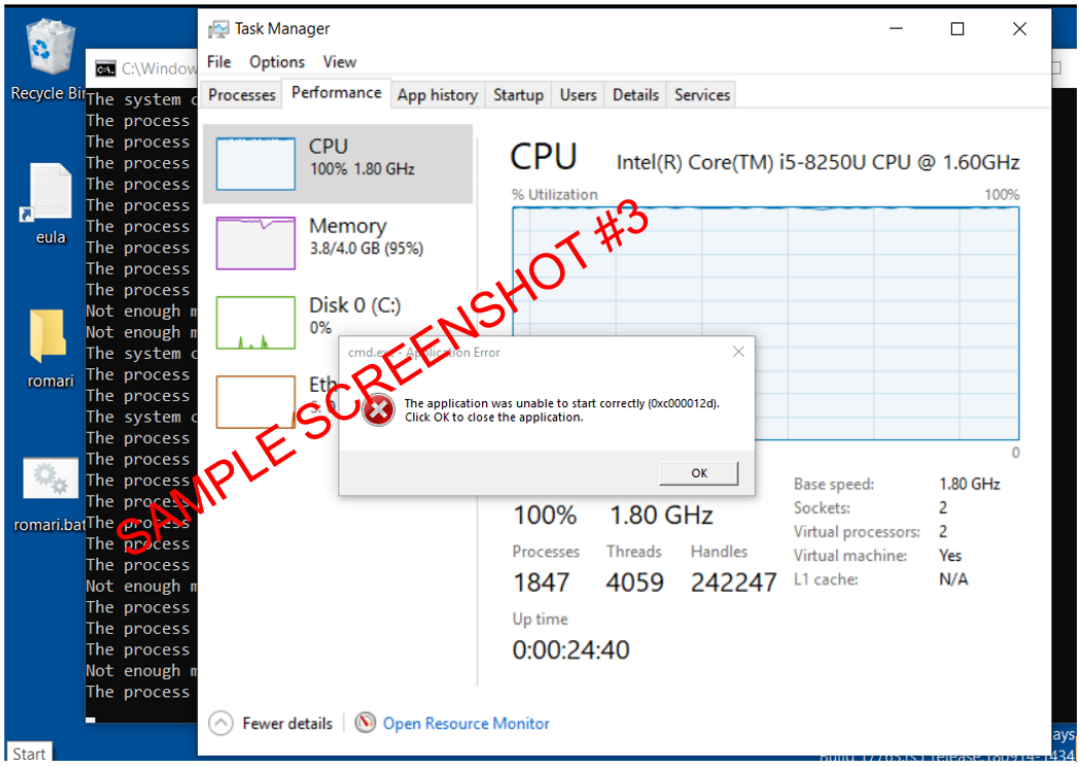


To execute the forkbomb file, on the desktop, right-click the forkbomb (*yourfirstname.bat*), and click **Open**.

The **Command Prompt** window opens and the forkbomb.bat (*yourfirstname.bat*) file starts executing recursively. Take a screenshot similar to the one below, and place it under Screenshot#2 in the answer file.



Click the Task Manager Window and note the CPU Utilization. Take a screenshot to replace the one below, and place it under Screenshot#3 in the answer file.



Note that the Win10VM would hang and could crash itself. You will probably have to reboot the machine.

Congratulations, you just created your first forkbomb!

End of Part 1

Part 2 – Malware Analysis

We will be using Malware and Trojan analysis tools to reverse engineer a malware .dll file. Every day, thousands of malware are released by hackers. These malware are designed to work in different ways. Some are not harmful and cause minimal damage and others are extremely chaotic in nature. You can use different types of malware and Trojan analysis tools to ensure their impact is reduced or minimized.

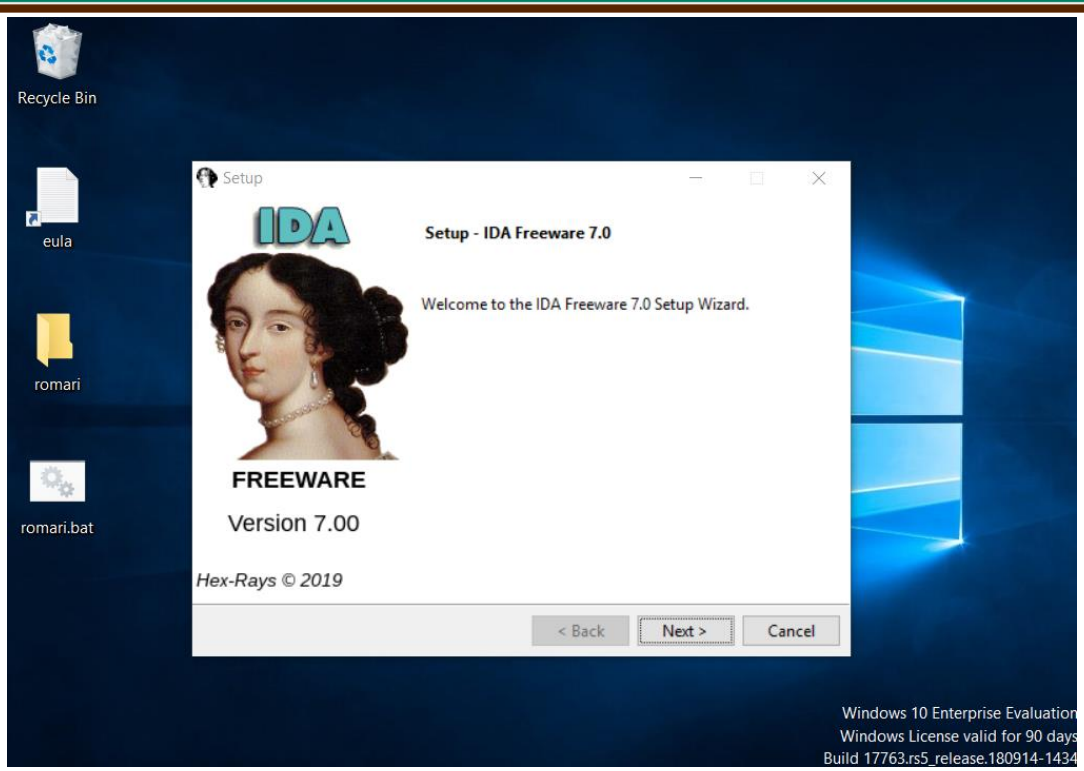
PEView or the **Interactive Disassembler**, is a software that can help you disassemble binary programs. If you do not have the source code of an executable, you can disassemble it to know its map of execution. This can be especially useful when you need to disassemble a virus to find its method of execution. In this exercise, you will learn how to use IDA Pro.

In this lab, we will be using IDA Disassembler. Read through the IDA General purpose tutorials available here: https://www.hex-rays.com/products/ida/support/tutorials/unpack_pe/manual/

Now, let's install IDA and analyse a malware file.

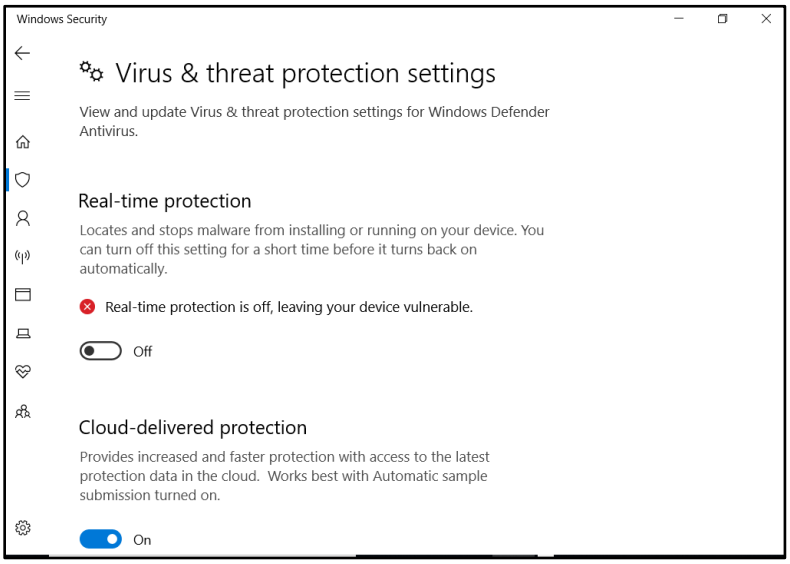
Step 1 - On your Windows 10 virtual machine, download and install **IDA 7.0 Freeware For Windows** (for non-commercial use) available here: https://www.hex-rays.com/products/ida/support/download/

Note this lab is created with IDA 7.0. There may be updated versions of IDA Freeware when you start working on this lab. You should use the latest one available on the website and work your way around the instructions to achieve the same goals with the new version. The principles of malware analysis and reverse engineering described here are the same, and so if you can't follow a step due to changes in the updated version, find a new way to perform the step in the new version.

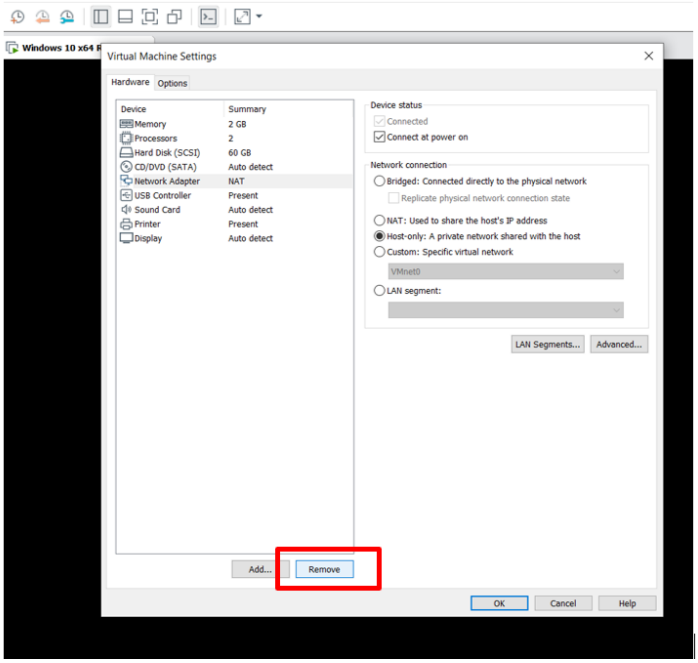


Step 2 – On your Windows 10 virtual machine **Win10VM** download (do not execute or open, just download) the *PracticalMalwareAnalysis-Labs.exe* available at this location: https://github.com/mikesiko/PracticalMalwareAnalysis-Labs

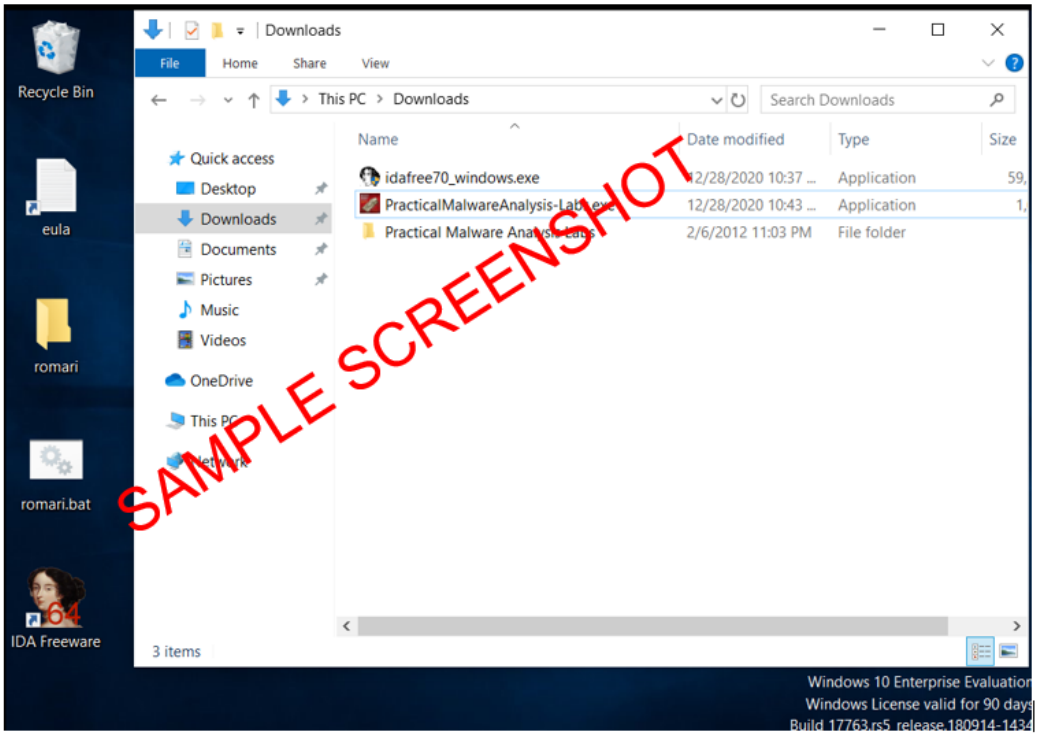
If your **Win10VM** antivirus blocks the download, disable the antivirus software real-time protection.



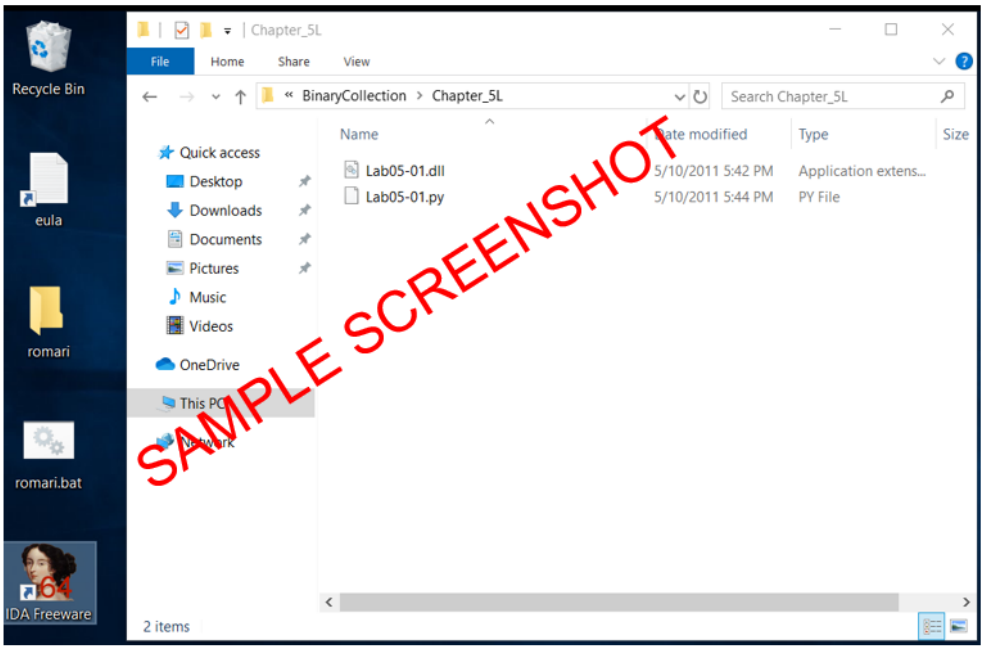
Step 3 – Once you have downloaded the PracticalMalwareAnalysis-Labs.exe file, disconnect your Win10VM from the network. You can do that by going to VM-> Settings-> and remove the network adapter (NIC) as shown below.



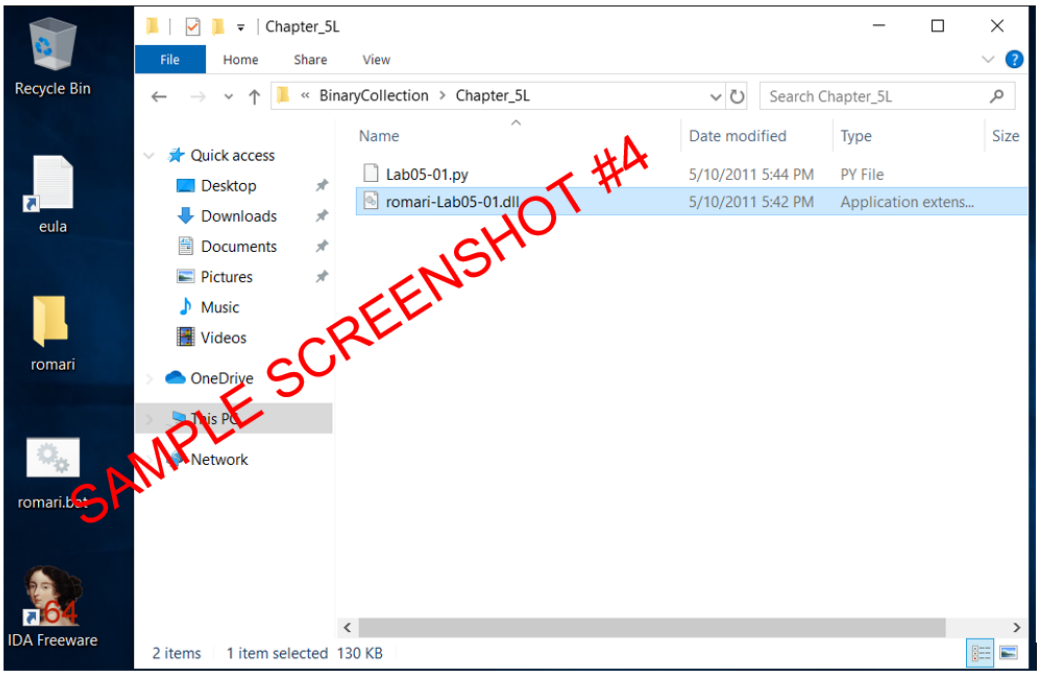
Step 4 - Run the *PracticalMalwareAnalysis-Labs* file. A new folder with the name *Practical Malware Analysis Labs* should appear as below.



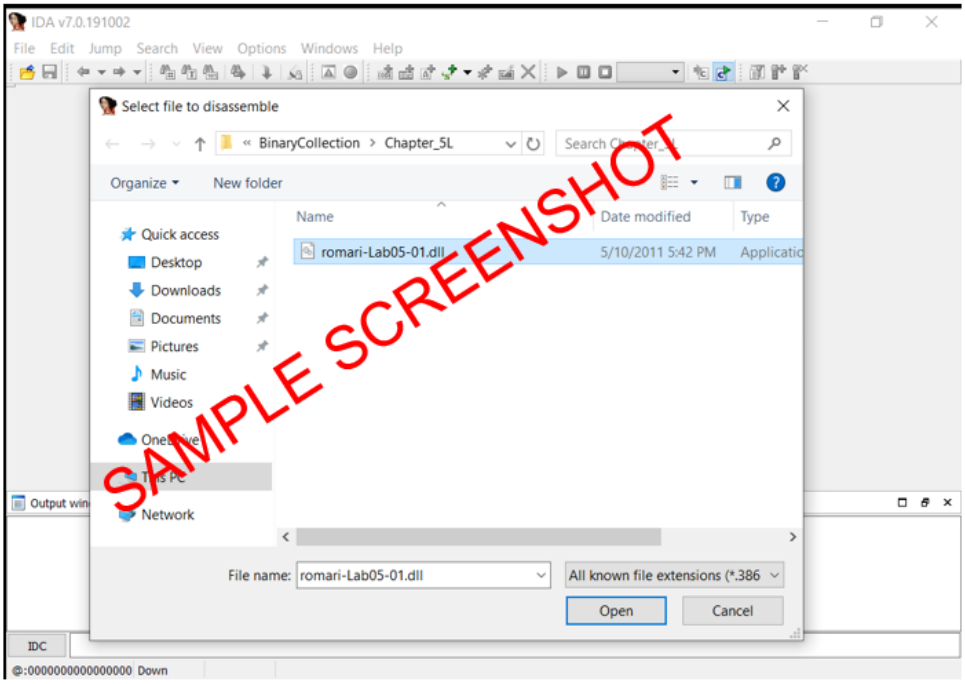
In this exercise, we will be using the `Lab05-01.dll` shown below.



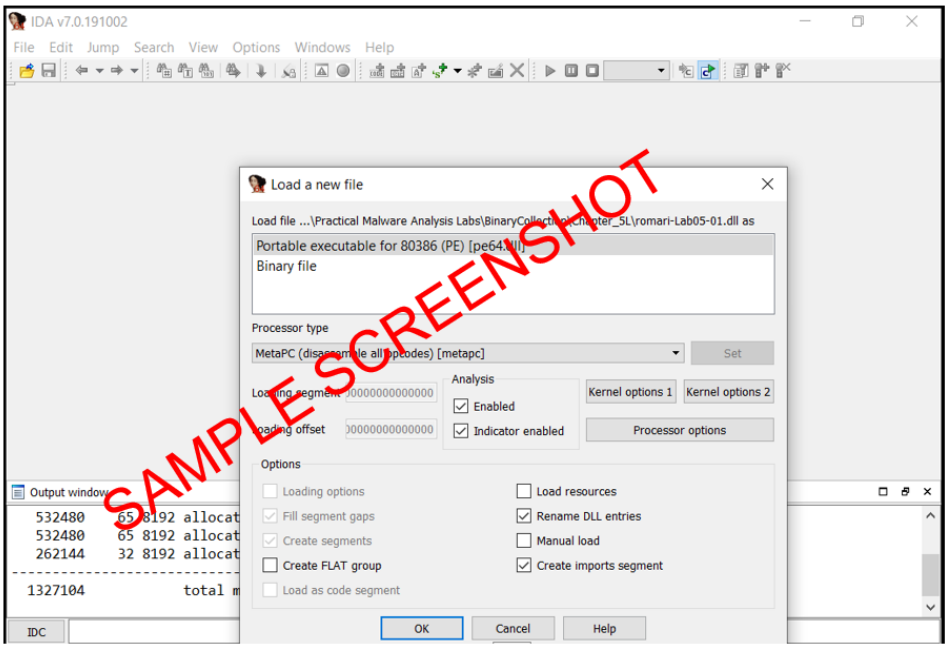
Right-click the `Lab05-01.dll` file and rename it to include your first name. The .dll file name should now be `yourfirstname-Lab05-01.dll`. In the screenshot below, I renamed my file to be `romari-Lab05-01.dll`. Take a screenshot to replace the one below, and place it under Screenshot#4 in the answer file.



Step 5 – Launch IDA, and choose **New – Disassemble a new file**. Browse to *Chapter_5L* in your *Practical Malware Analysis Labs* folder and choose *yourfirstname-Lab05-01.dll* file as shown below:



Click **OK**



Step 6 – Now you are ready to analyze the malware .dll file. Answer the following questions, and replace the provided screenshots with the ones you create.

Questions 1. What is the address of `DllMain`?

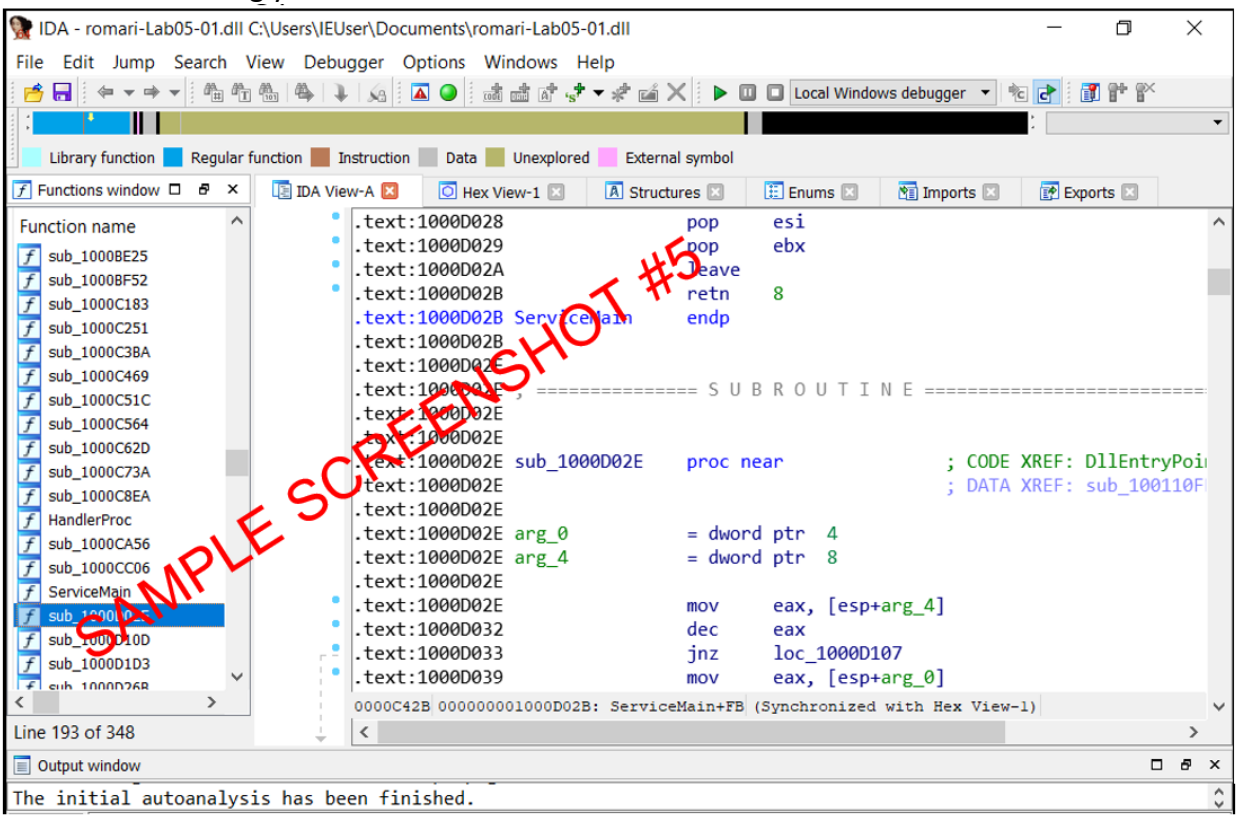
Once we load the malicious DLL into IDA, `DllMain` is located at the address of the first subroutine starting after the `ServiceMain` function/procedure.

To answer question 1, locate the `ServiceMain` function, and highlight the address of the `DllMain` that is right after it.

You may need to display line numbers in the graph view by using **Options** ▸ **General** and checking **Line Prefixes**, and you can toggle between the graph and traditional view by pressing the spacebar, which allows you to see the line numbers without changing the options.

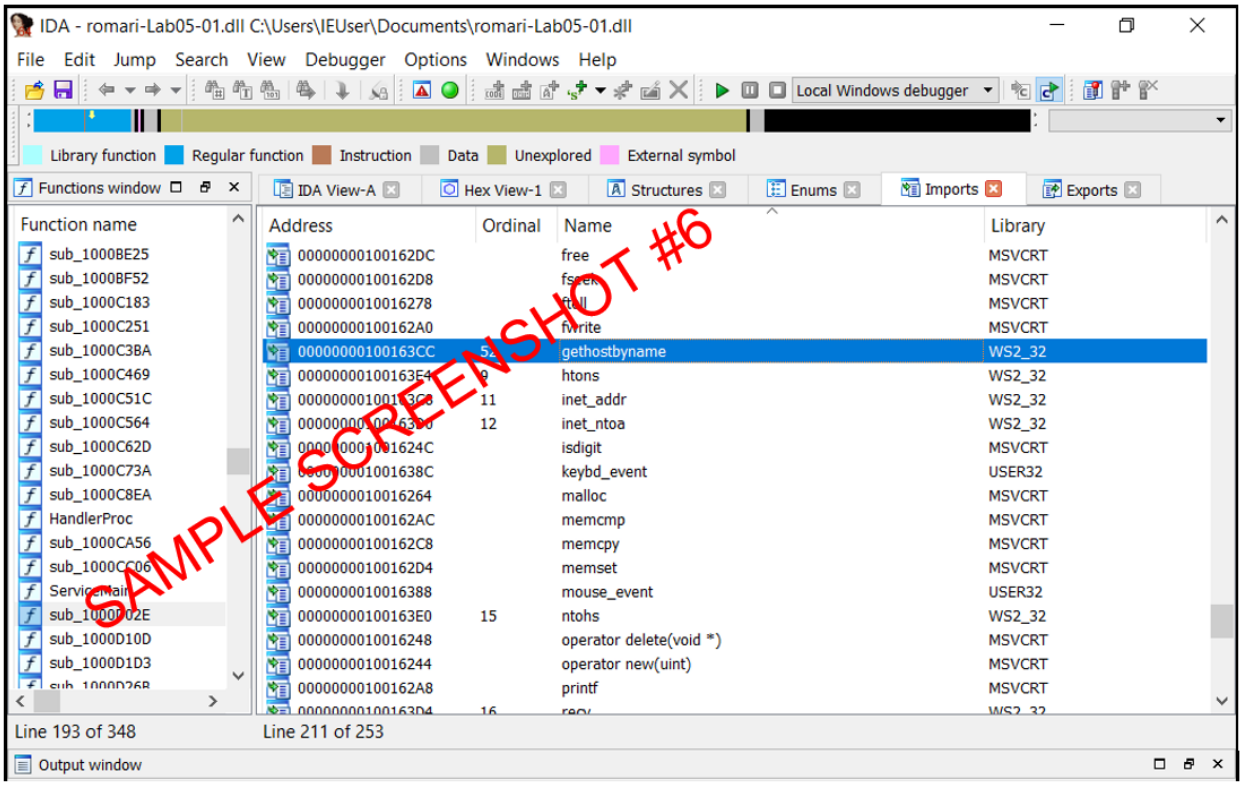
`DllMain` is where we want to begin the analysis, because all code that executes from the `DllEntryPoint` until `DllMain` has likely been generated by the compiler, and we don't want to get bogged down analyzing compiler-generated code.

In the screenshot below, the address of the `DllMain` is at `0x1000D02E`.
Take a screenshot to replace the one below, and place it under Screenshot#5 in the answer file. Ensure your file name is showing in your screenshot. There will be zero credit for screenshots that are not showing your file name.



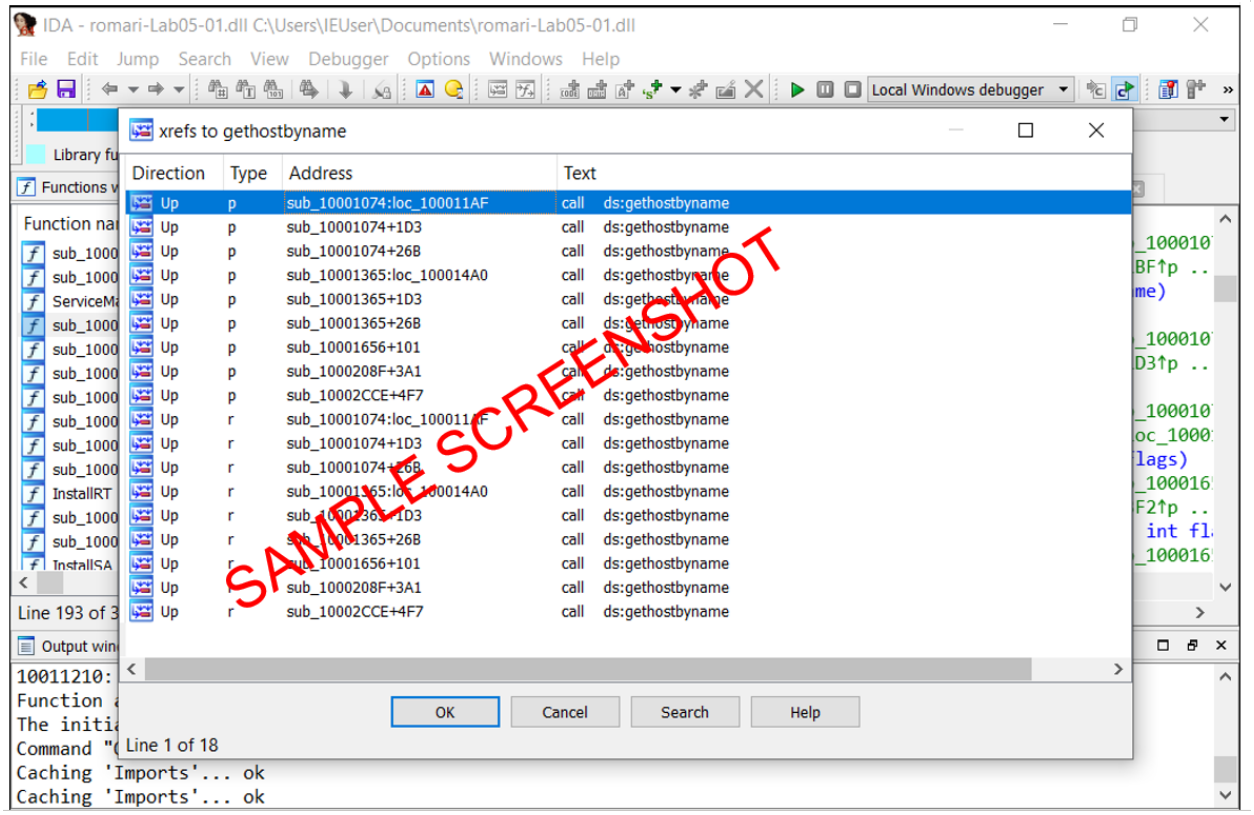
Questions 2. Use the Imports window to browse to `gethostbyname`. Where is the import located?

To answer questions 2 through 4, we begin by viewing the imports of this DLL, by selecting **View** ▸ **Open Subviews** ▸ **Imports**. In this list, locate `gethostbyname`.
The `gethostbyname` import resides at location 0x100163CC in the .idata section of the binary as you can see below. Take a screenshot to replace the one below, and place it under Screenshot#6 in the answer file.
Note that addresses may be different on your machine.



Questions 3. How many separate functions call `gethostbyname`?

To see the number of functions that call `gethostbyname`, first double-click `gethostbyname` to see it in the disassembly, then check its cross-references by pressing CTRL-X with the cursor on `gethostbyname`, which brings up the window shown below:



The text “Line 1 of 18” at the bottom of the window tells us that there are nine cross-references for `gethostbyname`. Some versions of IDA Pro double-count cross-references: p is a reference because it is being called, and r is a reference because it is a “read” reference (since it is call dword ptr [...] for an import, the CPU must read the import and then call into it).

Examining the cross-reference list closely, how many separate functions call the `gethostbyname` ?
Answer question 3 and Click Cancel.

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

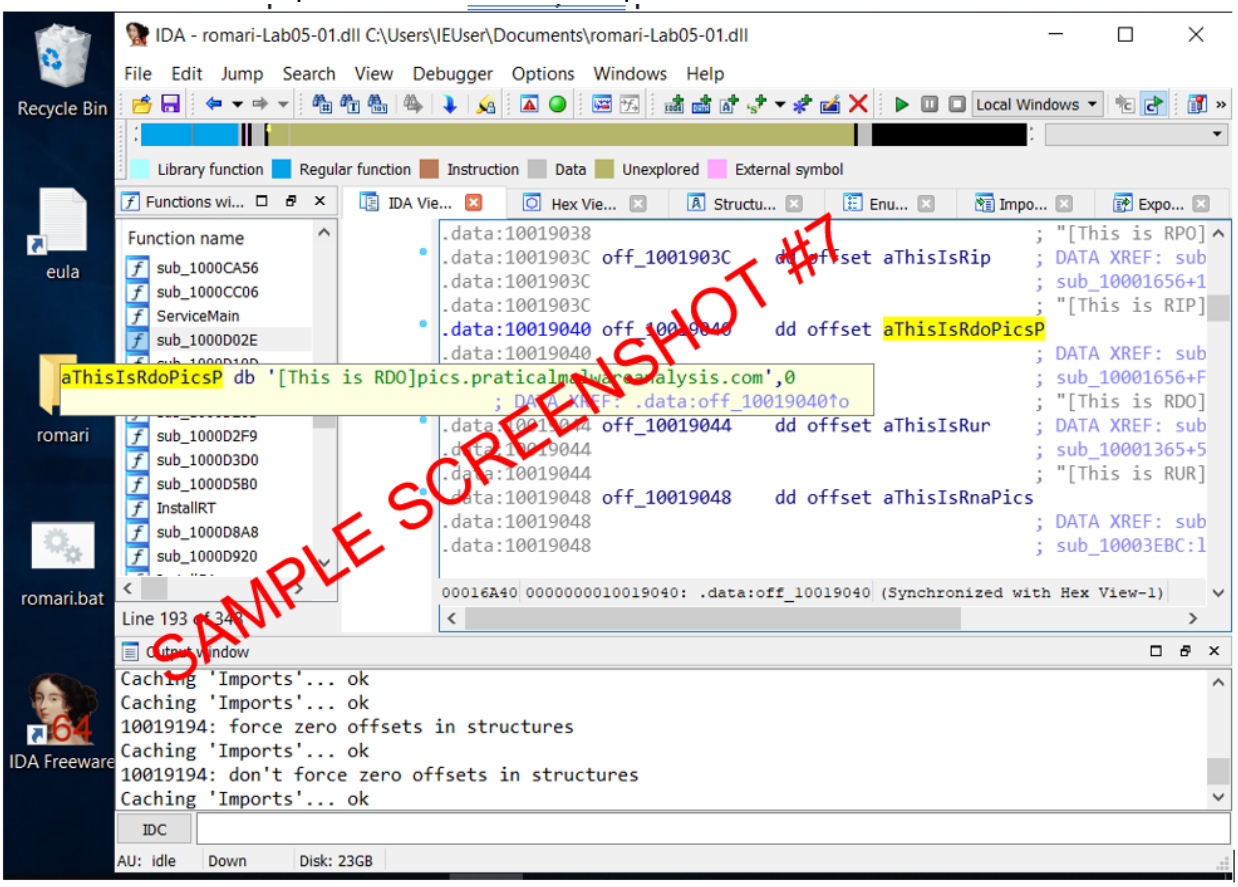
To help answer question 4, we press G on the keyboard to quickly navigate to `0x10001757`. Once at this location, we see the following code, which calls `gethostbyname`.

```
1000174E        mov     eax, off_10019040
10001753        add     eax, 0Dh ❶
10001756        push    eax
10001757        call    ds:gethostbyname
```

The `gethostbyname` method takes a single parameter—typically, a string containing a domain name. Therefore, we need to work backward and figure out what is in EAX when `gethostbyname` is called.

It appears that off_10019040 is moved into EAX. If we double-click that offset, we see the string [This is RDO]pics.practicalmalwareanalysis.com at that location.

Take a screenshot to replace the one below and place it under Screenshot#7 in the answer file.



As you can see at ❶, the pointer into the string is advanced by 0xD bytes, which gets a pointer to the string `pics.practicalmalwareanalysis.com` in EAX for the call to gethostbyname. The figure below shows the string in memory, and how adding 0xD to EAX advances the pointer to the location of the URL in memory. The call will perform a DNS request

to get an IP address for the domain.

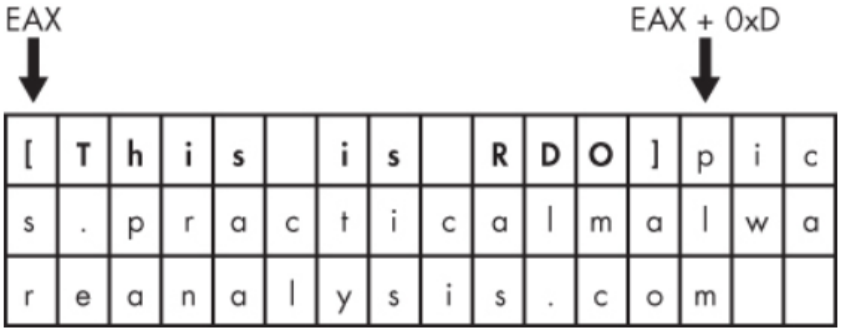
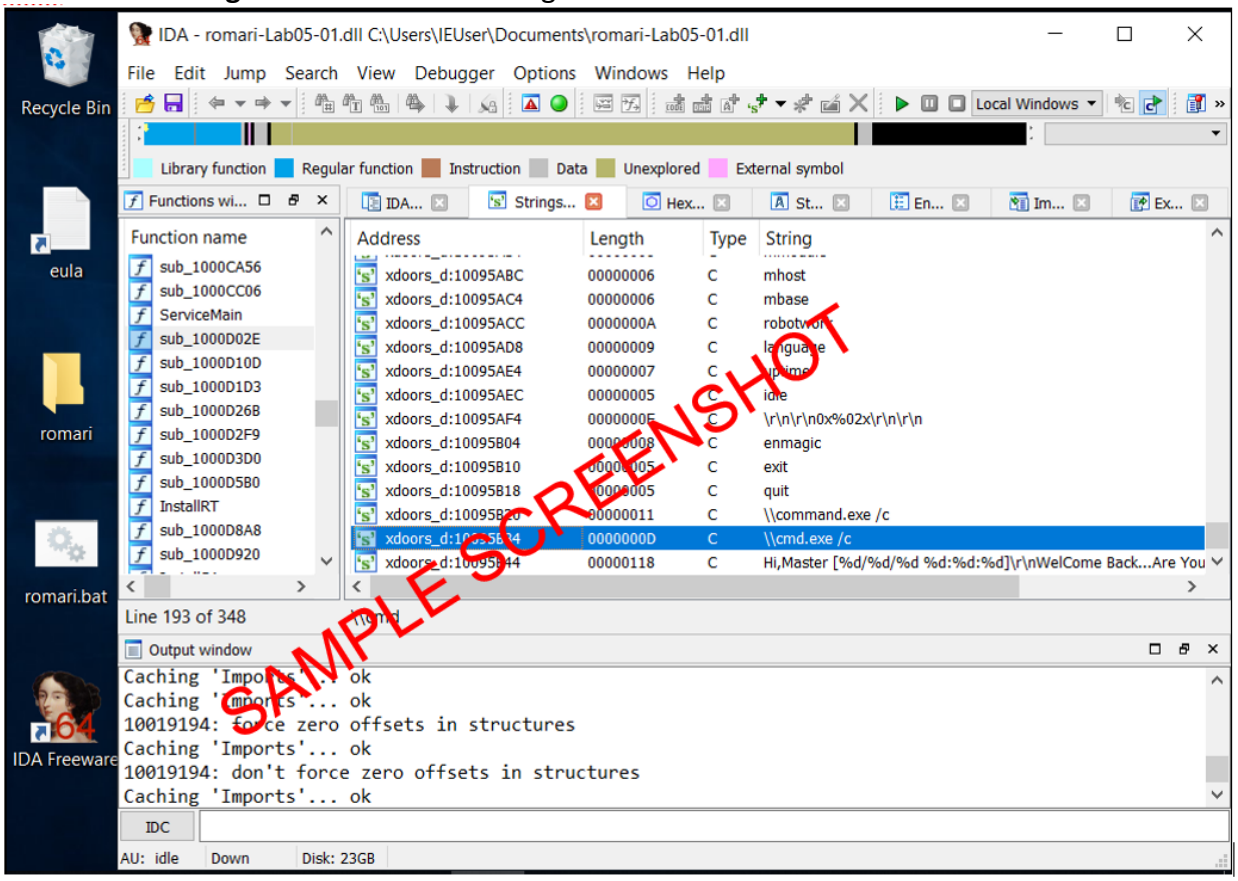


Figure C-13. Adjustment of the string pointer to access the URL

Questions 5. Use the Strings window to locate the string \cmd.exe /c in the disassembly. Where is it located?

To answer question 5, we begin by viewing the strings for this DLL by selecting **View ▸ Open Subviews ▸ Strings**. The address the string cmd.exe resides at is shown in the window below.

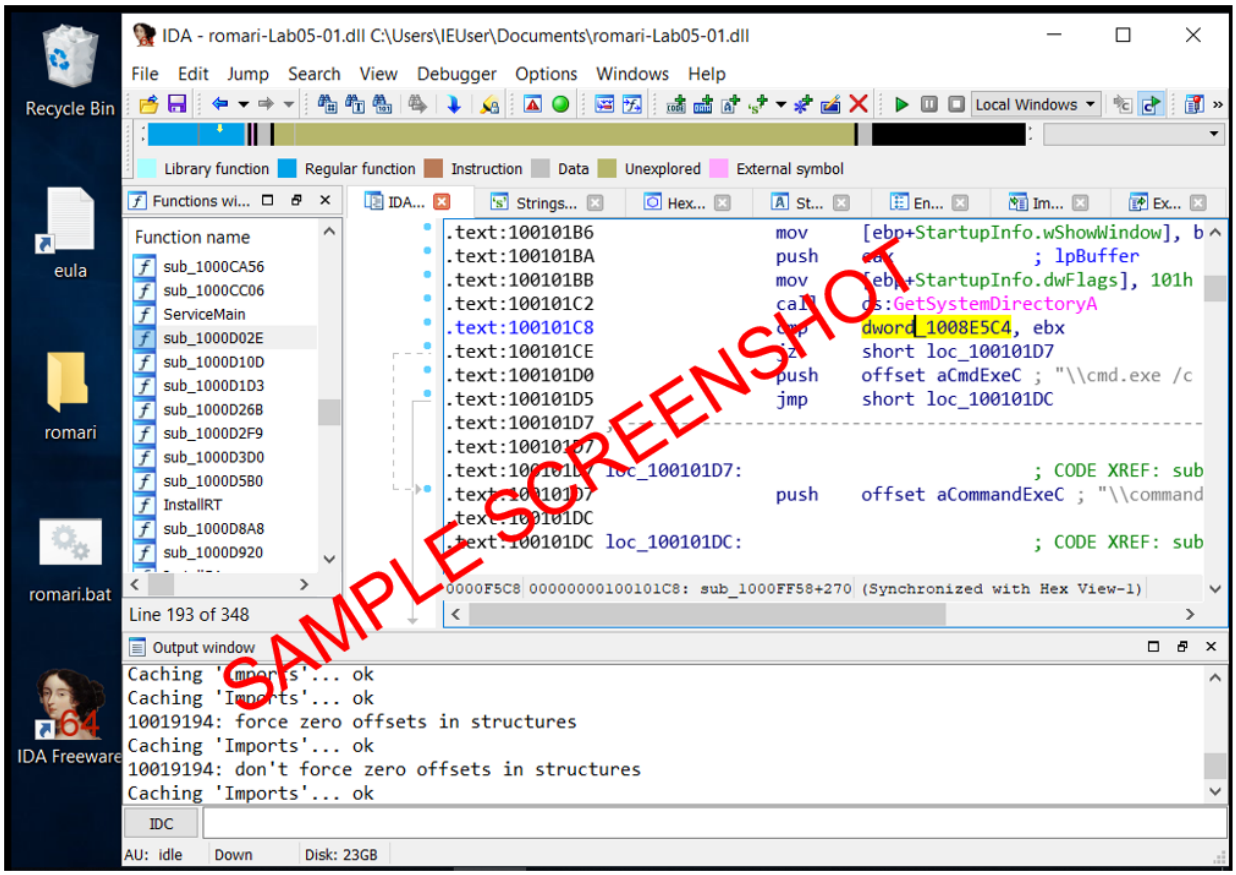


The remaining questions and screenshots are for your own exploration. You are expected to follow and complete the steps, answer the questions and take screenshots in your own

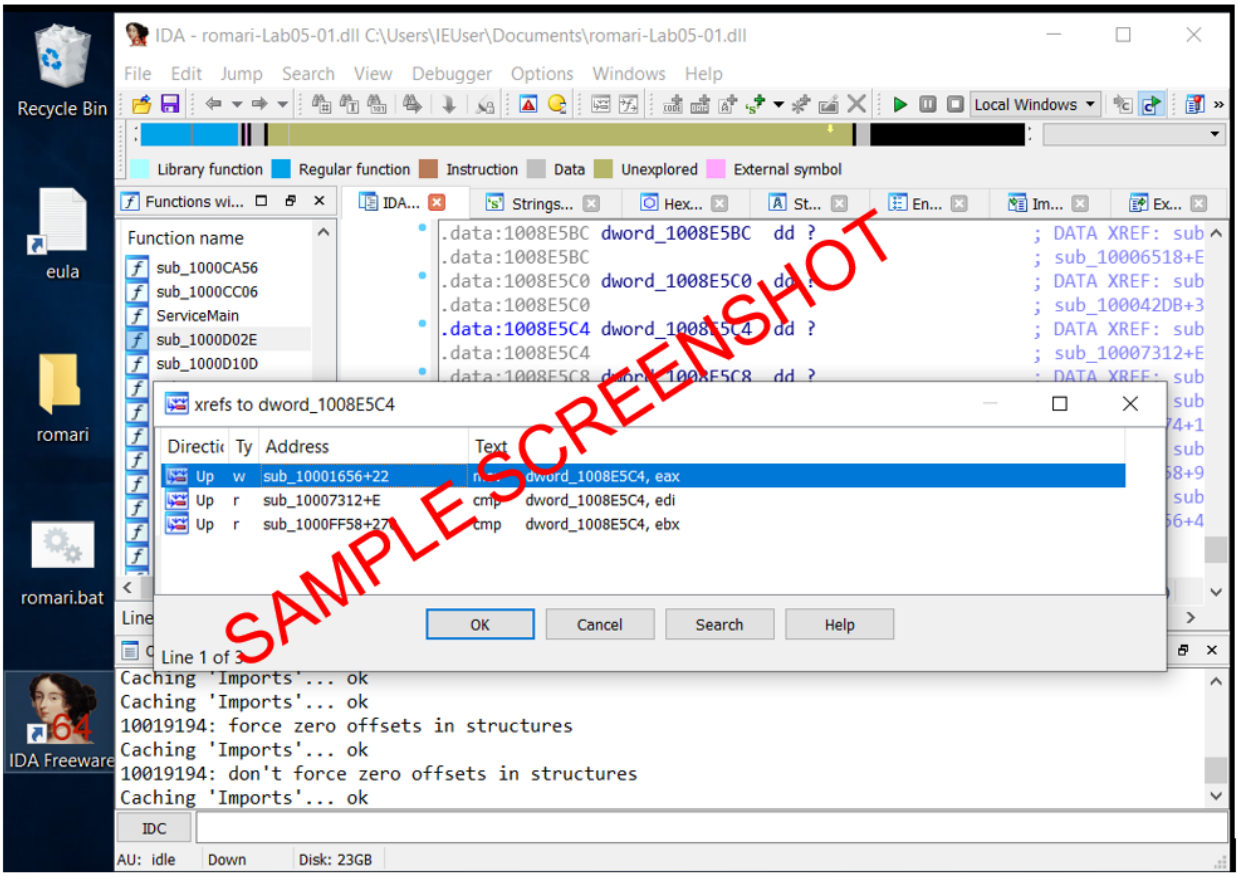
notebook. Do not submit answers or screenshots for the remaining instructions starting with Question 6.

Question 6. What is the value of eax as returned by sub_10003695 ?

To answer question 6, in the same area, browse to 0x100101C8, it looks like dword_1008E5C4 is a global variable that helps decide which path to take. We notice that the dword_1008E5C4 is a global variable that we can double-click (at 0x100101C8) to show its location in memory at 0x1008E5C4, within the .data section of the DLL.



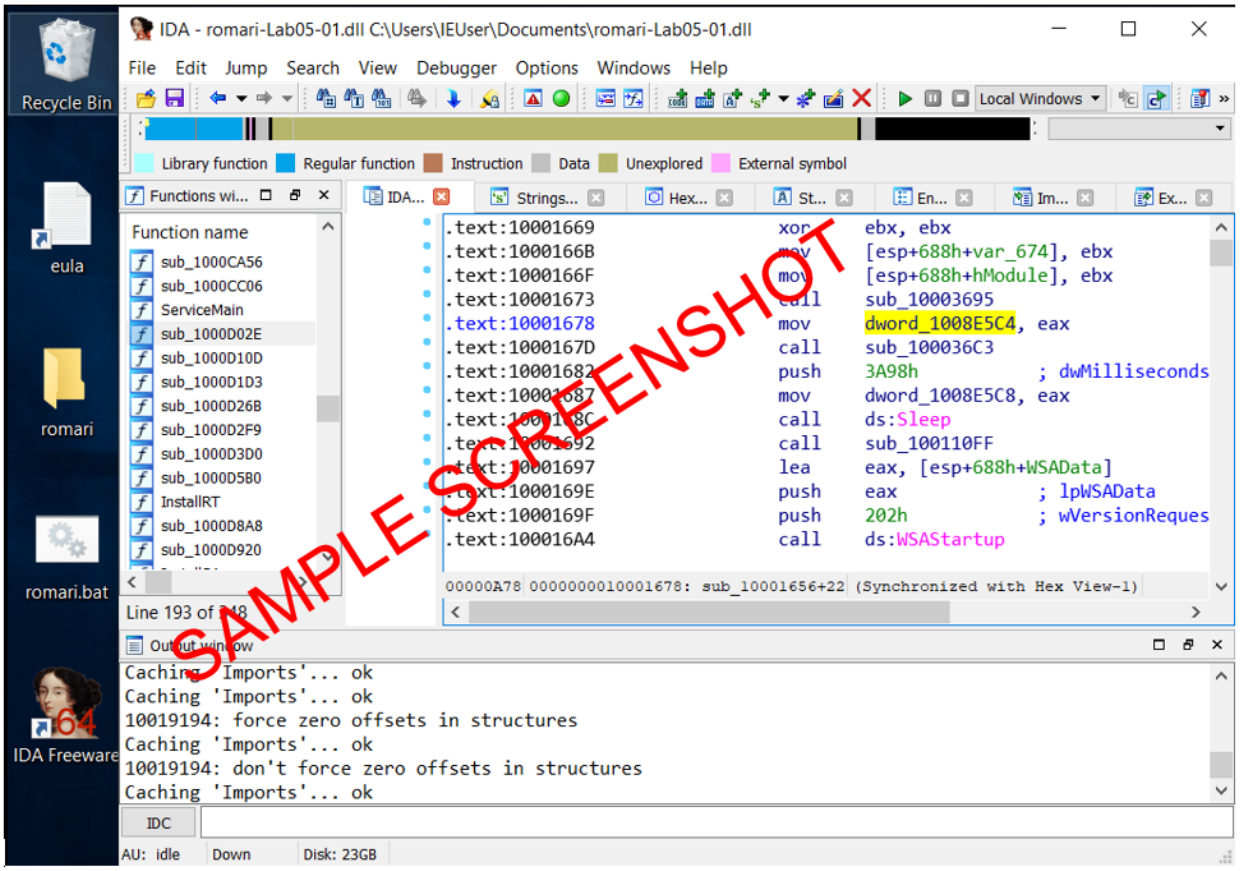
Double-clicking the address, and checking the cross-references by pressing CTRL-X shows that it is referenced three times:



But only one reference modifies dword_1008E5C4. Double click the entery with the w Type. You will notice the following listing which shows how dword_1008E5C4 is modified.

```
10001673          call     sub_10003695

10001678          mov      dword_1008E5C4, eax
```

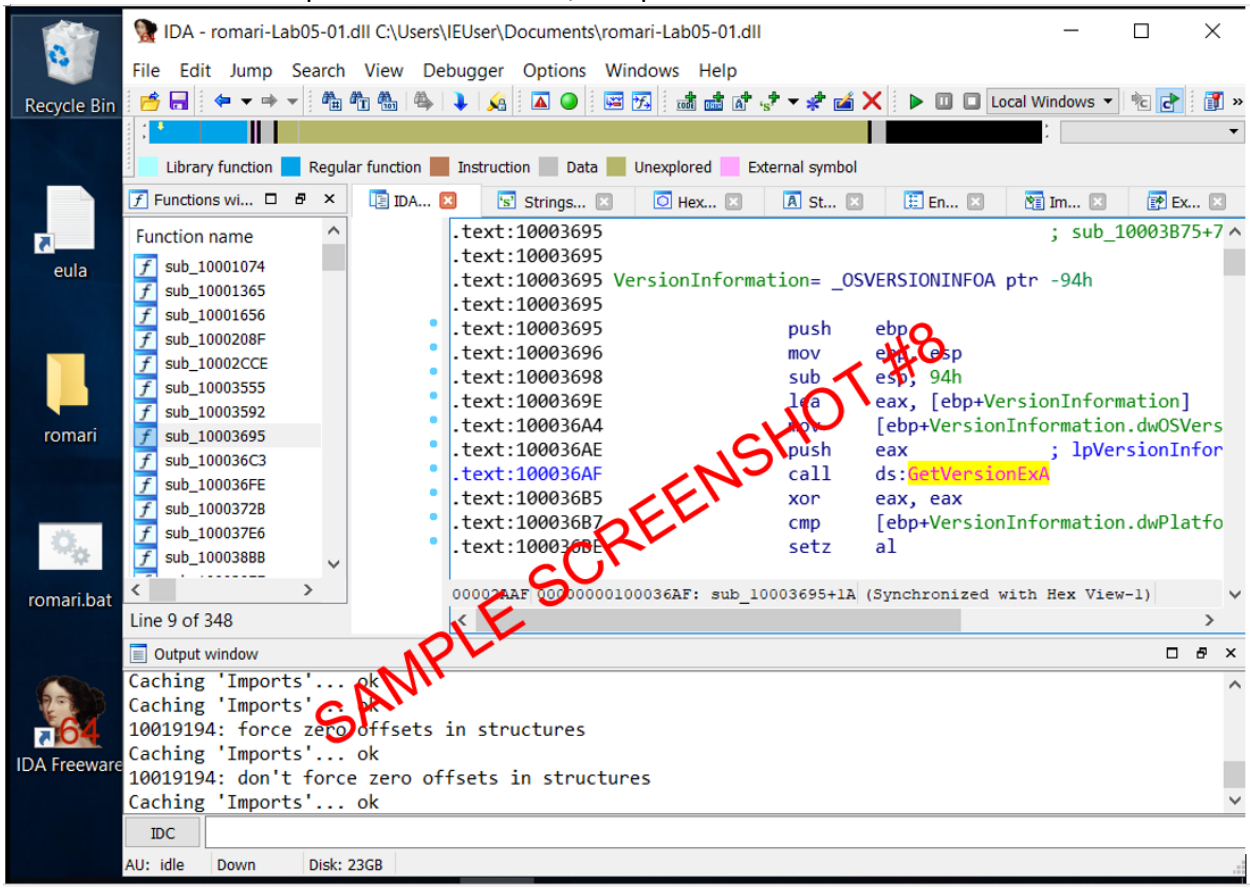


We see that EAX is moved into dword_1008E5C4, and that EAX is the return value from the function call made in the previous instruction. Therefore, we need to determine what that function returns. To do so, we examine sub_10003695 by double-clicking it and looking at the disassembly.
The sub_10003695 function contains a call to GetVersionEx, which obtains information about the current version of the OS, as shown in the following listing.

```
100036AF        call    ds:GetVersionExA
100036B5        xor     eax, eax
100036B7        cmp     [ebp+VersionInformation.dwPlatformId],
2
100036BE        setz    al
```

The dwPlatformId is compared to the number 2 in order to determine how to set the AL register. AL will be set if the PlatformId is VER_PLATFORM_WIN32_NT. This is just a simple check to make sure that the OS is Windows 2000 or higher, and we can conclude that the global variable will typically be set to 1.

Take a screenshot to replace the one below, and place it under Screenshot#8 in the answer file.



End of Part 2