

Final Project

Prof. Alberto LaCava

CY -640

MALWARE ANALYSIS & DEFENSE

Nikhil Patel

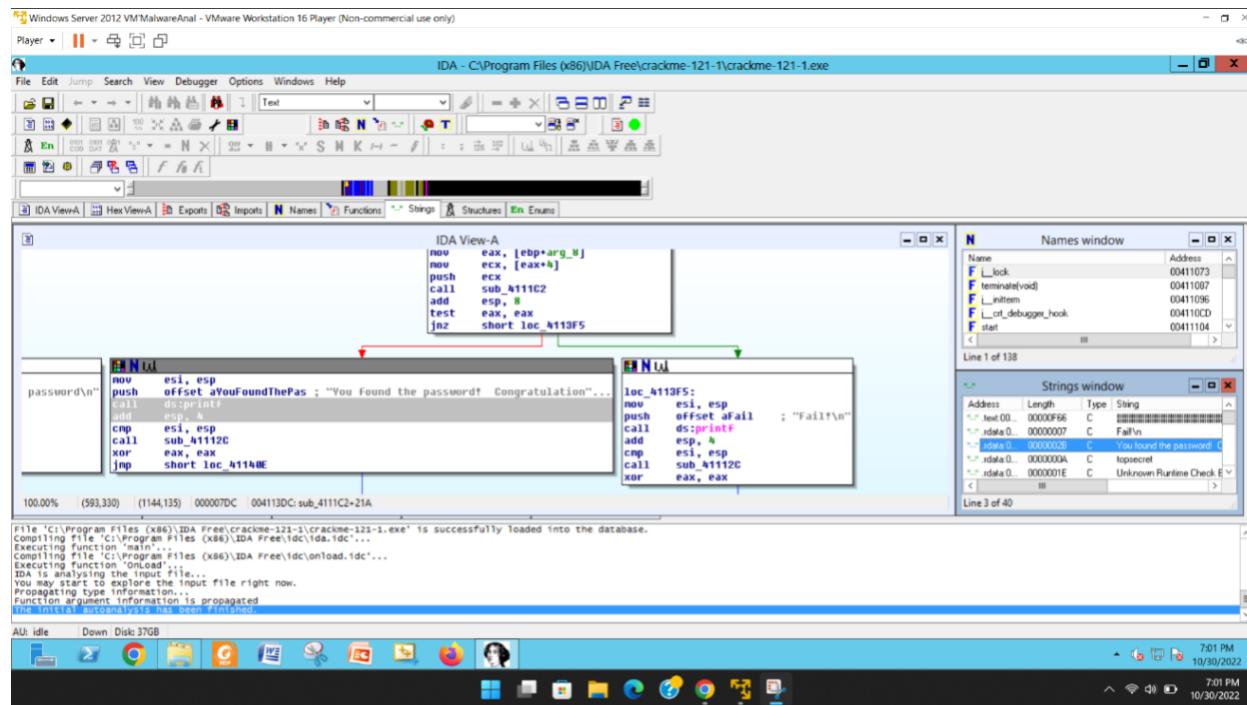
11/22/2022

Project 2: Reverse Engineering with IDA Pro Freeware

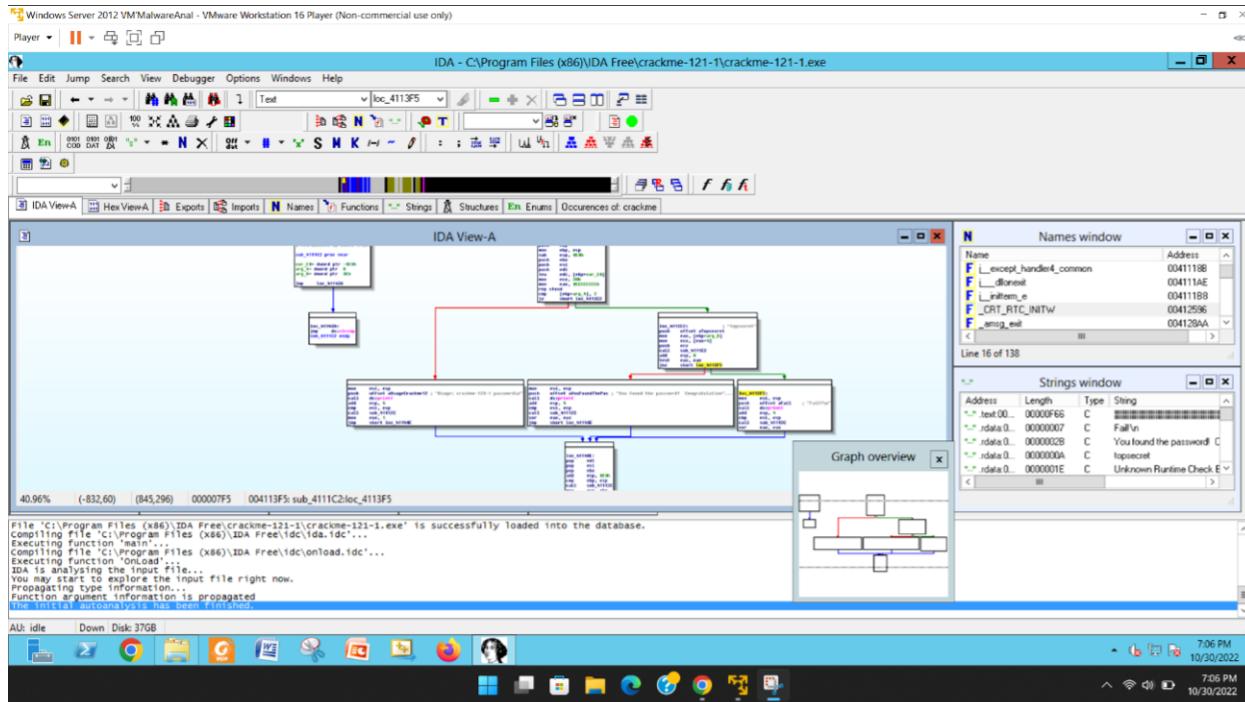
Overview: In this lab, I examined four different types of executable files and attempted to decrypt them using IDAPro.

Procedure:

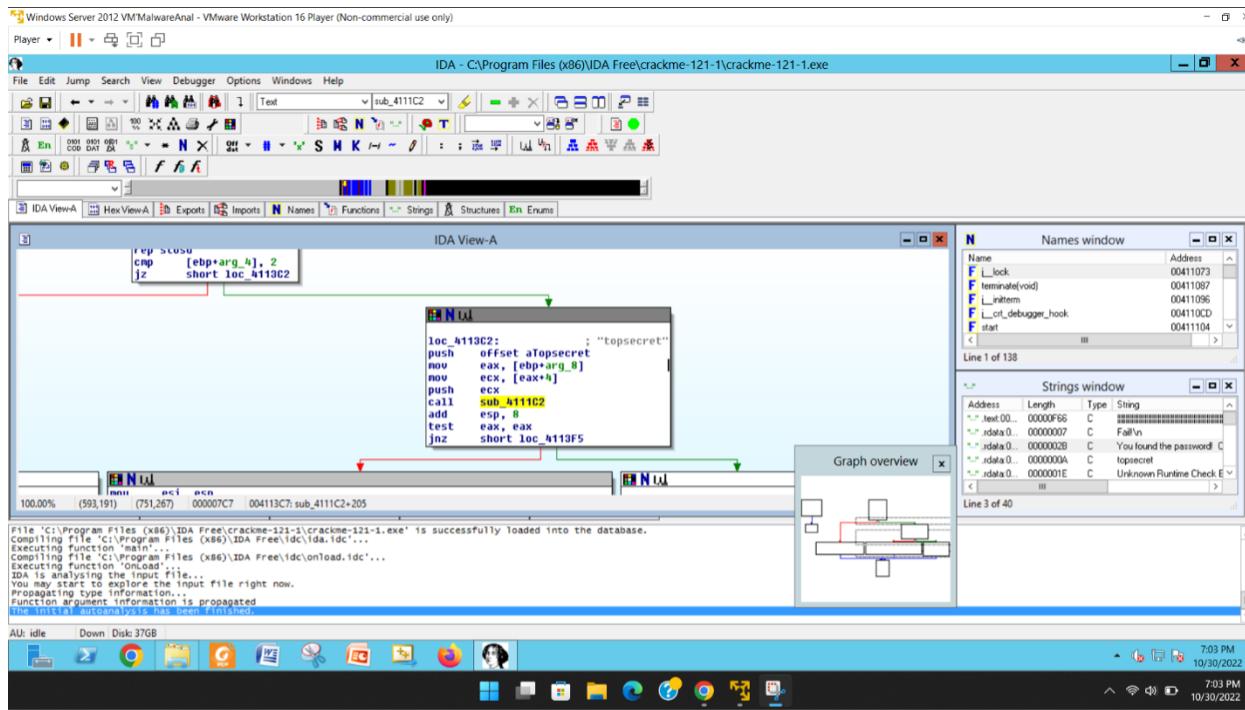
1. Go to Blackboard and look for a file called crackme-121-1.exe. Open the file with IDA Pro after downloading it.
2. Now, after it's opened, look in the string module for the string "You discovered password."



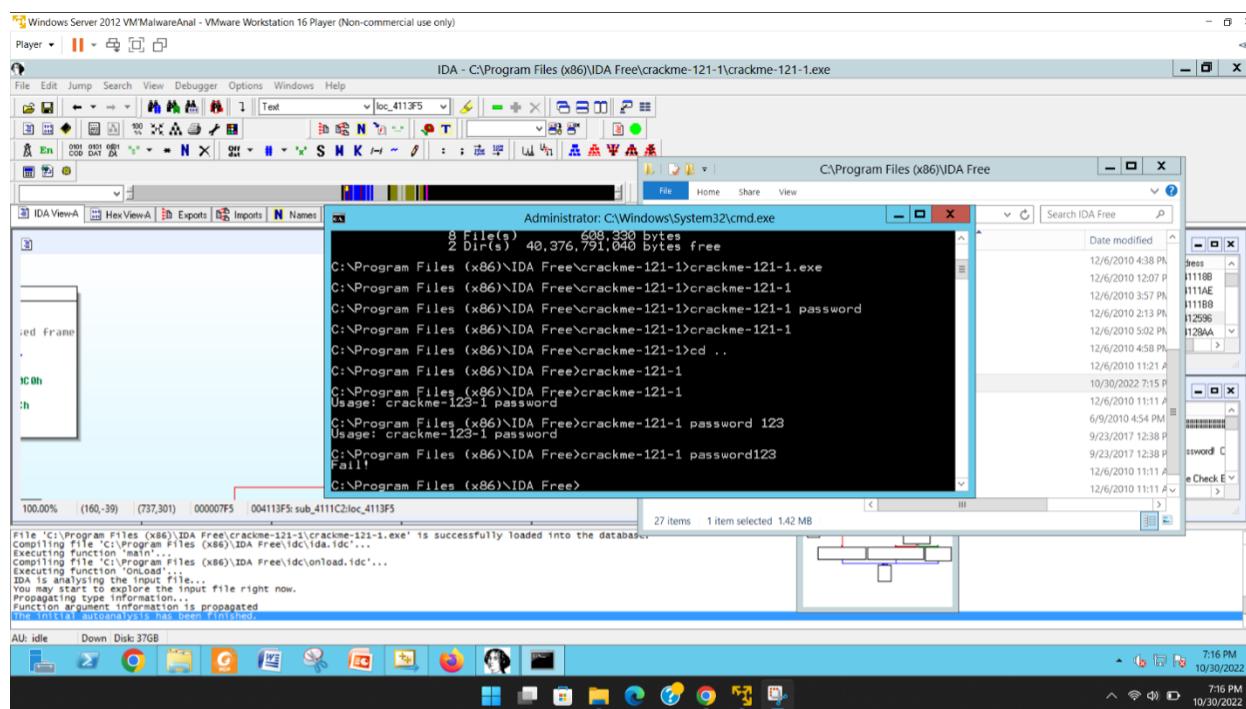
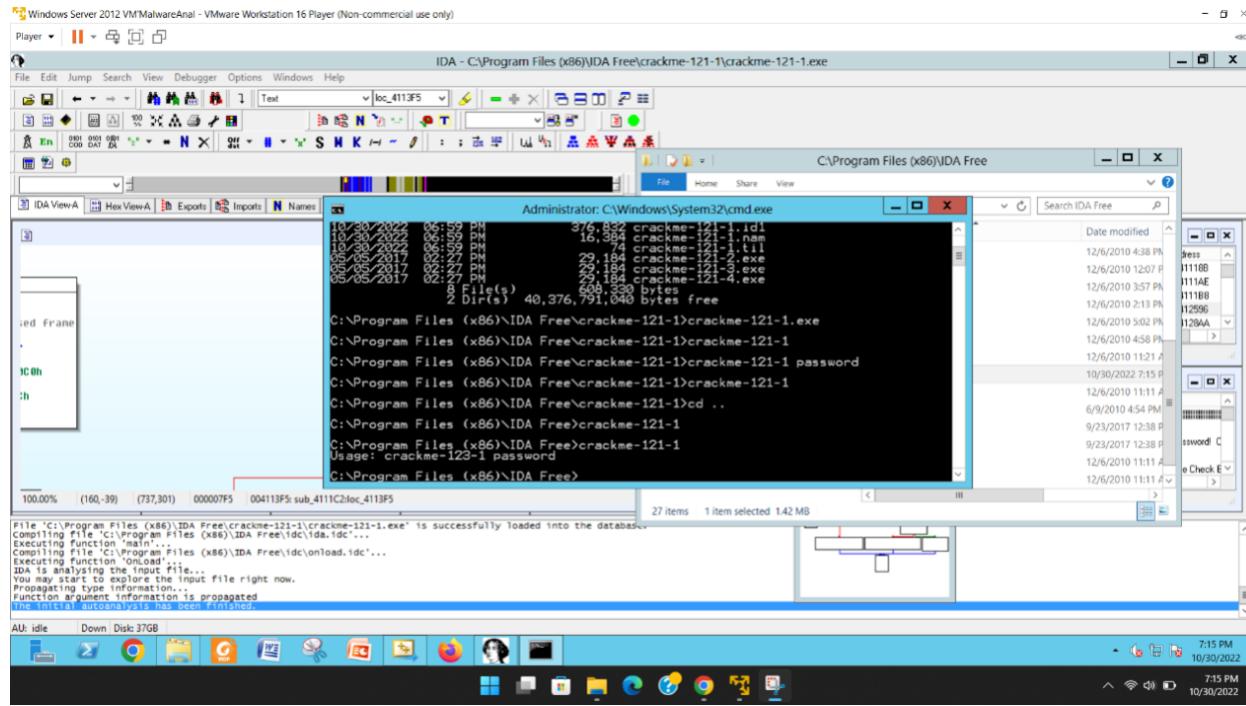
3. Zoom out to examine the whole flow of the exe file.

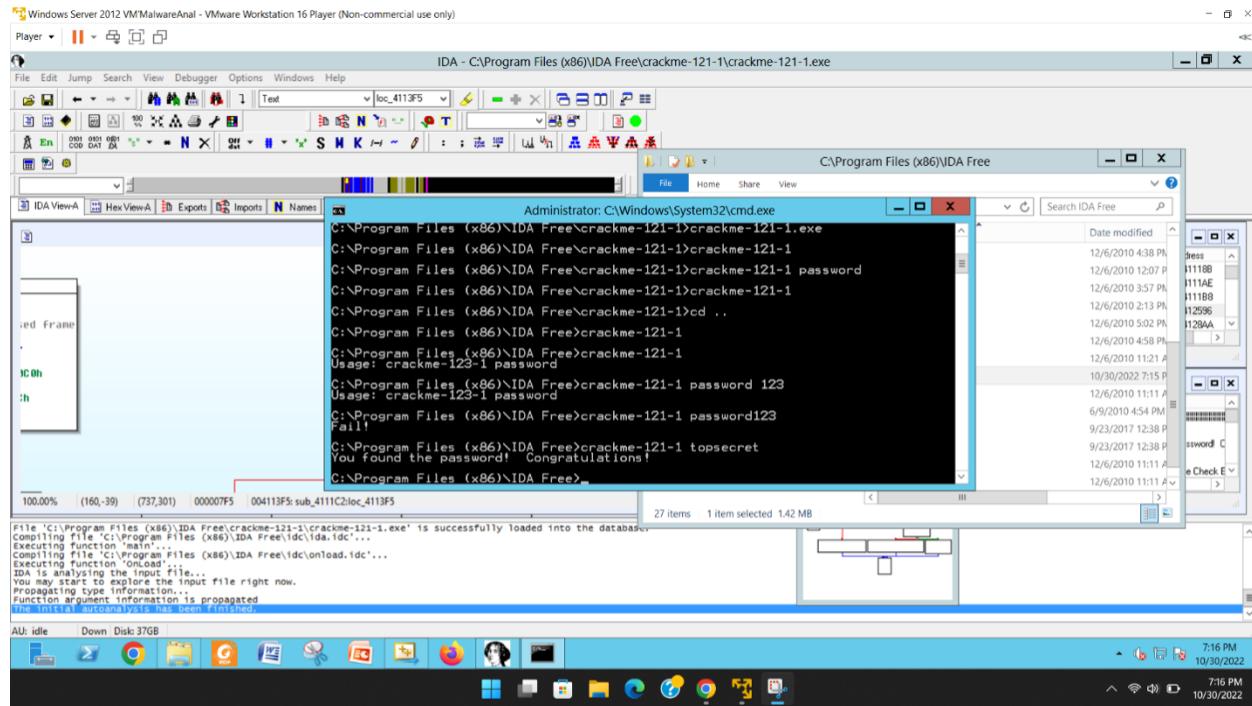


4. Following the flow will lead you to a password (the password is "top-secret").



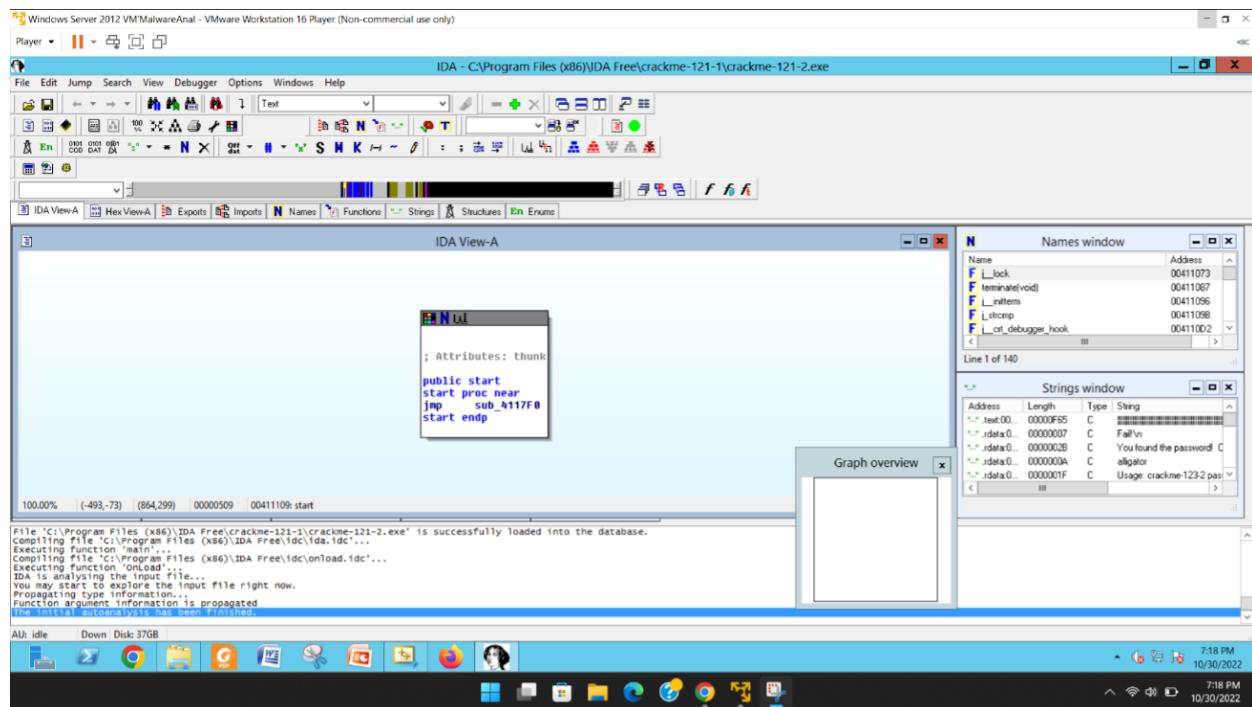
5. Open cmd and go to the crackme directory, then execute the crackme-121-1.exe file, which will display instructions (it will ask you to enter a password). So enter the password you discovered in IDAPro.





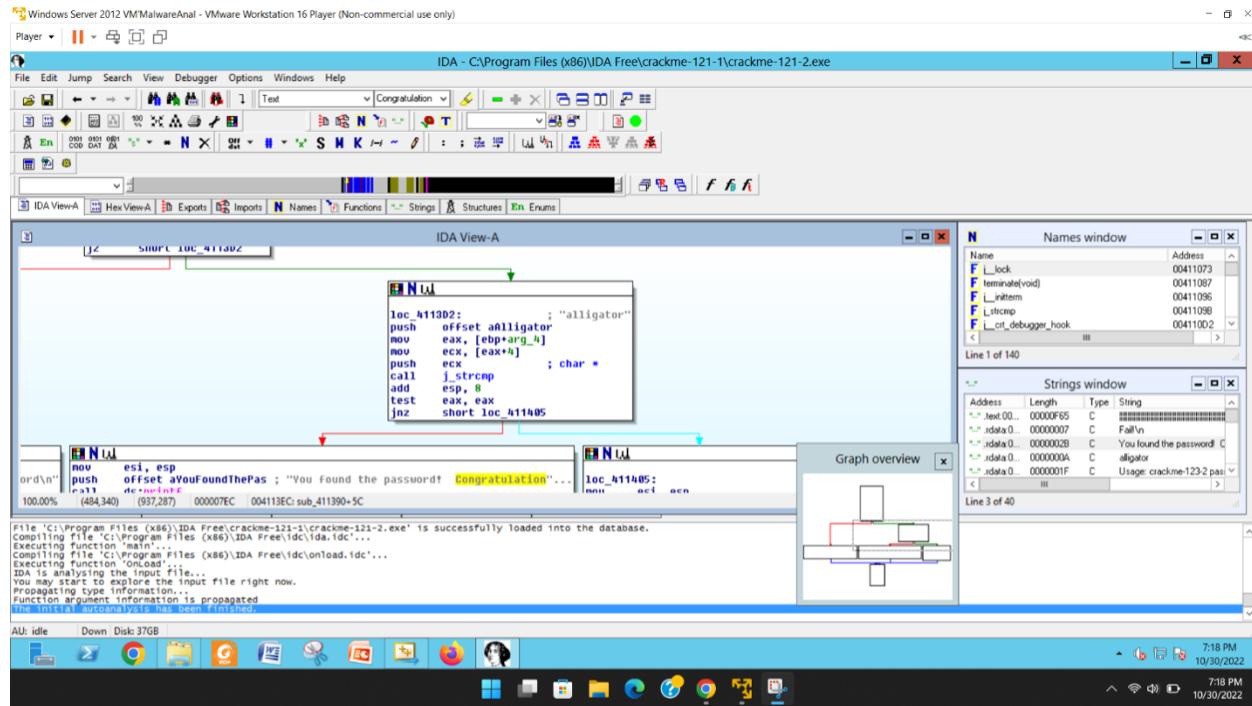
Crackme-121- 2.exe

1. Load crackme-121-2.exe into IDAPro and look in the strings module for the string "discovered password."
2. It will take you to the DATA XREF: sub 411390+5c location, which you should double-click on.

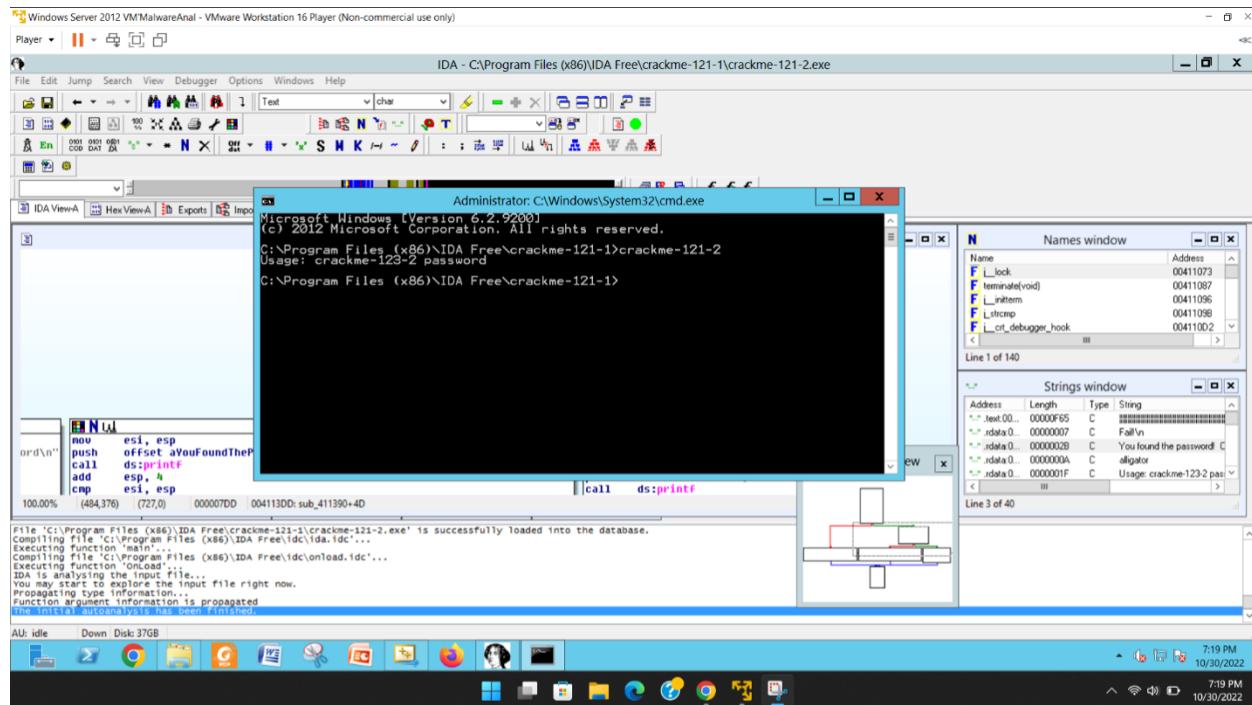


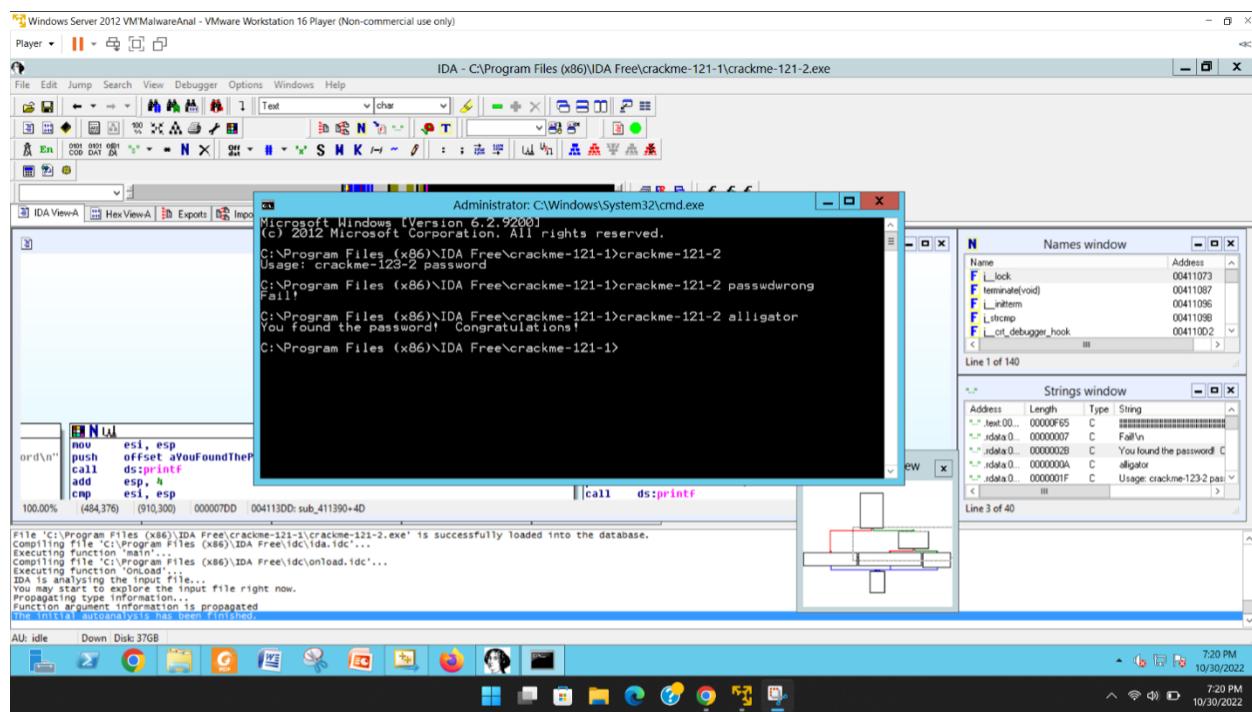
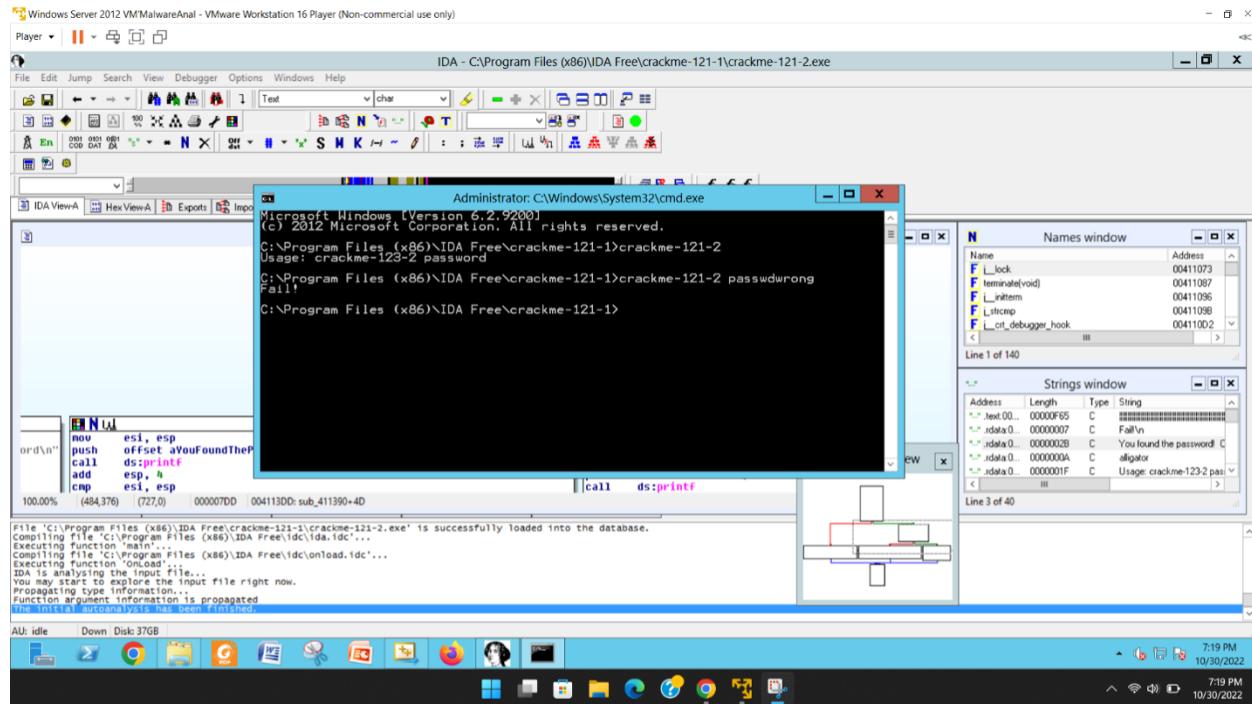
The screenshot shows the IDA Pro debugger interface running on Windows Server 2012. The main window displays assembly code for the file 'crackme-121-1\crackme-121-2.exe'. The assembly code includes instructions like .rdata, db, and DATA XREF. The 'Names window' on the right lists symbols such as '_lock', '_interate(void)', '_intern', '_lstrcpy', and '_ct_debugger_hook'. The 'Strings window' below it shows strings like 'You found the password!', 'alligator', and 'Usage: crackme-123 pas'. The bottom status bar provides system information and the date/time.

3. It will show you how it flows. So, zoom out the graph and start looking for the password.
4. You have discovered the password.



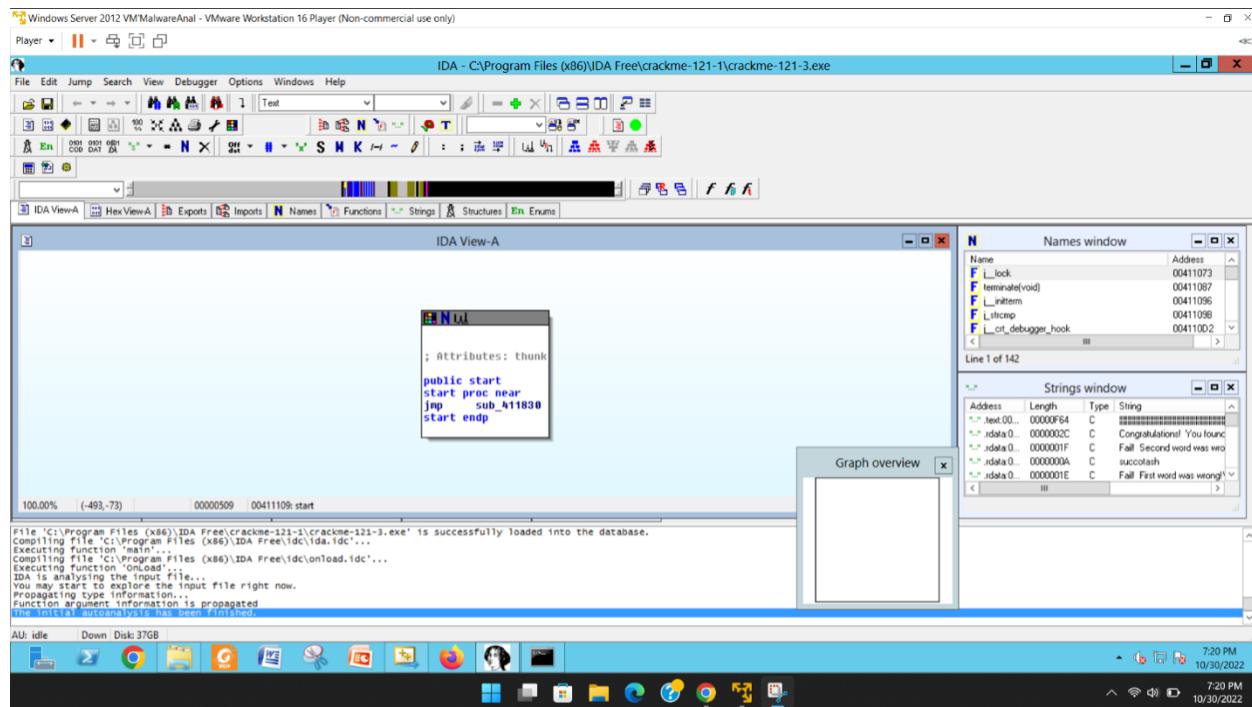
5. Open cmd and run crackme-121-2.exe, which will prompt you for a password.



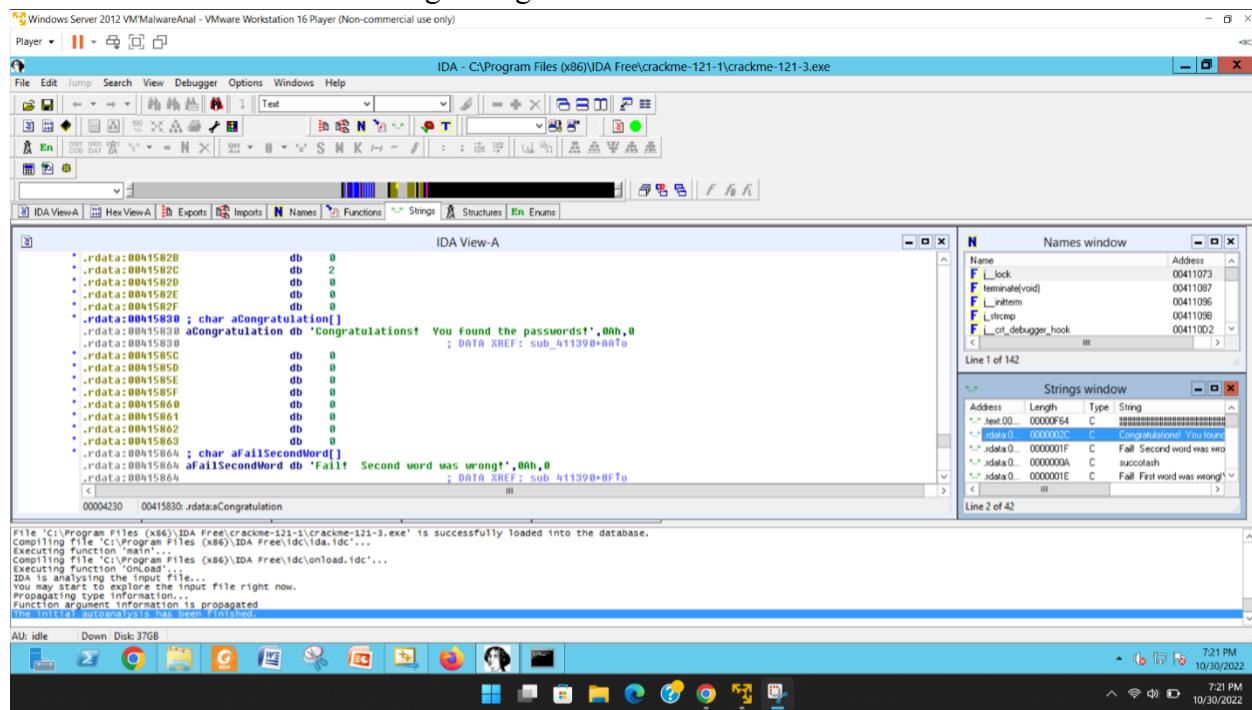


Crackme-121- 3.exe

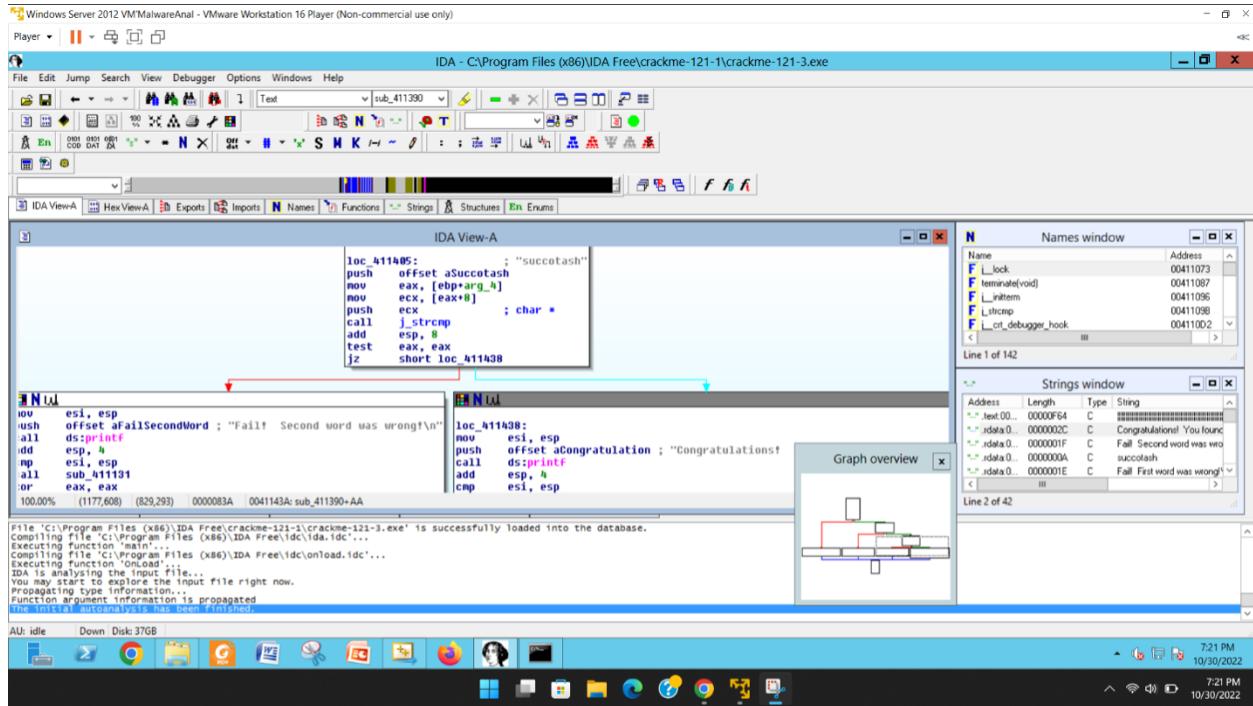
1. In IDAPro, open crackme-121-3.exe.



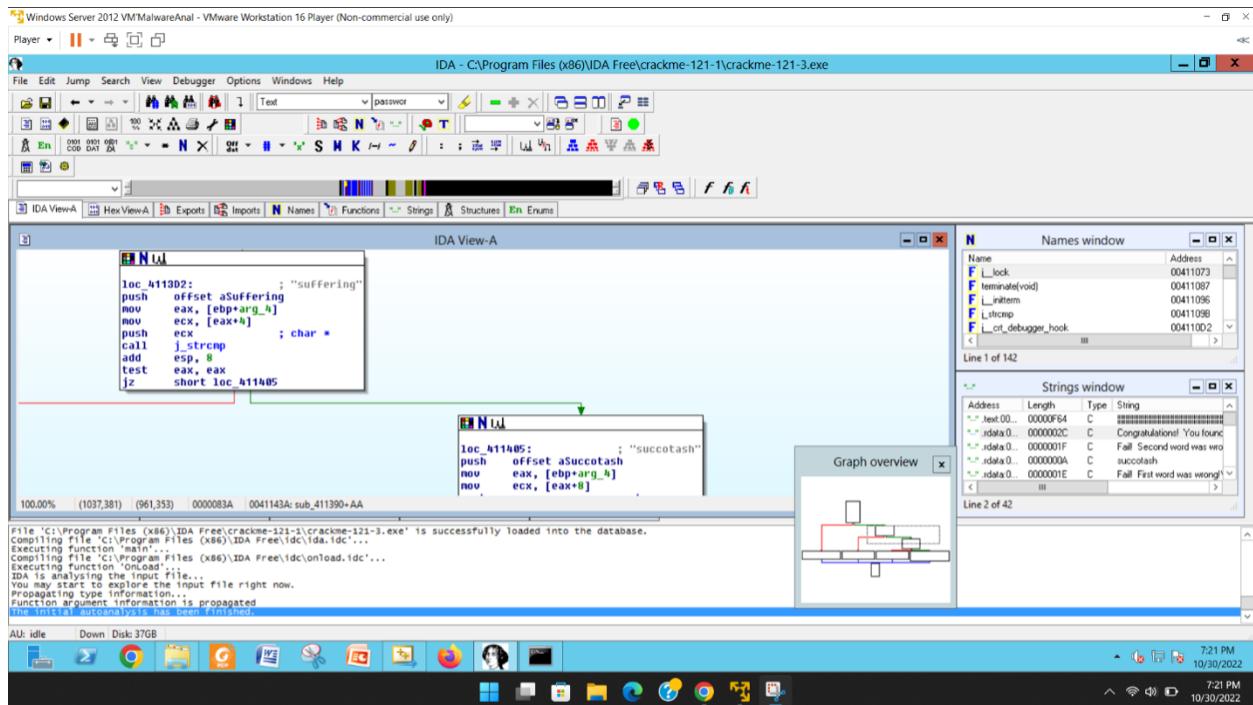
2. Double-click on the string "Congratulations."



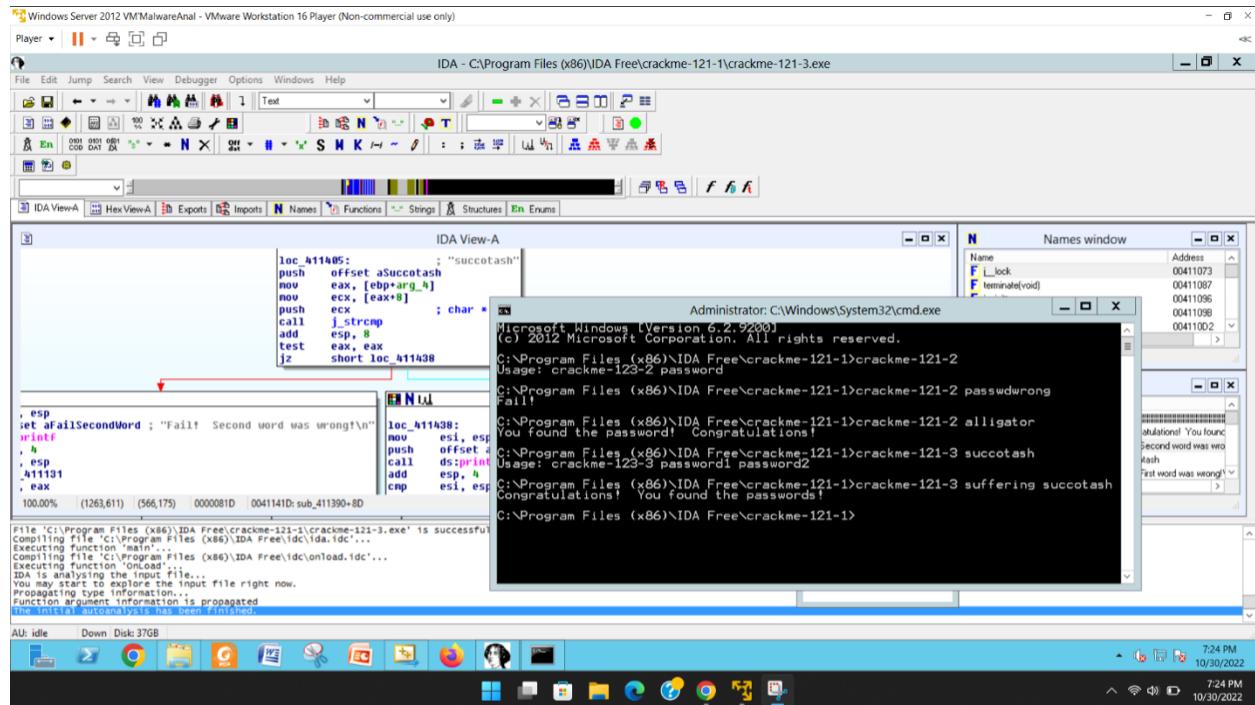
3. Now that you know where it is. When you double-click it, it will take you to a graphical depiction.



4. Find the password; you will be given two passwords that differ from the last exe file. We comprehend it by referring to the graph flow chart (password1=suffering, password2=succotash).

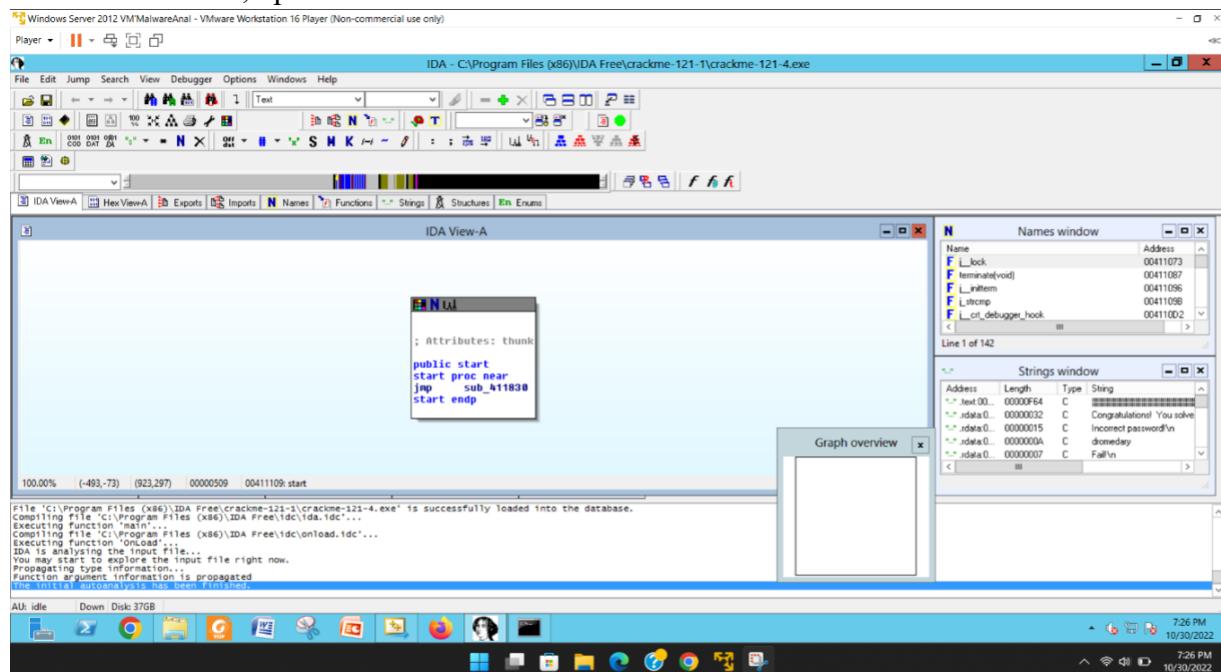


5. Open cmd and run the crackme-121-3.exe file (when we run it, it will prompt us to enter both passwords).

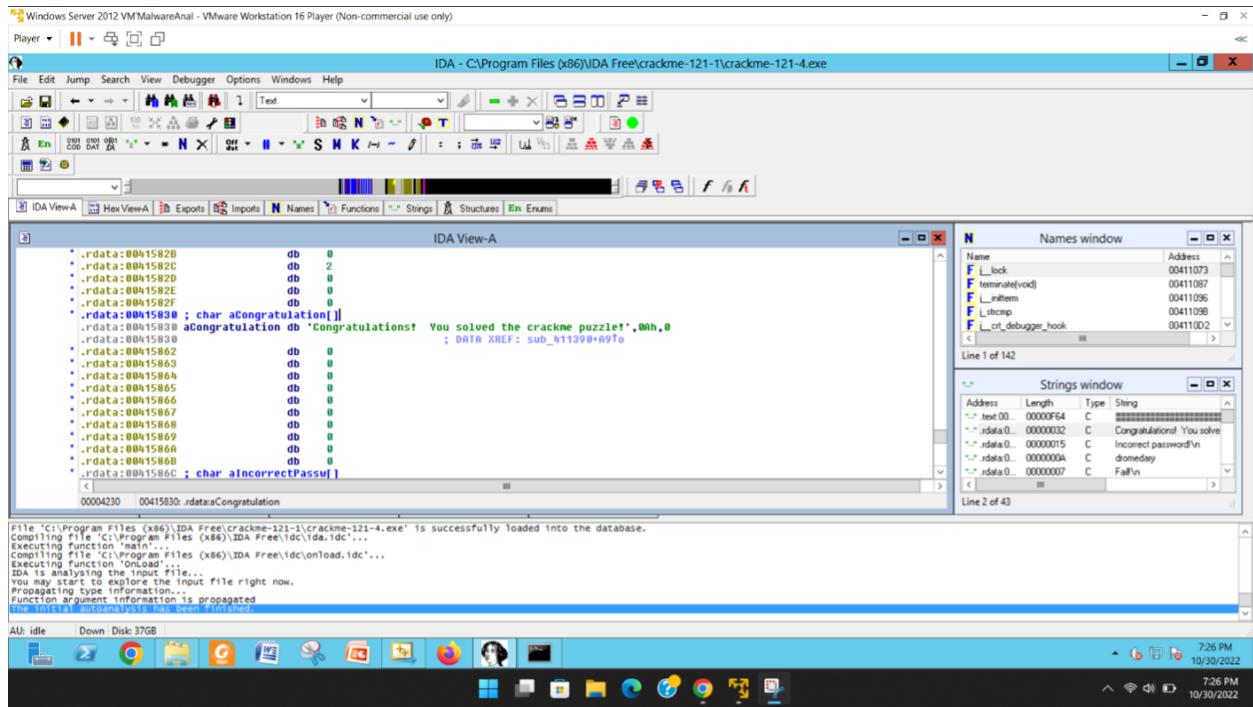


Crackme-121- 4.exe

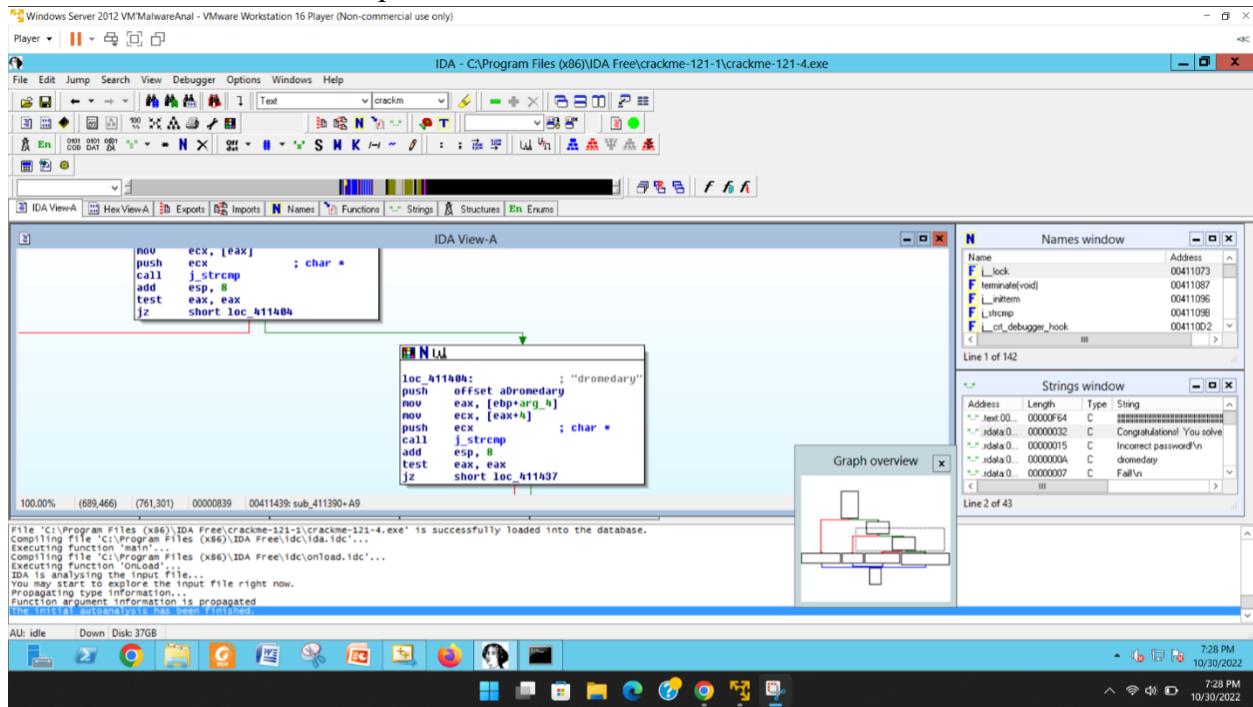
1. In IDAPro, open crackme-121-4.exe.



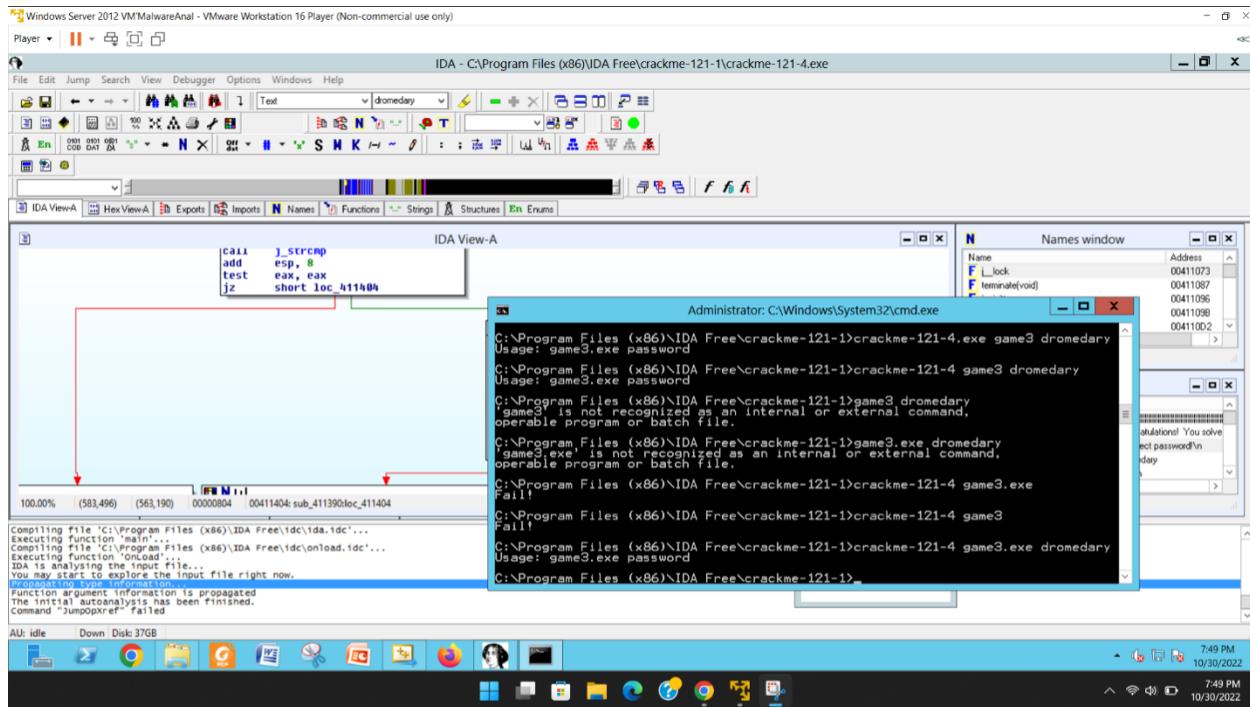
2. In the strings module, look for the string "Password discovered."



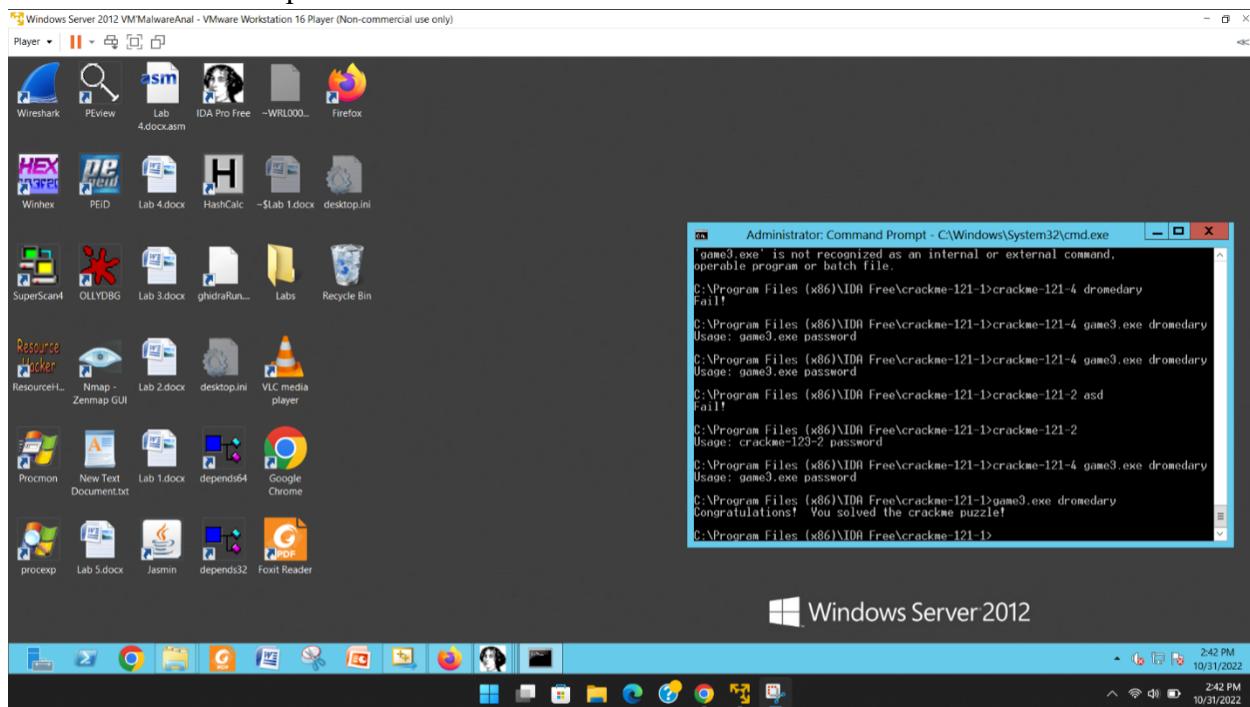
3. Double-click on the spot where we found it.



4. We obtained a password dromedary by following the graph flow, however, it would not execute in cmd. Launch cmd and run crackme-121-4.exe.



5. So we need to comprehend what it suggests to do here. As we can see, it uses the game3.exe password, thus we need to execute the game3.exe file and then supply the password, but we don't have game3.exe, so what do we do now? So we go to the crackme folder and rename crackme-121-4.exe to game3.exe so that we can run game3.exe and then enter the password.

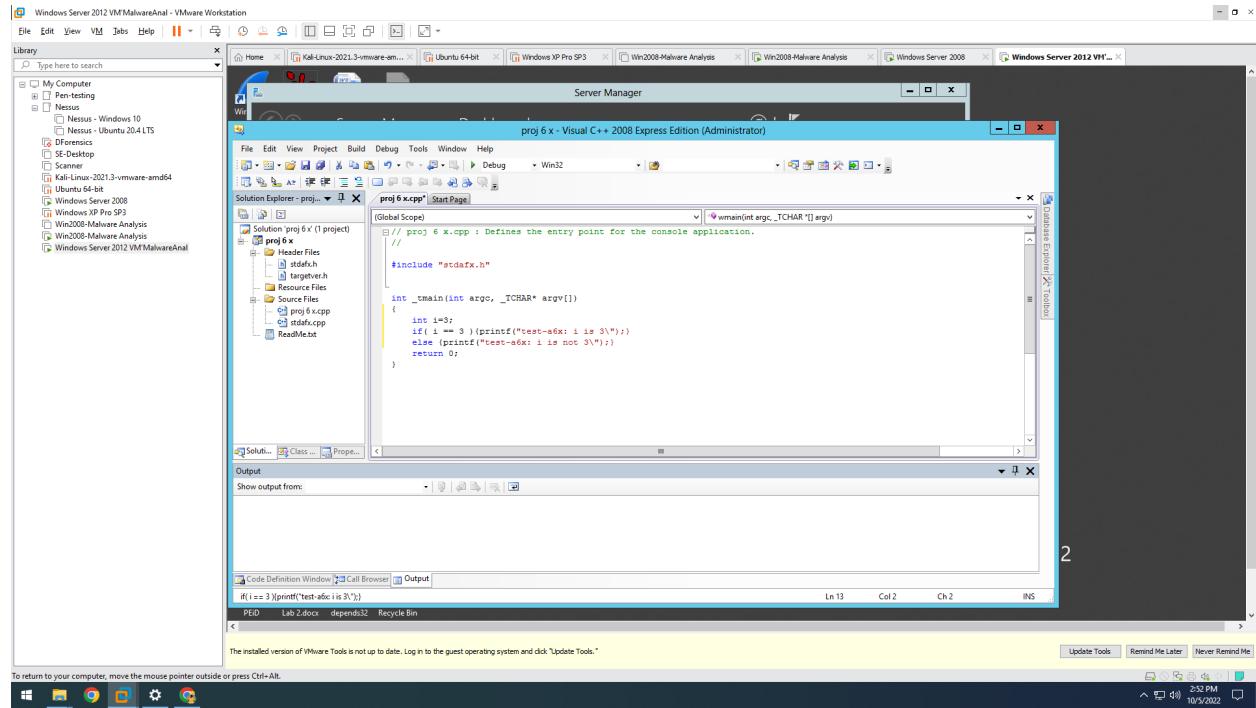


Proj 6: Disassembling C on windows part 3

Overview: In this lab, we will create a program utilizing if statements. And analyze it with an IDAPro disassembler to see how assembly language appears.

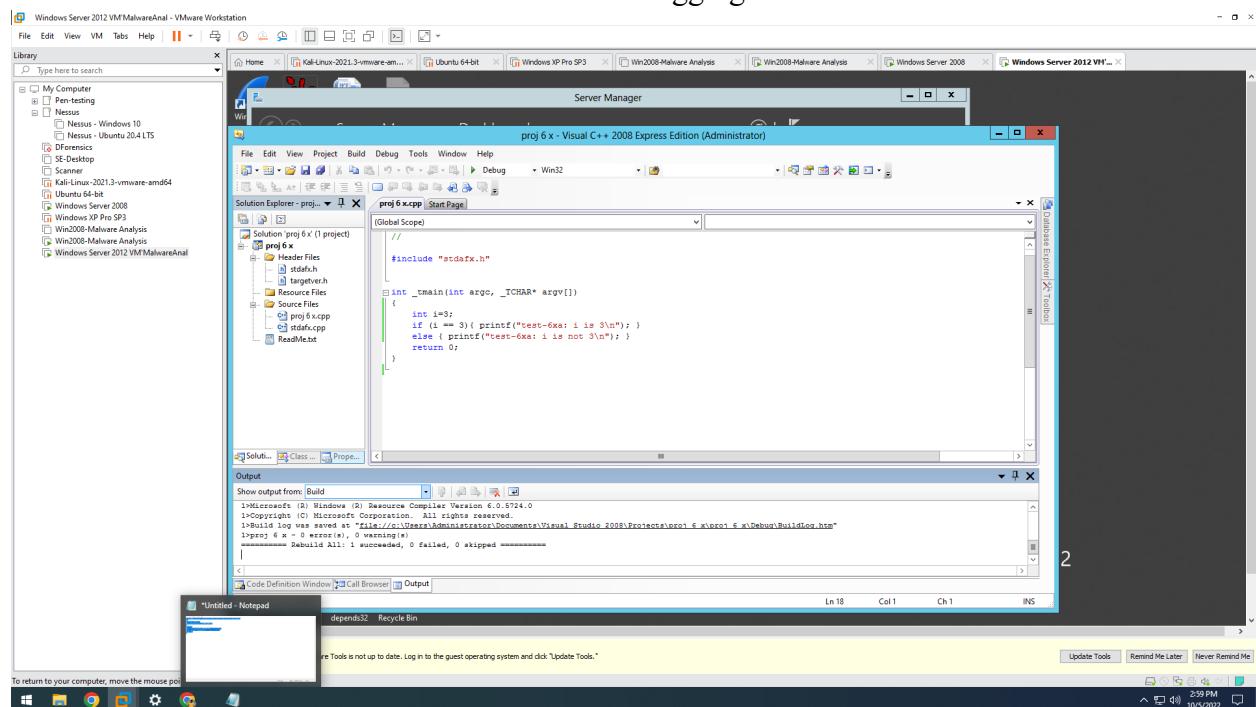
Open Visual C++ and create a c program. Select and accept the win32 console application after clicking on win32.

```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{
    int i=3;
    if (i == 3)
        { printf("YOURNAME-6xa: i is 3\n"); }
    else { printf("YOURNAME-6xa: i is not 3\n"); }
    return 0;
}
```

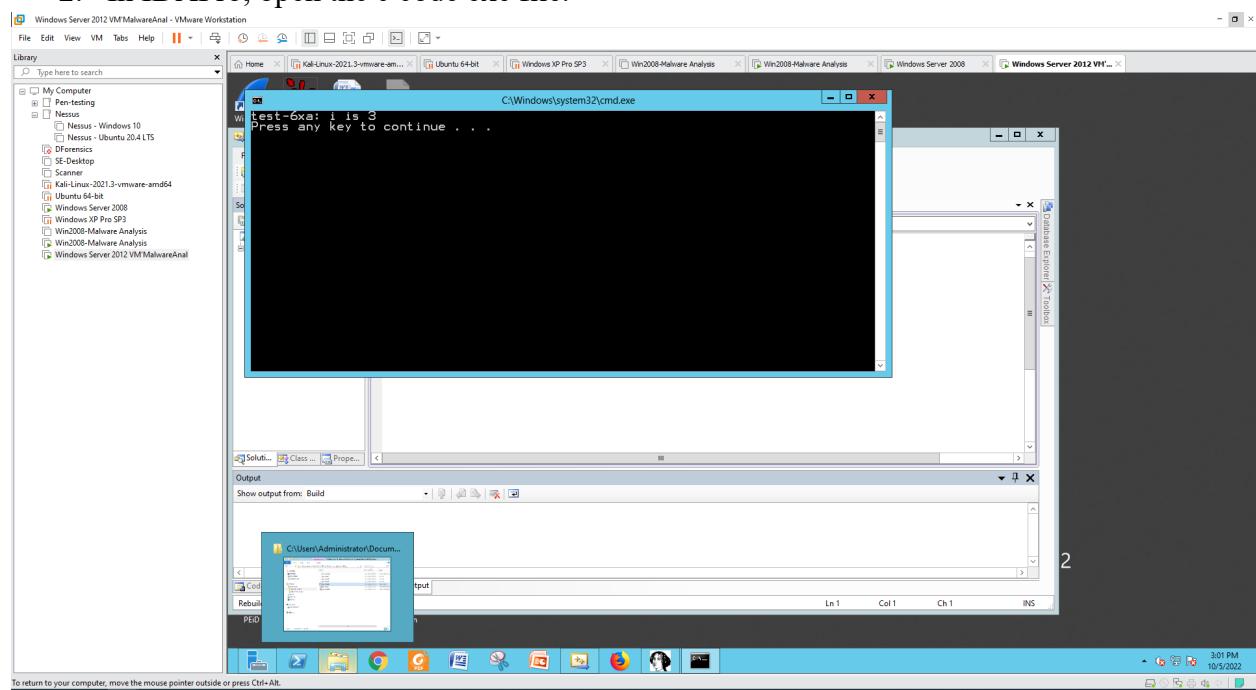


Procedure:

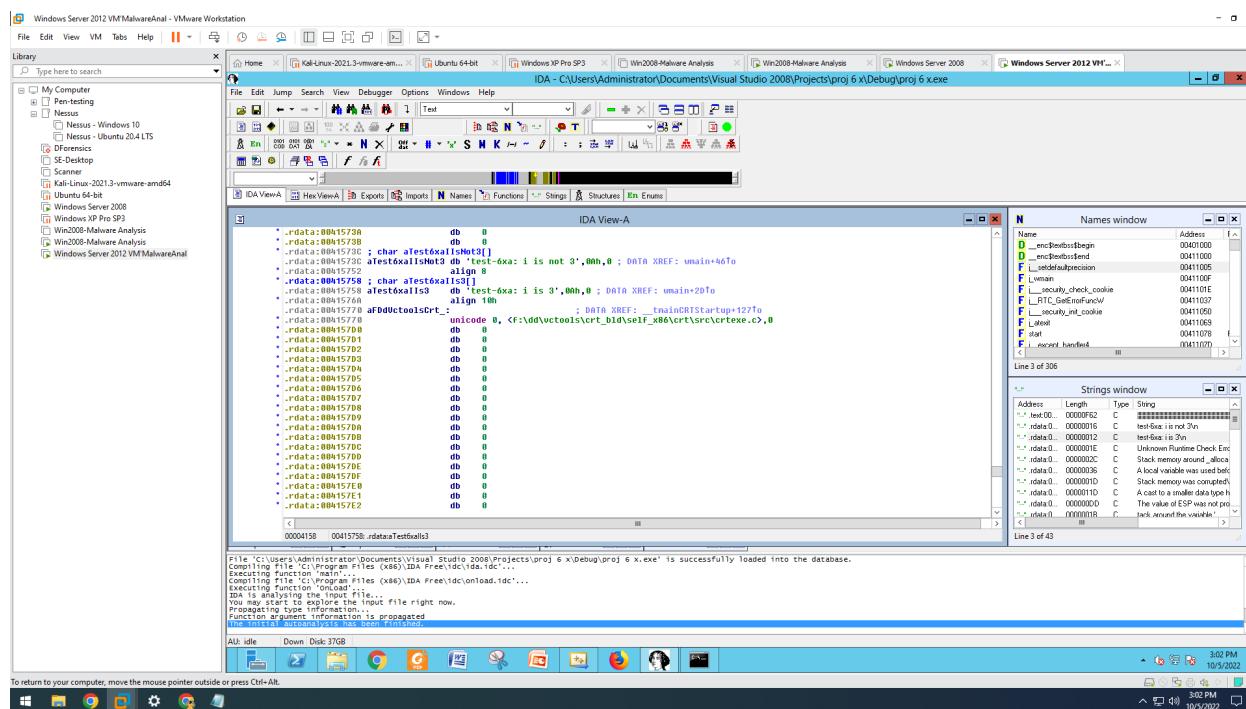
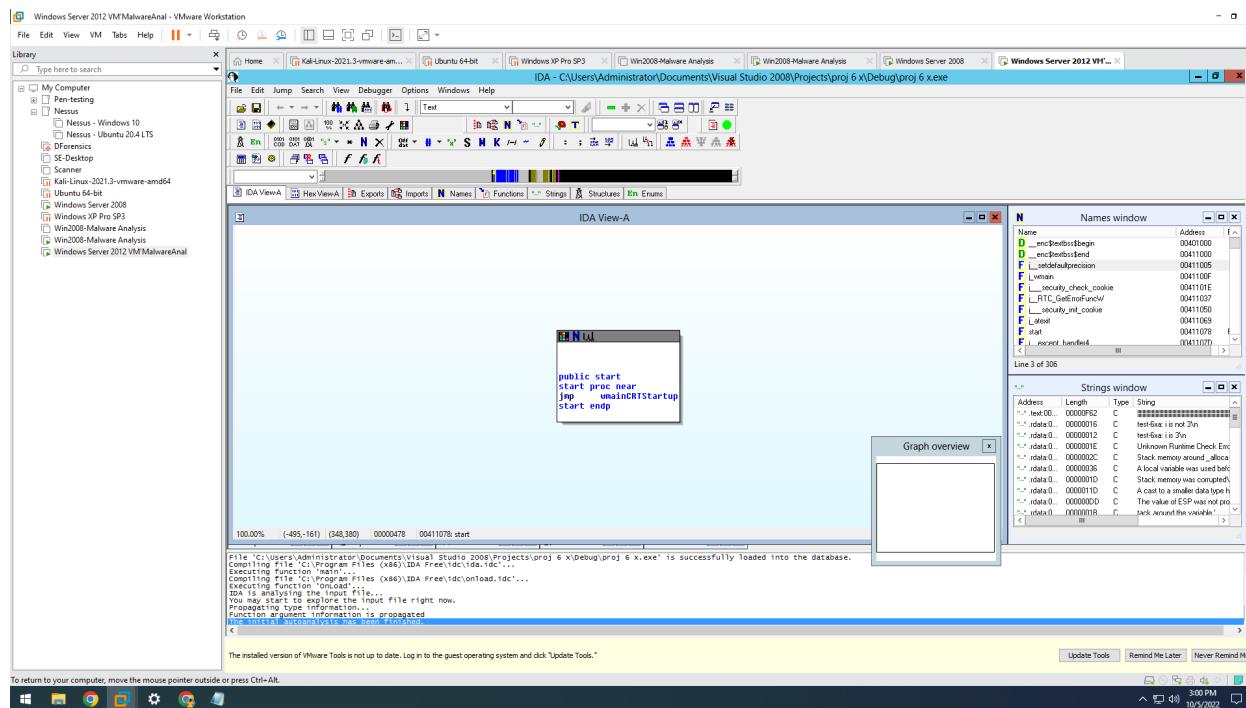
1. Build and execute the solution without debugging.

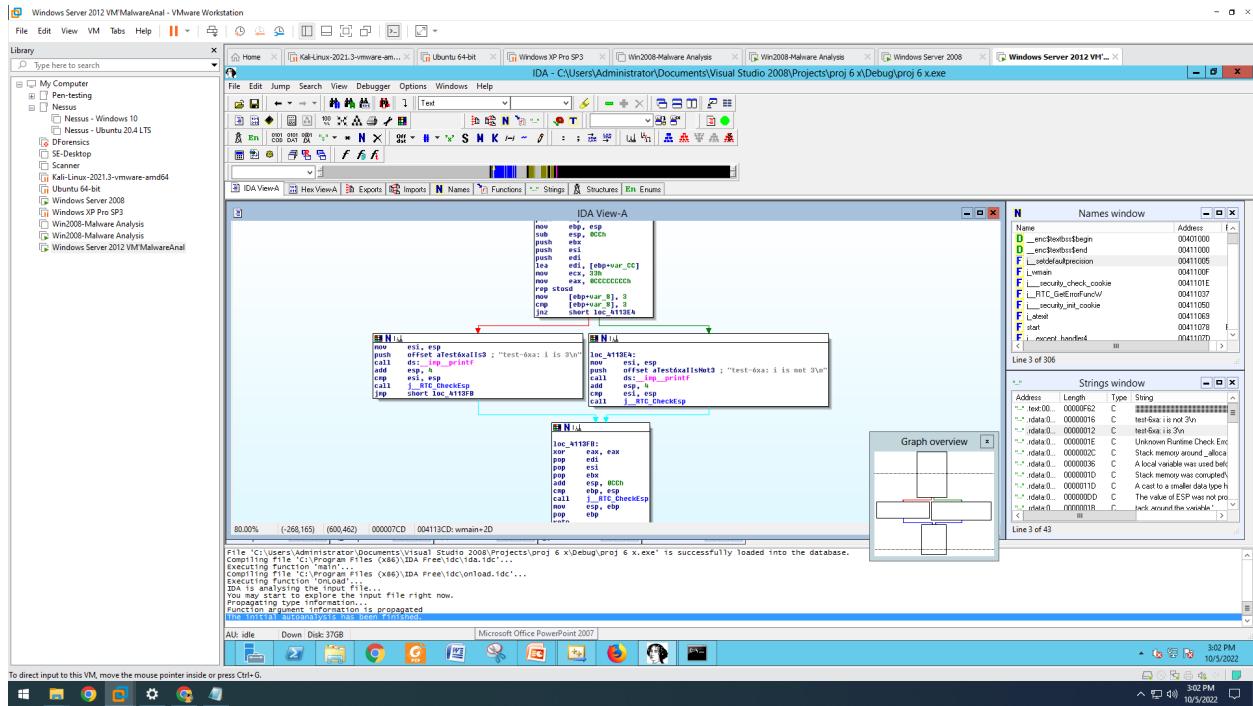


2. In IDAPro, open the c code exe file.



3. Find and select test-6ax in the string module. Then it will redirect you to the XREF address, where you should double-click.



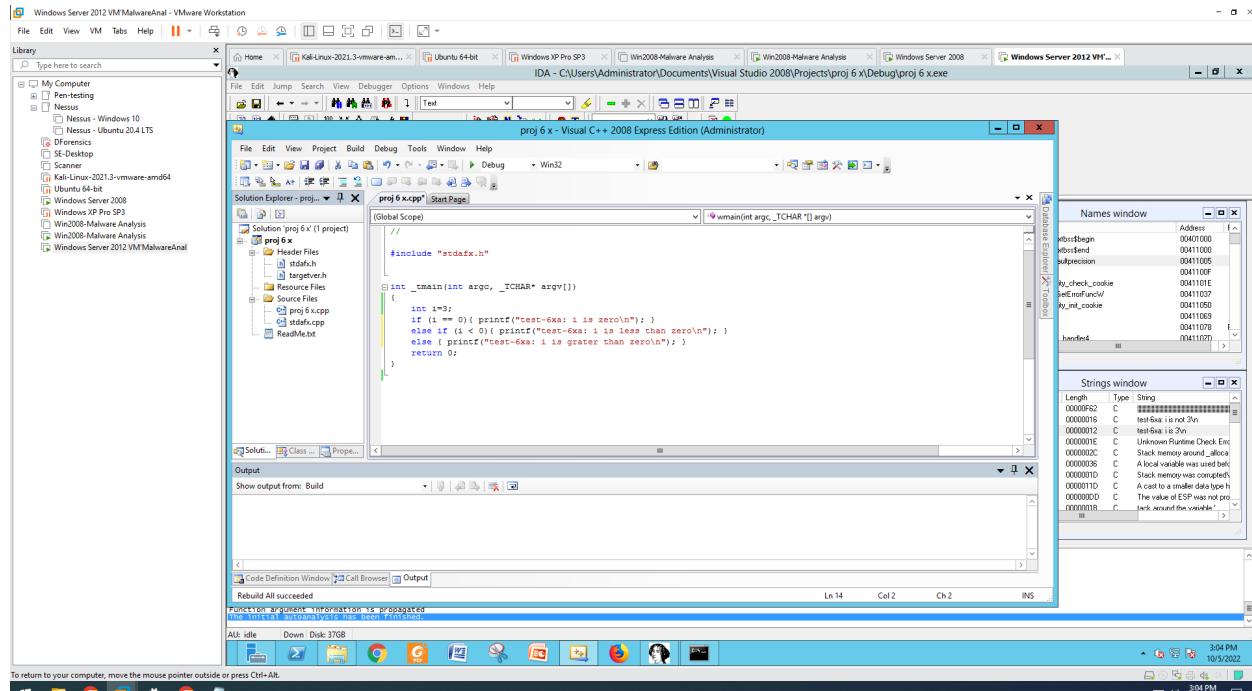


CHALLENGE

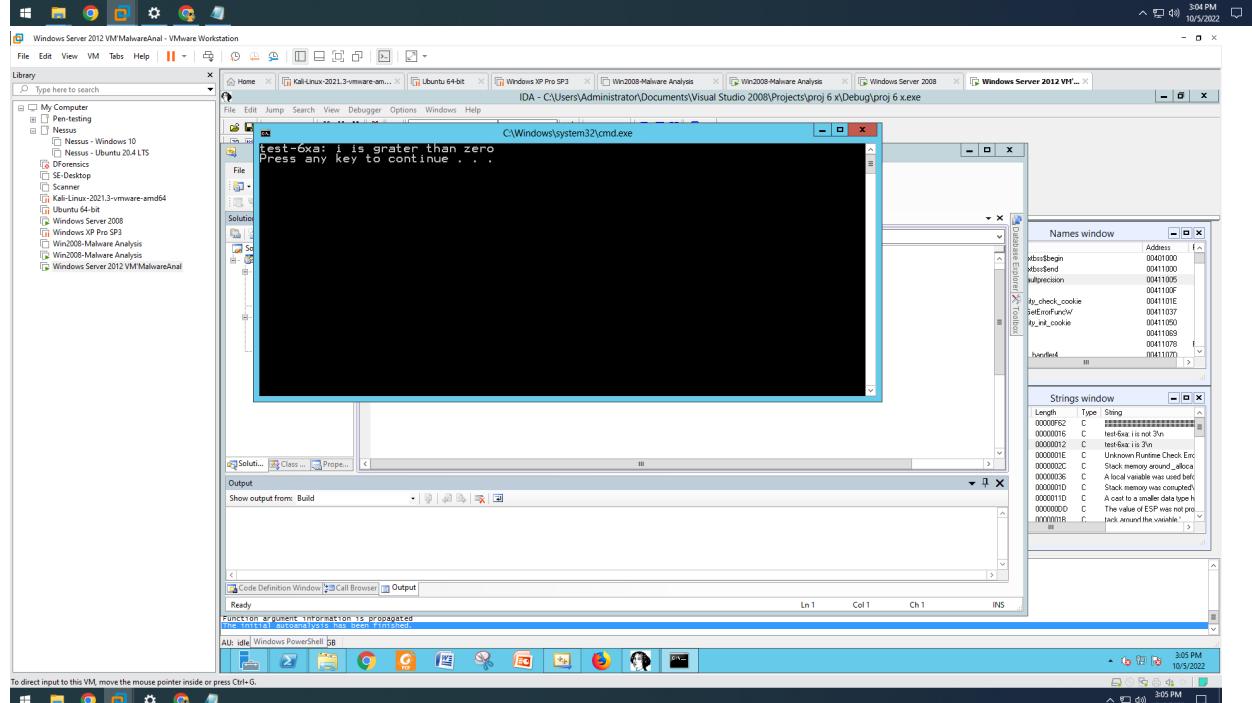
- Modify the c program to run a three-way test to determine whether a variable is less than, equal to, or higher than zero.
- In all three circumstances, print suitable messages mentioning your name.
- Compile and disassemble it to generate an assembly code identical to the one shown below.
- Create a c++ program with Visual Studio.

Procedure:

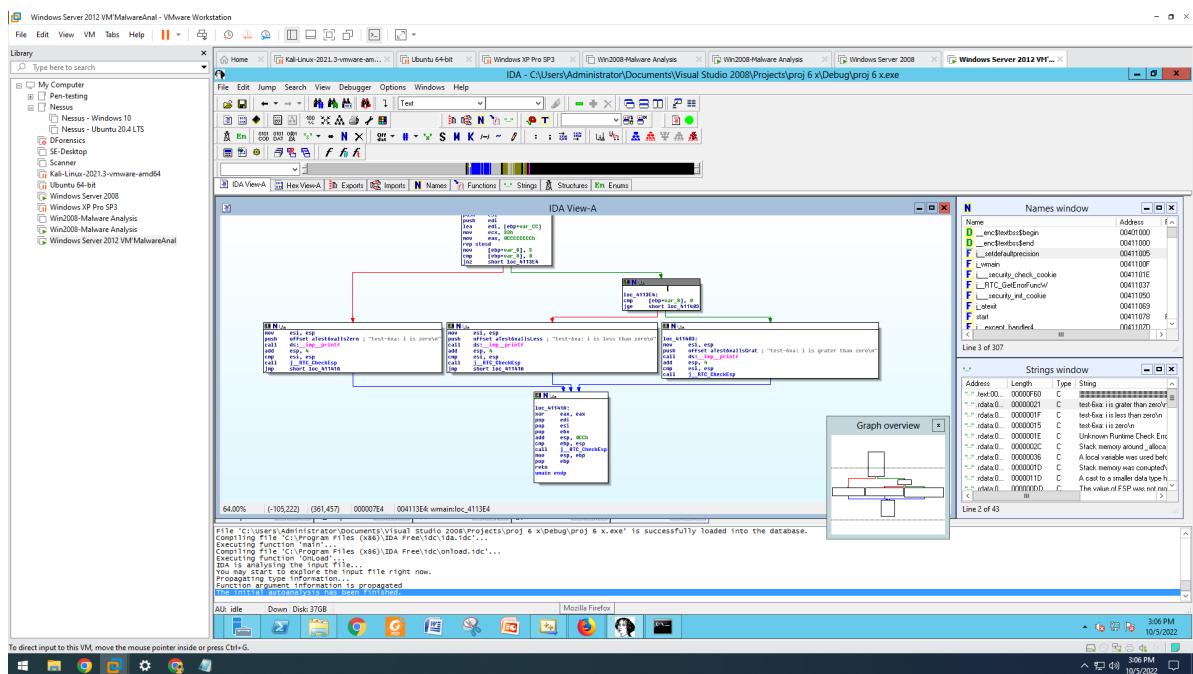
1. Build the solution and run it without debugging, like in the previous procedure, to determine the flow of code in IDAPro.



To return to your computer, move the mouse pointer outside or press Ctrl+Alt.



2. In IDAPro, open the c code exe file.
3. To do so, open IDAPro and search for a string in the strings module before clicking it. You will then be taken to the XERF location, so double-click it.

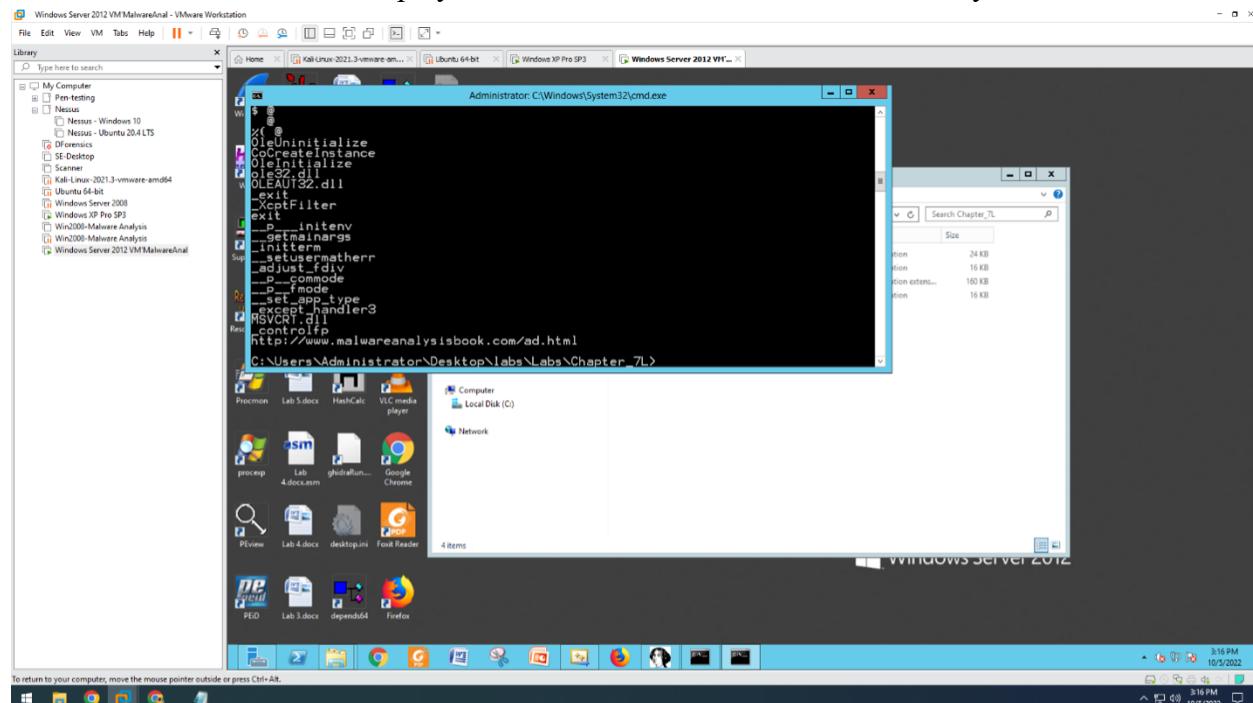


Proj 7: Analyzing malicious windows programs

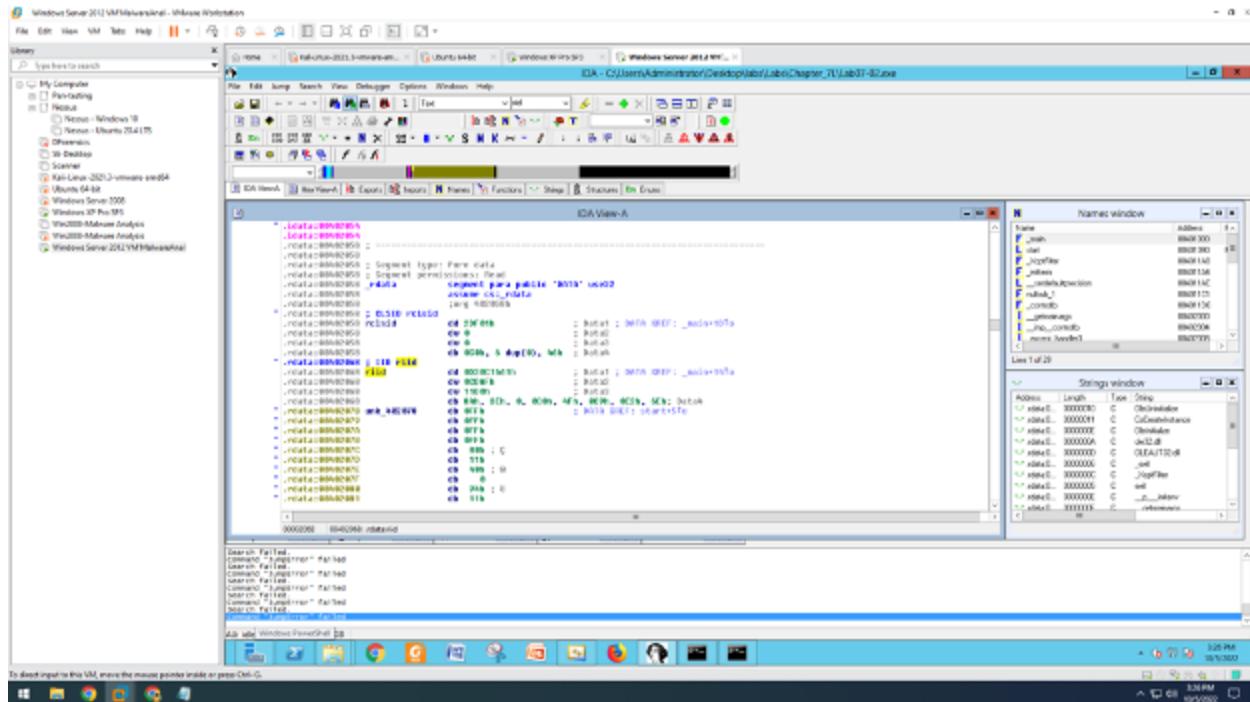
Overview: In this lab, I examined a malicious file that contained a virus and examined how it changed the registry after infecting the system.

Procedure:

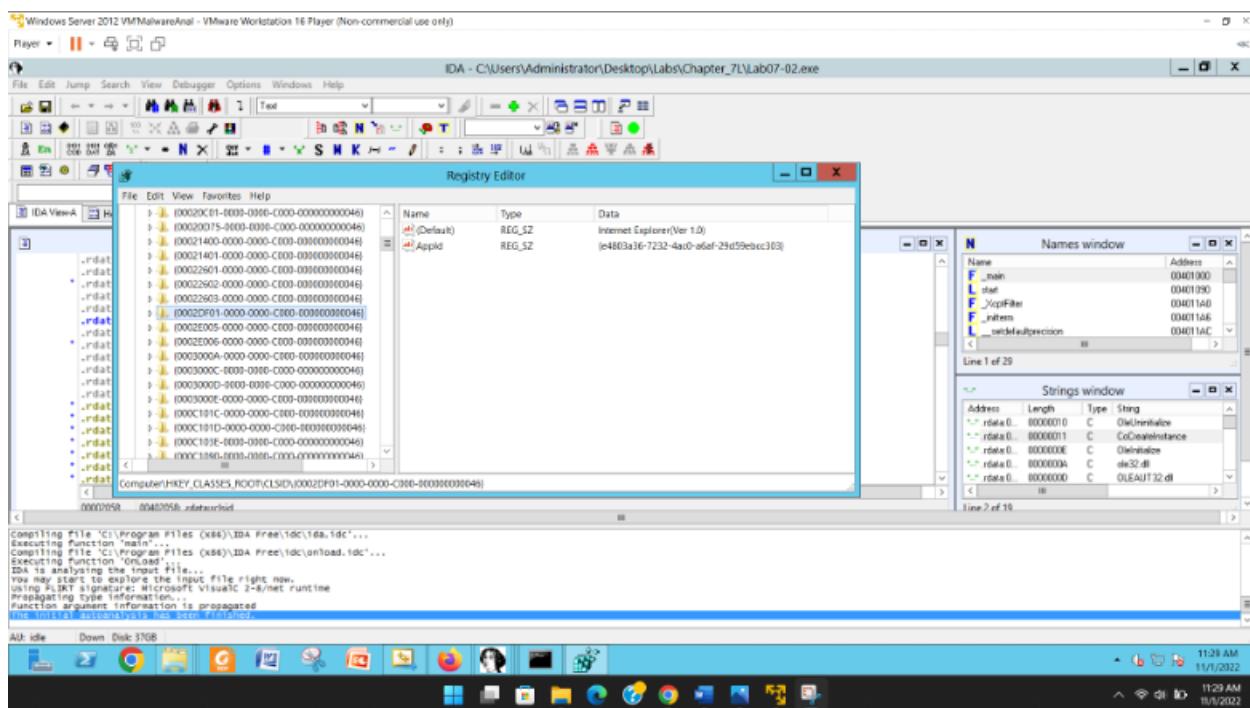
1. Use command to run strings. So, first, go to the lab Chapter 7 folder and launch strings Lab07-02.exe, as it displays a malicious website called malwareanlaysisbooks.com.



2. Enter Lab07-02.exe into IDAPro and look for the rclsid. It's already coded across numerous lines, as seen below, but you can reconstruct it from a single 128-bit value string beginning with 2DF0 and ending with 0046.



- To open the registry, go to Windows Run (ctrl+r), type Regedit, then option, locate, and type 0002D.

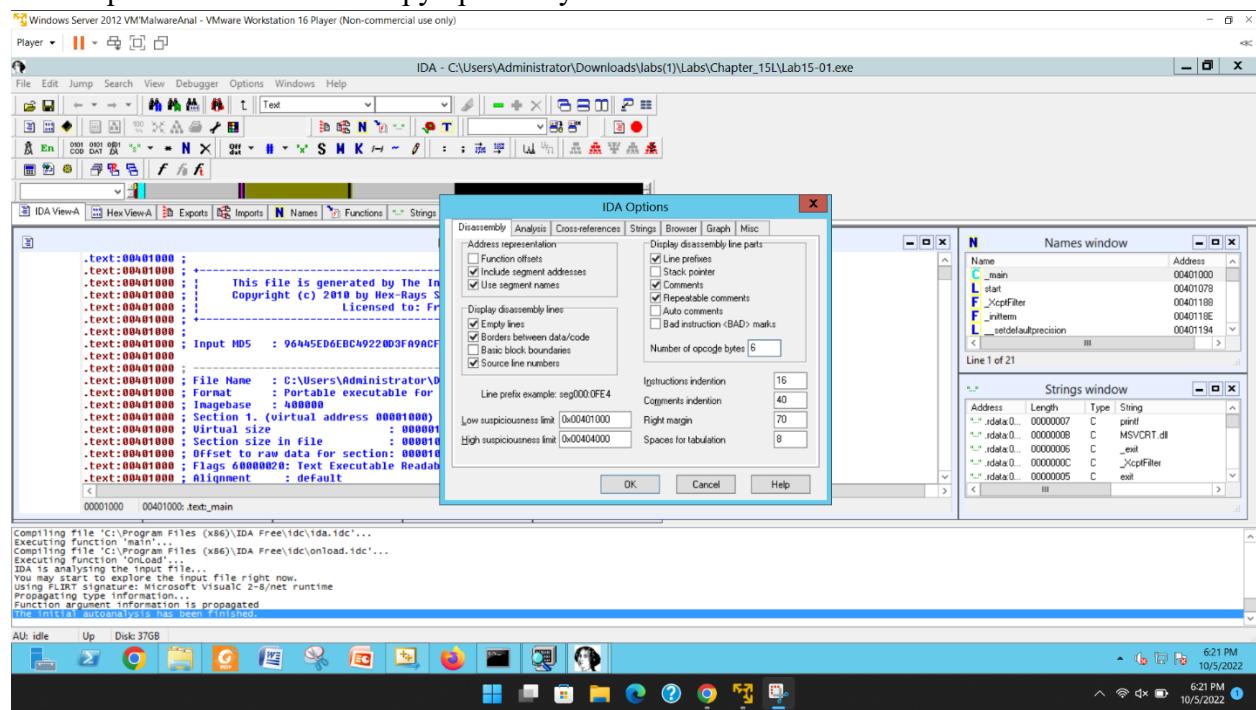


Proj 12: Anti-Disassembly

Overview: In this lab, I will utilize Lab15-01.exe in IDAPro for disassembly to overcome anti-disassembly.

Procedure:

1. Load Lab15-01.exe into IDAPro, then choose Options, General, and then "Line preference." Enter 6 copy opcode bytes and then click OK.



The screenshot shows the IDA Pro debugger interface with the following windows open:

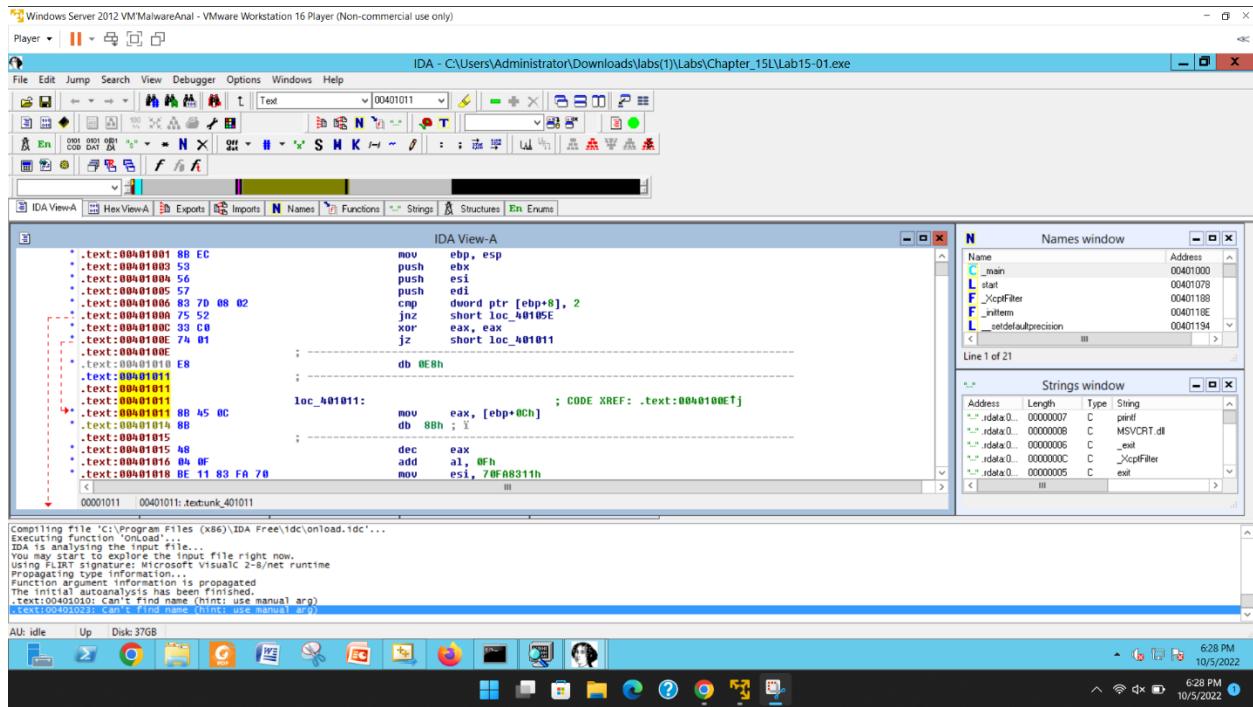
- IDA View-A**: The main assembly view showing the assembly code for the file `C:\Users\Administrator\Downloads\Jobs(1)\Labs\Chapter_15L\Lab15-01.exe`. The assembly code includes instructions like push ebx, push esi, push edi, cmp dword ptr [ebp+8], jnz short loc_40105E, xor eax, eax, and jz short near ptr loc_401010+1.
- Names window**: A list of symbols defined in the file, including `main`, `_XcptFilter`, `_infinim`, and `_serdefaultprecision`.
- Strings window**: A list of strings found in the file, such as `printf`, `MSVCRT.dll`, `_exit`, `_XcptFilter`, and `exit`.

The status bar at the bottom displays the message "AU: idle Up Disk: 37GB".

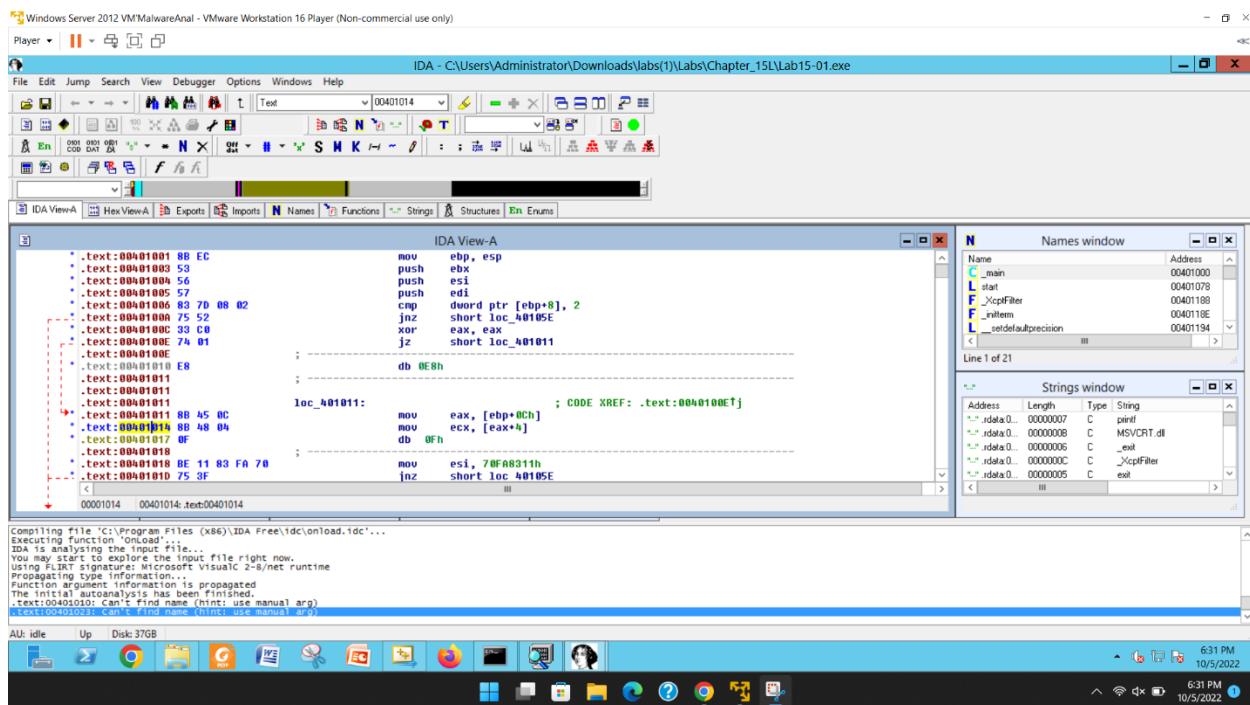
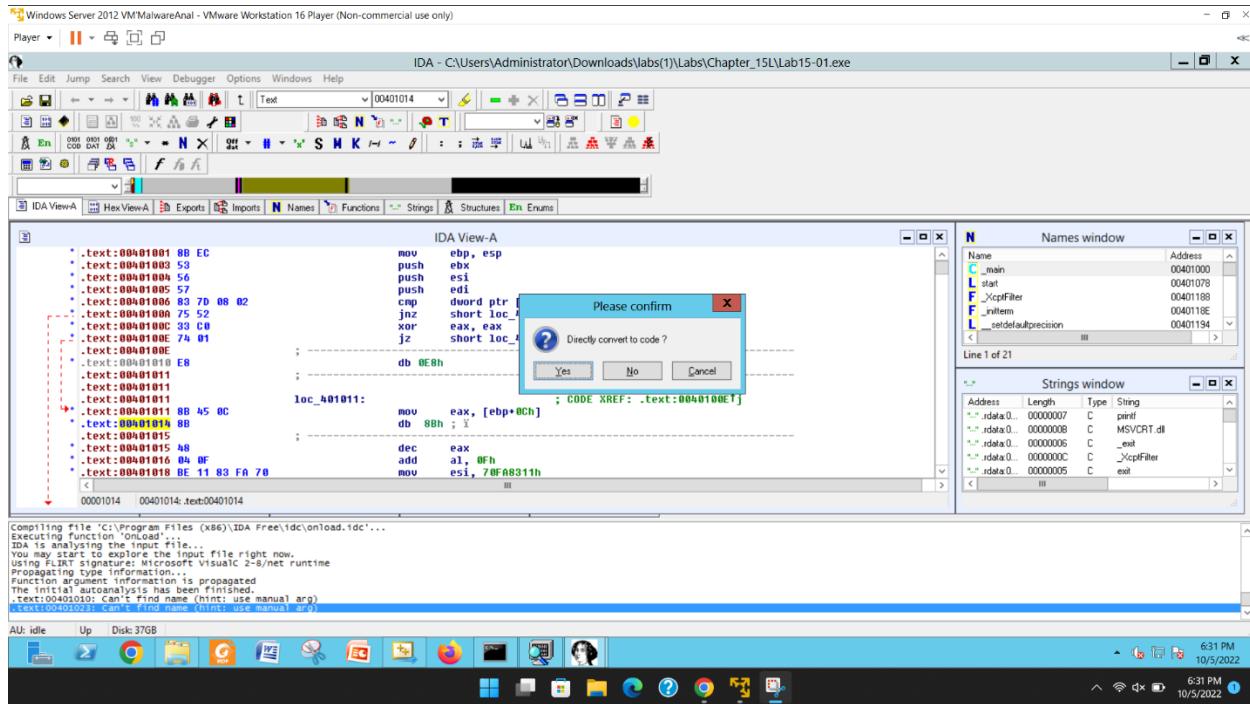
2. 401011 code is being fixed. The first issue is obvious: the instruction at location 40100E is a jz to 401010+1, or 401011. The condition is always true since the preceding instruction was xor eax, eax, hence the code will always skip over the byte at location 40100E. Click any of the 401010 addresses in the left column to adjust the disassembly. They all turn yellow at the same time. To convert these five bytes to data, hit d on the keyboard. A box with the words "Please confirm" appears. Select Yes. Following the jz instruction, the code now displays five bytes of data, as shown below.

The screenshot shows the IDA Pro debugger interface running on Windows Server 2012. The main window displays assembly code for the file C:\Users\Administrator\Downloads\labs(1)\Labs\Chapter_15\L\Lab15-01.exe. The assembly code includes instructions like mov, push, and cmp, along with labels such as unk_401011 and loc_40105E. The Names window on the right lists symbols like main, start, _XcpFilter, _inithrm, and _seidelprecision. The Strings window below it shows various string literals. The bottom status bar indicates the application is idle.

- Because the jz restarts execution at 401011, we must instruct IDA to interpret that address as code.



- Click the 401011 addresses in the left column. To convert these five bytes to code, hit c on the keyboard.
- As shown below, three bytes beginning at 401011 are now interpreted as a MOV instruction.
- However, the byte at 401014 is still interpreted as data, rendering the disassembly below it incorrect.
- Click the 401014 addresses in the left column. To convert these five bytes to code, hit C on the keyboard. A box with the words "Please confirm" appears. Select Yes.
- This directs IDA to successfully disassemble another instruction while leaving another byte abandoned and interpreted as data at 401017, as illustrated below.



9. Click the 401017 addresses in the left column. To convert these five bytes to code, hit c on the keyboard. A box with the words "Please confirm" appears. Select Yes.
10. Finally, as shown below, IDA displays the right code, with no stray "db" bytes in the center.

Windows Server 2012 VM/MalwareAnal - VMware Workstation 16 Player (Non-commercial use only)

Player | || | X

File Edit Jump Search View Debugger Options Windows Help

IDA View-A HexViewA Exports Imports Names Functions Strings Structures Enums

IDA View-A

```

.text:00401001 8B EC      mov    ebp, esp
.text:00401002 5B          push   ebx
.text:00401003 56          push   esi
.text:00401004 57          push   edi
.text:00401005 7D 70 08 02  cmp    dword ptr [ebp+8], 2
.text:00401006 75 52        jnz    short loc_40105E
.text:00401007 33 C0        xor    eax, eax
.text:00401008 74 01        ja     short loc_401011
.text:00401009 48           dd    0E8h
.text:00401010 E8          ; CODE XREF: .text:0040100E!
.text:00401011 8B A5 0C      mov    eax, [ebp+8Ch]
.text:00401012 8B A8 04      mov    ecx, [eax+4]
.text:00401013 8D BE 11      movsx  edx, byte ptr [ecx]
.text:00401014 75 70        cmp    edx, 70
.text:00401015 75 5F        jnz    short loc_40105E
.text:00401016 33 C0        xor    eax, eax
.text:00401017 00             dd    00000000

```

Names window

Name	Address
C_main	00401000
L_start	00401078
F_XcpFilter	00401188
F_initem	0040118E
L_setdefaultprecision	00401194

Strings window

Address	Length	Type	String
idaa0. 00000007	C	perfI	
idaa0. 00000008	C	MSVCR7.dll	
idaa0. 00000006	C	_exit	
idaa0. 0000000C	C	_XcpFilter	
idaa0. 00000005	C	exit	

Compiling file 'C:\Program Files (x86)\IDA Free\idc\onload.idc'...
Executing function 'OnLoad'...
IDA: Compiling file 'onload.idc'...
You may start to explore the input file right now.
Using FLIRT signature: Microsoft VisualC 2-8/net runtime
Propagating type information...
Function argument information is propagated
The initial autoanalysis has been finished.
.text:00401000: Can't find name (hint: use manual arg)
.text:00401001: Can't find name (hint: use manual arg)
.text:00401002: Can't find name (hint: use manual arg)
.text:00401003: Can't find name (hint: use manual arg)
.text:00401004: Can't find name (hint: use manual arg)

AU: idle Up Disk: 37GB 6:32 PM 10/5/2022

6:32 PM 10/5/2022

11. Use the same method for 401023, 401024, 401027, 40102A, 40102E, 401031, 401033, 401037, 401038, 40104C, and 401063.

Windows Server 2012 VM/MalwareAnal - VMware Workstation 16 Player (Non-commercial use only)

Player | || | X

File Edit Jump Search View Debugger Options Windows Help

IDA View-A HexViewA Exports Imports Names Functions Strings Structures Enums

IDA View-A

```

.text:00401010 83 FA 70      cmp    edx, 70h
.text:00401011 75 3F        jnz    short loc_40105E
.text:00401012 33 C0        xor    eax, eax
.text:00401013 74 01        jz     short near ptr unk_401024

```

unk_401024

```

db 0E8h
db 8Bh ; ; CODE XREF: .text:00401021!
db 45h ; E
db 0Ch

```

Names window

Name	Address
C_main	00401000
L_start	00401078
F_XcpFilter	00401188
F_initem	0040118E
L_setdefaultprecision	00401194

Strings window

Address	Length	Type	String
idaa0. 00000007	C	perfI	
idaa0. 00000008	C	MSVCR7.dll	
idaa0. 00000006	C	_exit	
idaa0. 0000000C	C	_XcpFilter	
idaa0. 00000005	C	exit	

You may start to explore the input file right now.
Using FLIRT signature: Microsoft VisualC 2-8/net runtime
Propagating type information...
Function argument information is propagated
The initial autoanalysis has been finished.
.text:00401020: Can't find name (hint: use manual arg)
.text:00401021: Can't find name (hint: use manual arg)
.text:00401022: Can't find name (hint: use manual arg)
.text:00401023: Can't find name (hint: use manual arg)
.text:00401024: Can't find name (hint: use manual arg)

AU: idle Up Disk: 37GB 6:35 PM 10/5/2022

6:35 PM 10/5/2022