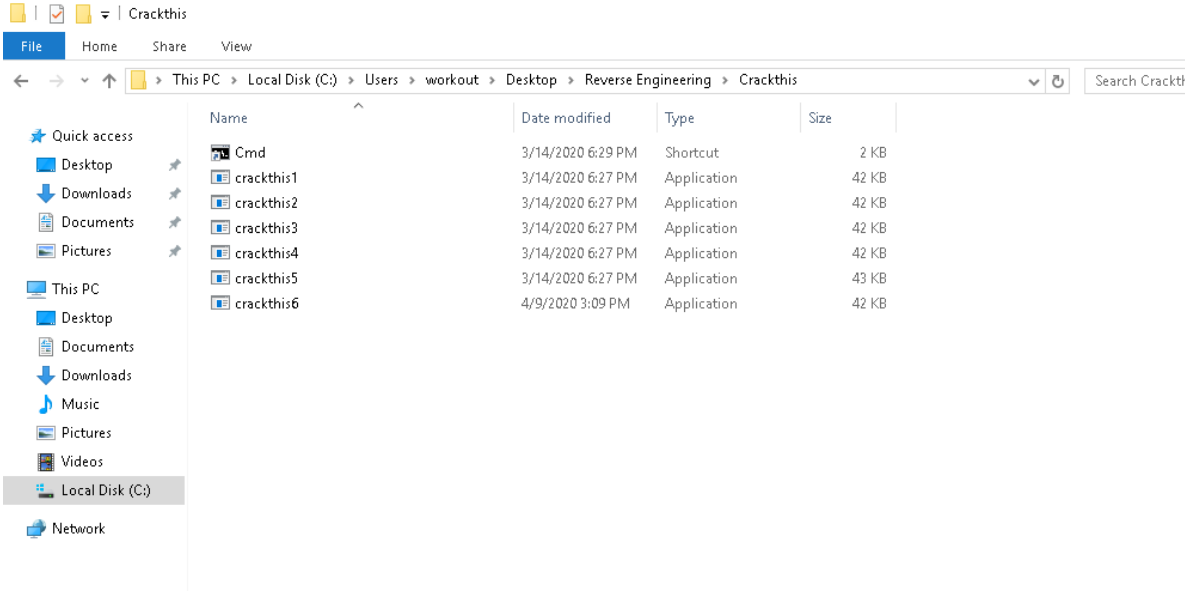
Project Reversus : Teacher Workout

Introduction:

In this workout, students will learn how to analyze and reverse engineer a program.
A module will be completed if the program outputs "Access granted!"
These instructions will step you through two of the six modules.

The Mission:

To begin, students should browse to the Crackthis folder.
First, click on the folder called Reverse Engineering and then the one called Crackthis.
It should look like the following screenshot.



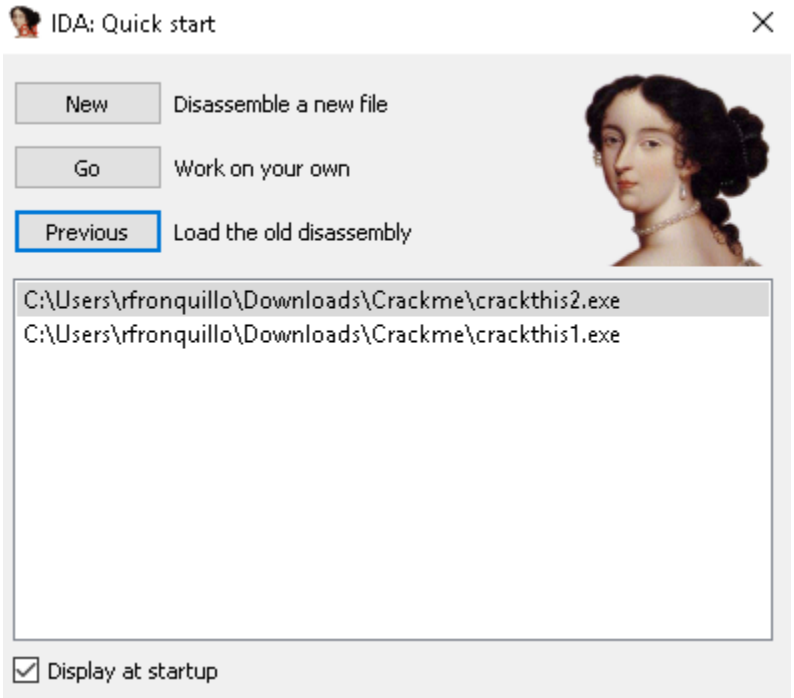
Crackthis1

Let's begin by double clicking on the program and seeing what it does. If it gives you a warning, just click on run.
You should be presented with a command line window like the following screenshot.

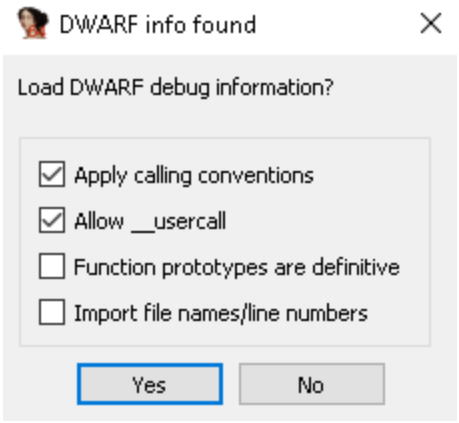
Try typing in anything. It should give you an "Access denied!"

There's no source code, so that means you'll have to use a tool that can disassemble the program and give a close representation of
what the program should look like in Assembly code.

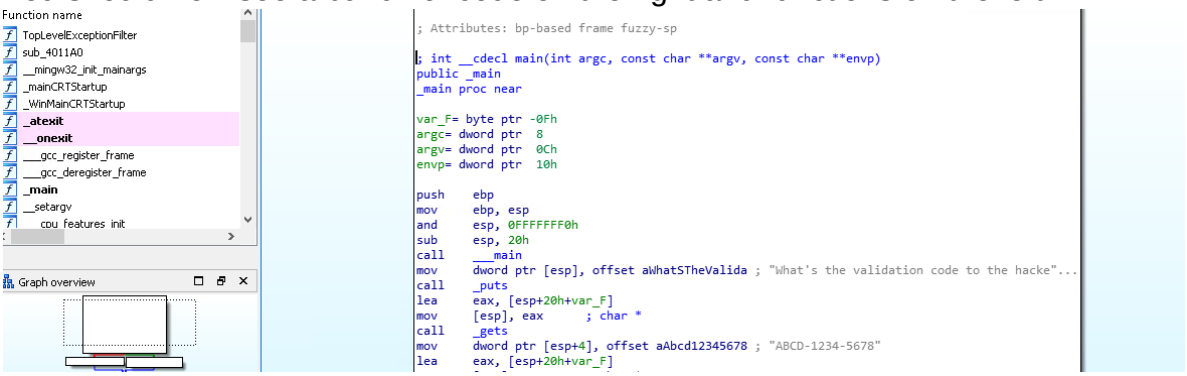
For this demo, we'll be using a tool called IDA Freeware. You'll find it on your desktop.
Double-click on the program and it will give you a few options. Click on new to disassemble a new file.



Now it will ask you what file you want to analyze, select crackthis1 and then hit OK when it asks to load a new file.
When it asks for DWARF INFO, keep the defaults and hit Yes.



You should now see a bunch of code on the right and functions on the left.



For now, just focus on the code on the right. What's most important is to get a feel for how the program works and flows instead of focusing on each individual instruction.
Take note of an mov, cmp, or call instruction in this program. Mov is an instruction that will move data from one location to another. Compare will compare values between data locations. Call will access a function.

Perhaps you've already noticed that the text you saw when opening up the program is being moved into a location in memory. A few lines down, you can see another string being moved into a memory location. Afterwards, a call to a function named strcmp is made. String compare looks at two strings and checks to see if they are the same or not. You can infer that maybe the string that looks like a validation code might be what's being compared.

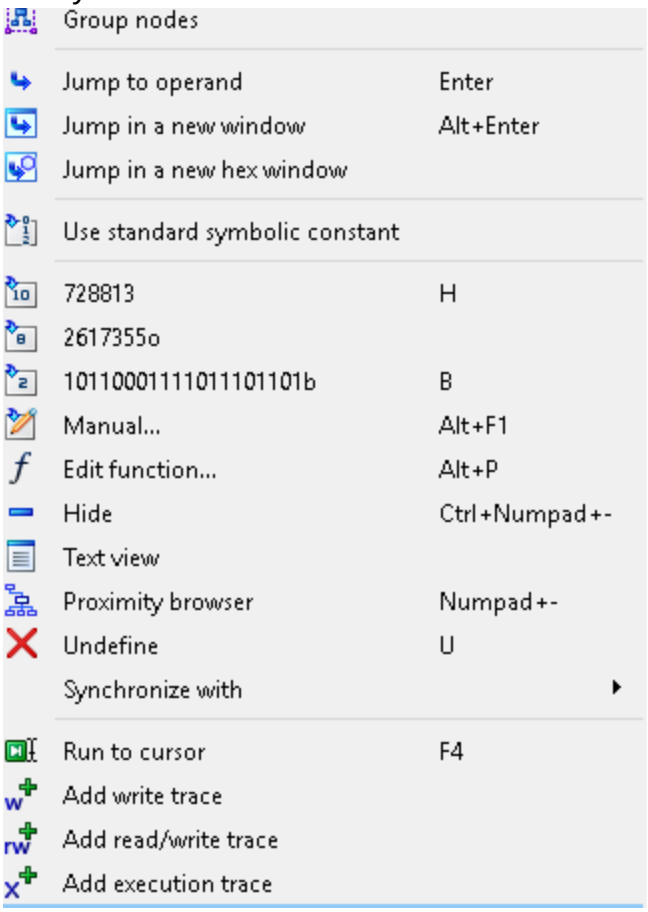
```
mov     dword ptr [esp], offset aWhatSTheValida ; "What's the validation code to the hacke"...
call    _puts
lea     eax, [esp+20h+var_F]
mov     [esp], eax      ; char *
call    _gets
mov     dword ptr [esp+4], offset aAbcd12345678 ; "ABCD-1234-5678"
lea     eax, [esp+20h+var_F]
mov     [esp], eax      ; char *
call    _strcmp
test    eax, eax
jnz     short loc_40145C
```

Open up crackthis1 again and put in the string **ABCD-1234-5678**. You should now be greeted with an "Access granted!" That concludes the first module.

Crackthis2

Crackthis2 can be solved with the same steps as above. There are a few small differences in that instead of strcmp, there's now just a cmp instruction being used. Not only that, but there's no string code this time. Instead, there's a %d being moved into a memory location. In C programming, %d is used read and print out an integer value. Look at the cmp instruction as well, there's a value called B1eed in hex being compared.

You can try putting this into the program but you'll be denied access. What you need to do is convert the hex value into decimal. To do this in IDA, just hover over the value and right click on it. Near the top of the menu, you'll see a list of values in binary and decimal.



The one in decimal is the one with the H next to it. If you put that value you in, you should receive an "Access granted!" That's the end of the second module.