**Lab 9: Working with IDA**

**Solutions**

**Overview**

Today we will practice advanced static analysis using IDA. The goal is to gain familiarity with IDA and advanced static analysis, not necessarily to perform a full analysis using IDA.

**Part 1: Crack the file(s) using the graph view!**

1. Download the “IDA\_practice\_crackme.7z” archive from Canvas, move it to your Windows VM, and unzip it using “cse434” as the password.

2. For each of the files:

* Run the program in the command-line. Explain what the program does.
* Use IDA to analyze the executable and crack the password. (Note: you will need to download an older version of IDA to run on our Windows 7 VM. The older version (version 5.0) is available as a link from this post: <https://www.scummvm.org/news/20180331/> .)
* Write abstract pseudocode to describe the program flow you identified.

--------------------------------------------------------------------------------------------

\* crackme-1.exe

1. It looks like a program with a password crack challenge.

2. The password is ‘topsecret’

3. pseudocode

Main() {

If there is no input parameter, then print ‘Usage: crackme-123-1 password’

Check a user input param with the ‘topsecret’

If succ, then print ‘You found the password! Congratulations!’

Else, then print ‘Fail!’

}

\* crackme-2.exe

1. It looks like a program with a password crack challenge.

2. The password is ‘alligator’

3. pseudocode

Main() {

If there is no input parameter, then print ‘Usage: crackme-123-2 password’

Check a user input param with the ‘alligator’

If succ, then print ‘You found the password! Congratulations!’

Else, then print ‘Fail!’

}

\* crackme-3.exe

1. It looks like a program with a password crack challenge.

2. You need to give two passwords in order. ‘suffering’ and ‘succotash’. You need to give them in order.

3. pseudocode

Main() {

If there are not 2 input parameters, then print ‘Usage: crackme-123-3 password1 password2’

Check user’s first input param with the ‘suffering’

If first input param doesn’t match, then print ’ Fail! First word was wrong!’

If second input param doesn’t match, then print ‘Fail! Second word was wrong!’

If two parameters are correct, then print ‘Congratulations! You found the passwords!’

}

\* crackme-4.exe

1. It looks like a program with a password crack challenge.

2. The password is ‘dromedary’. But before giving the password, you need to change the file name from ‘crackme-4.exe’ to ‘game3.exe’

3. pseudocode

Main() {

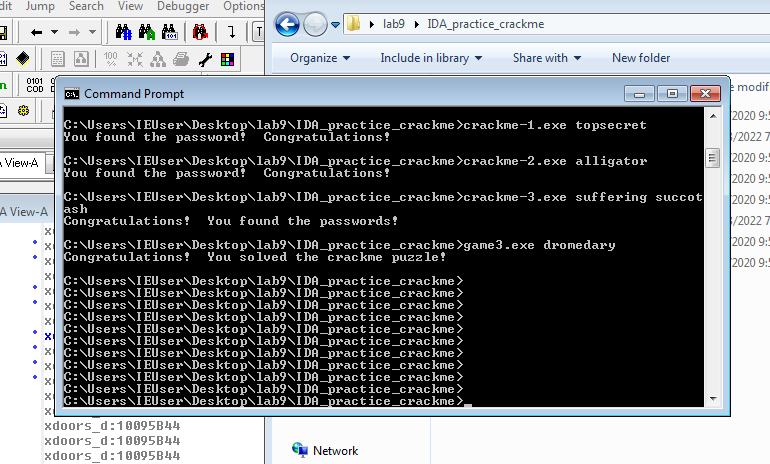
If there is no input parameters, then print ‘Usage: game3.exe password’

Check execution file name and user’s first input param.

If execution file name is ‘game3.exe’ and the first parameter is ‘dromedary’, then print ’ Congratulations! You solved the crackme puzzle!’

Else print ‘Fail!’

}



--------------------------------------------------------------------------------------------

**Part 2: More IDA functionality**

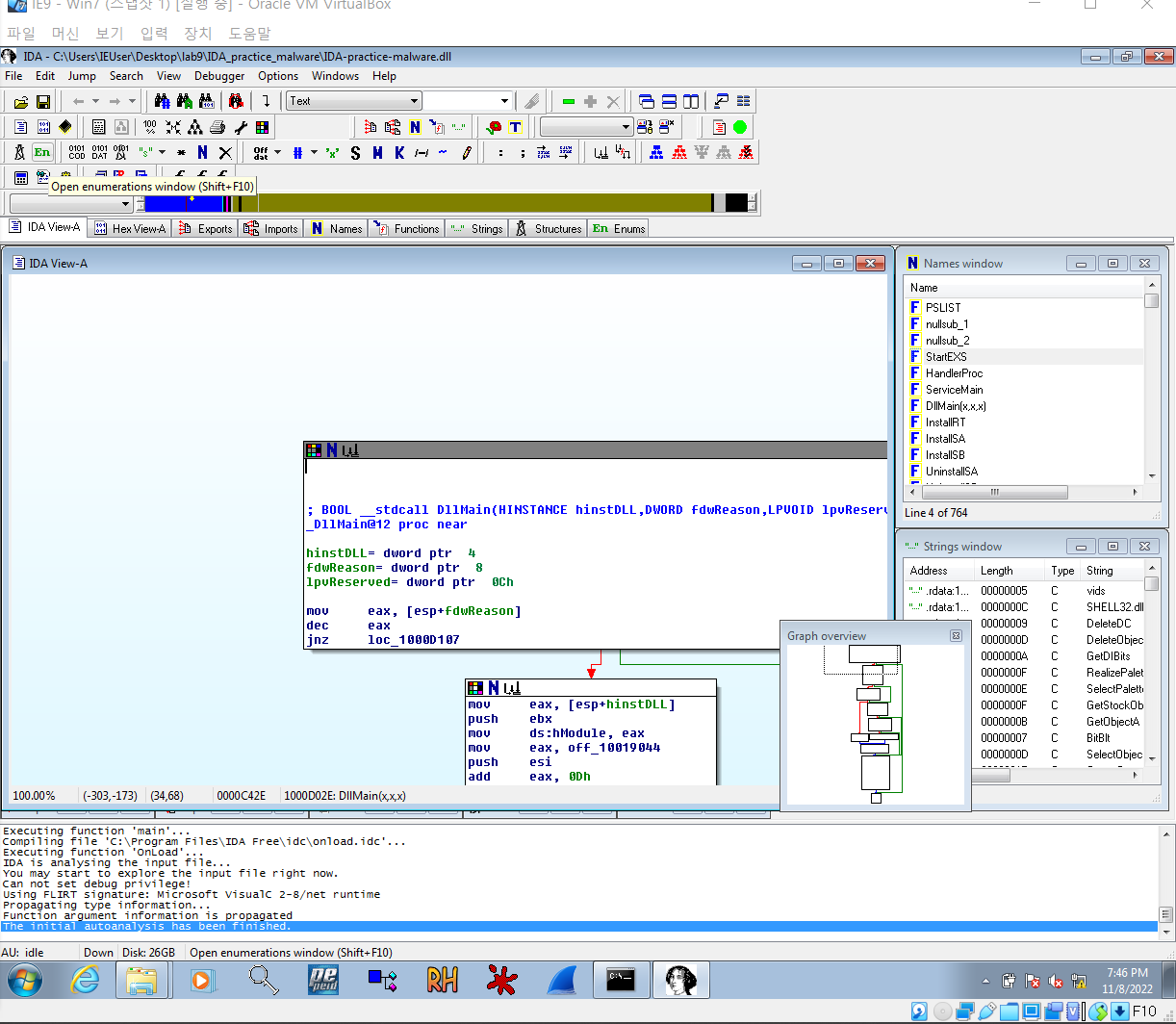
Download the “IDA\_practice\_malware.7z” archive from Canvas, move it to your Windows VM, and unzip it using “infected” as the password. This part of the lab is based on chapter 5 of “Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software.”

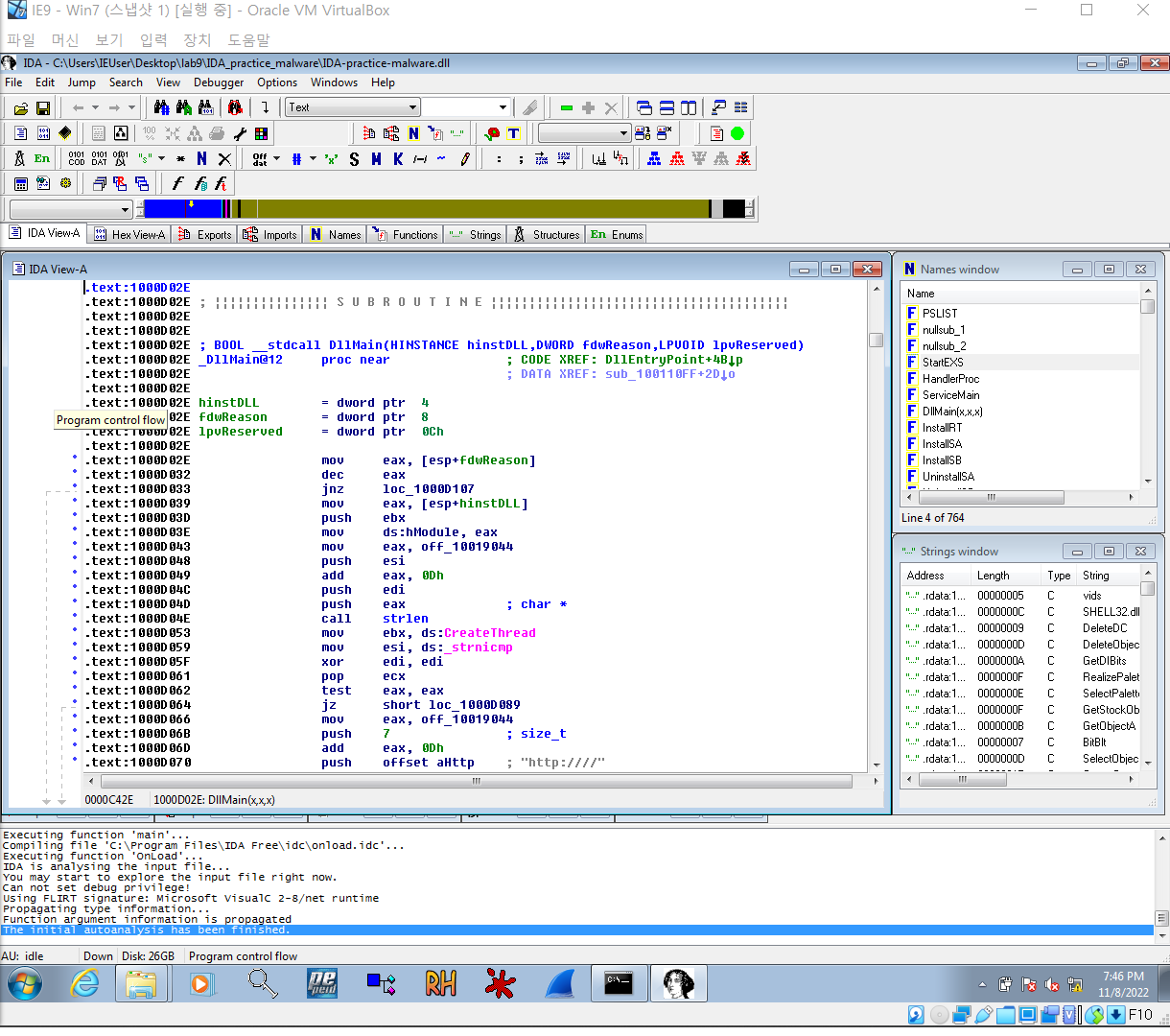
Launch IDA and open ‘IDA\_practice\_malware.dll’. Make sure you’re viewing the "IDA View-A" window. Press the spacebar to see the code.

Switch back to the "graph mode", and repeat the same steps to display this additional information in “graph mode”.

Take a screenshot of what you see:

--------------------------------------------------------------------------------------------





--------------------------------------------------------------------------------------------

"Gethostbyname" is a [Windows API function](https://msdn.microsoft.com/en-us/library/windows/desktop/ms738524(v=vs.85).aspx) that can perform a DNS lookup.

Switch to the “Imports” tab. Click the Name header to sort by name and find "gethostbyname". (Note that capital letters and lowercase letters sort into separate groups.)

Double-click gethostbyname.

The code for the function opens in Text mode. Click gethostbyname. Yellow highlights appear on both occurrences of that name. Can you determine how many times this program is being called?

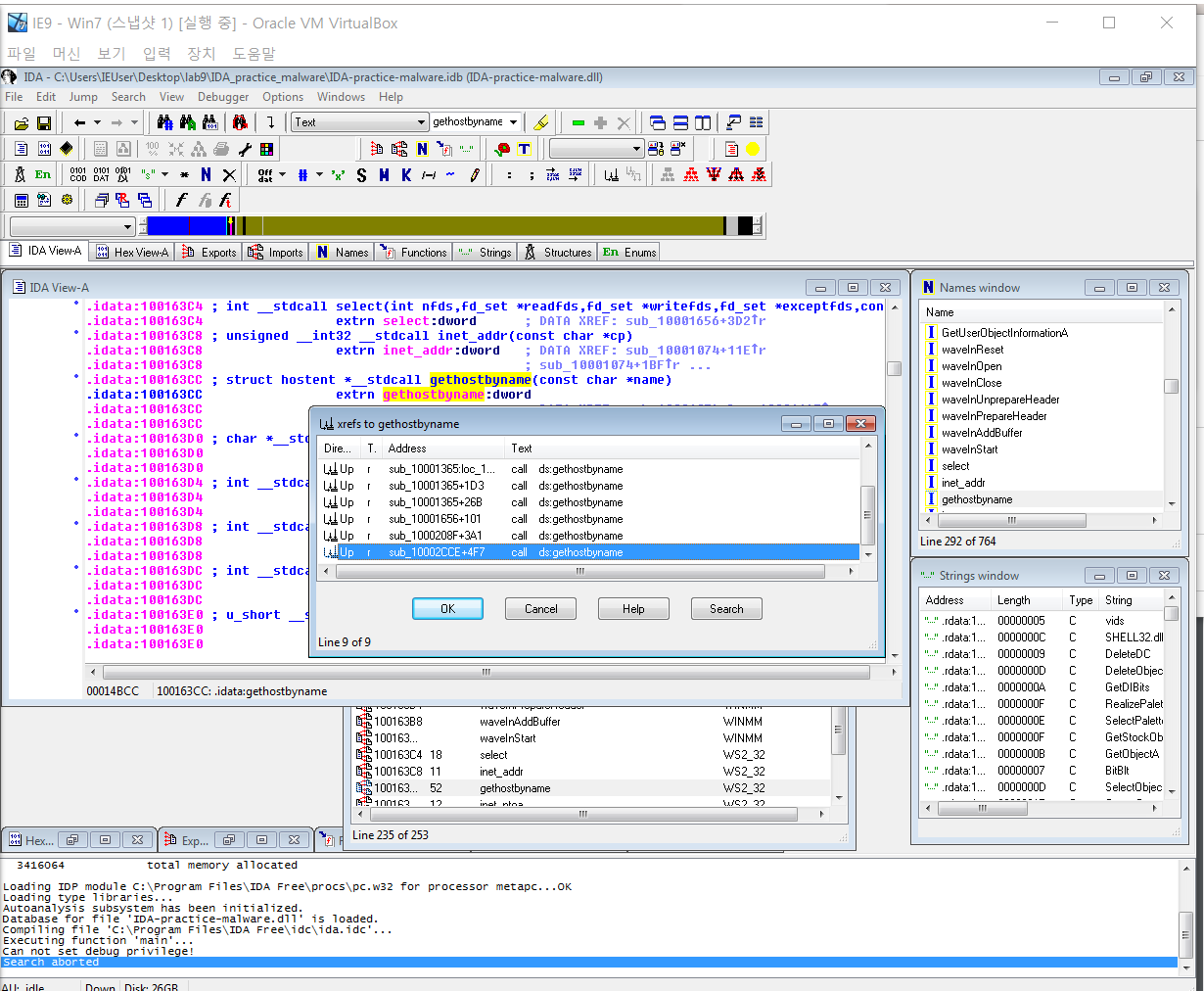
--------------------------------------------------------------------------------------------

The ‘gethostbyname’ is called 9 times.

--------------------------------------------------------------------------------------------

How did you find your answer? If you didn’t use xrefs (by pressing ‘x’), try it now. Take a screenshot of the window.

--------------------------------------------------------------------------------------------



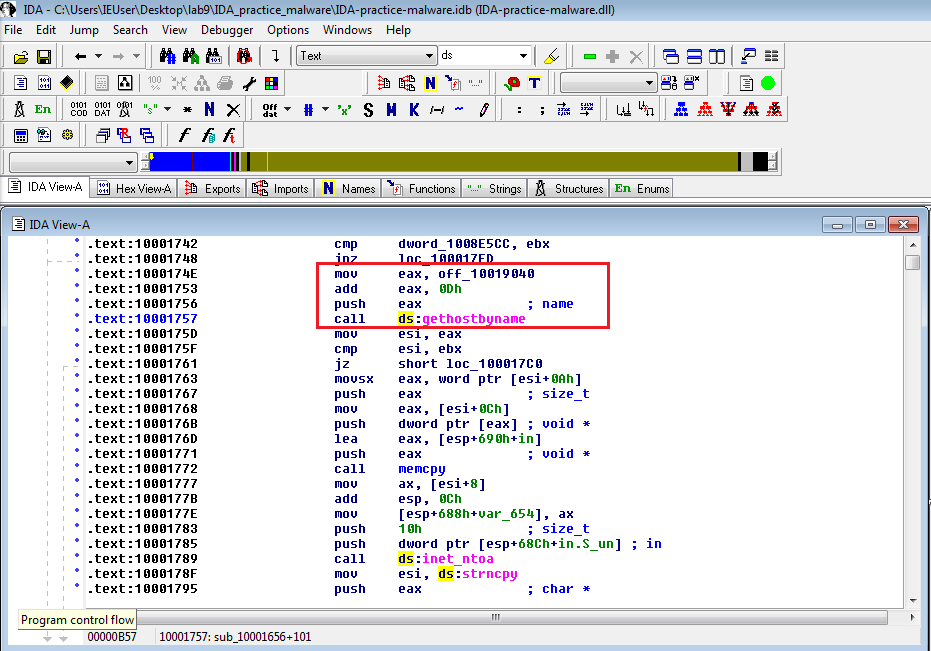
--------------------------------------------------------------------------------------------

Double-click the line that has 1001656+101.

Now you are looking at a function. You can see that it loads an address named off\_10019040 into register eax, adds 13 to it, pushes that address onto the stack, and calls gethostbyname.

Take a screenshot of the function and make sure that the described operation is shown in your screenshot.

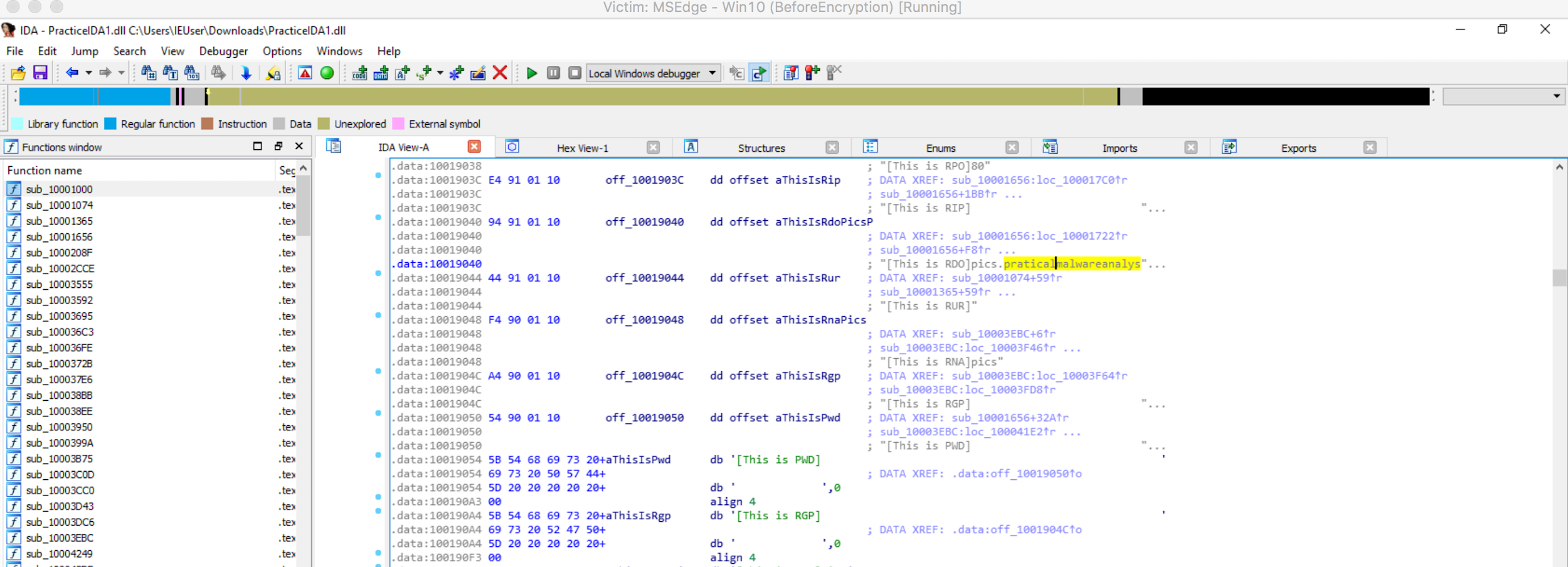
--------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------

Double-click off\_10019040.

The Text view shows that this location contains a pointer to a string containing "praticalmalwareanalys", as shown below.



Choose “praticalmalwareanalys” string, and click the "Hex View-A" tab.

The four bytes starting at 10019040 contain a 32-bit address in little-endian order. That address is 10019194. Find this address in the presented view, and report the domain name you found:

--------------------------------------------------------------------------------------------

I found ‘pics.practicalmalwareanalysis.com’ at address ‘10019194’

--------------------------------------------------------------------------------------------

This is the domain that will be resolved by calling gethostbyname. Do you understand why?

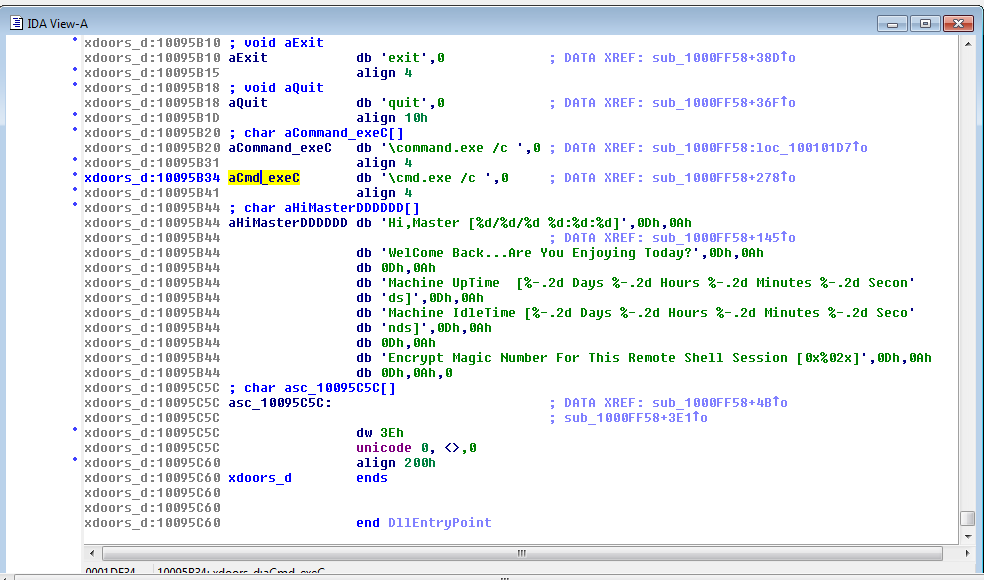
Open the Strings tab (View > Open subviews > Strings). Click the gray String column header to sort the data.

Scroll down about 3/4 of the way, and find the String "\\cmd.exe /c",

Double-click "\\cmd.exe /c", and now move to “IDA View-A" tab.

Take a screenshot of what you see.

--------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------

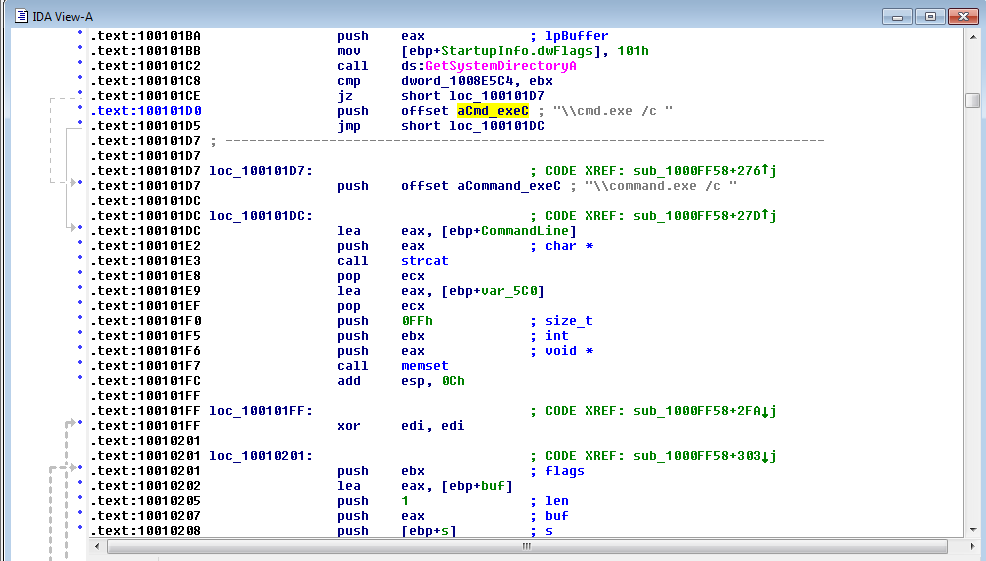
The string appears in text mode. Click in the word cmd so it's highlighted and press x. A "xrefs to aCmd\_exeC" box appears.

In the "xrefs to aCmd\_exeC" box, double-click sub\_1000FF58+278.

You see the code that uses this string. There are two boxes of code, one that starts a string with "cmd.exe -c" and the other that starts it with "command.exe /c".

Take a screenshot of what you see.

--------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------

This looks like a remote shell, executing commands from the botmaster for either a 32-bit or 16-bit system.

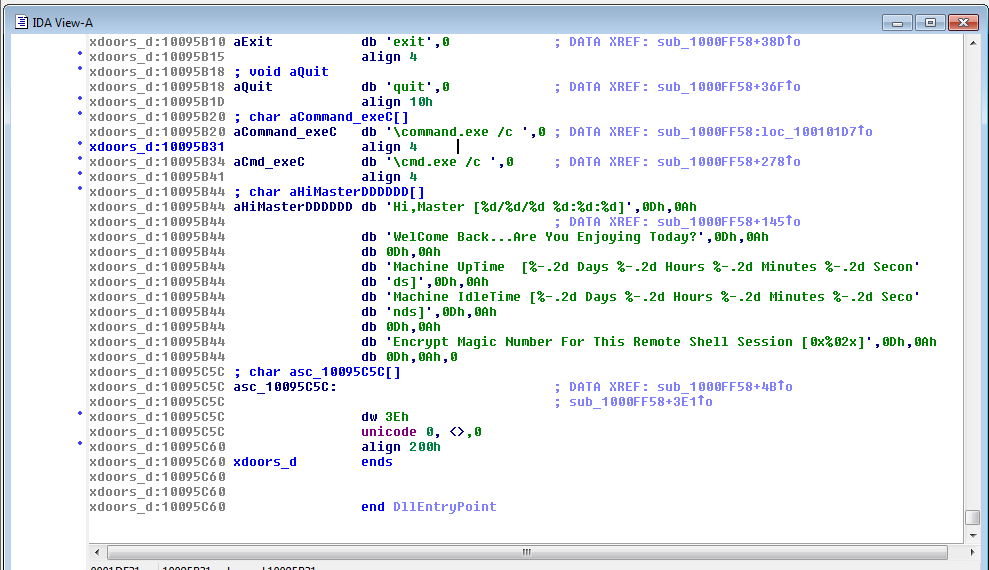
Drag the code boxes down to see the module containing "Hi, Master".

Hover the mouse over aHiMasterDDDDDD to see more of the referenced strings.

This looks like a message the bot sends to the botmaster, further confirming that this is a RAT (Remote Administration Tool / Remote Access Trojan).

Take a screenshot of what you see.

--------------------------------------------------------------------------------------------



--------------------------------------------------------------------------------------------

**Part 3 - Check your understanding**

The .dll you analyzed in Part 2 is Lab05-01.dll provided by the book. Go over the Lab questions on page 107, and try to answer them. The book provides answers, so you can verify you can check your answers as needed. This part is optional, and no submission is required.