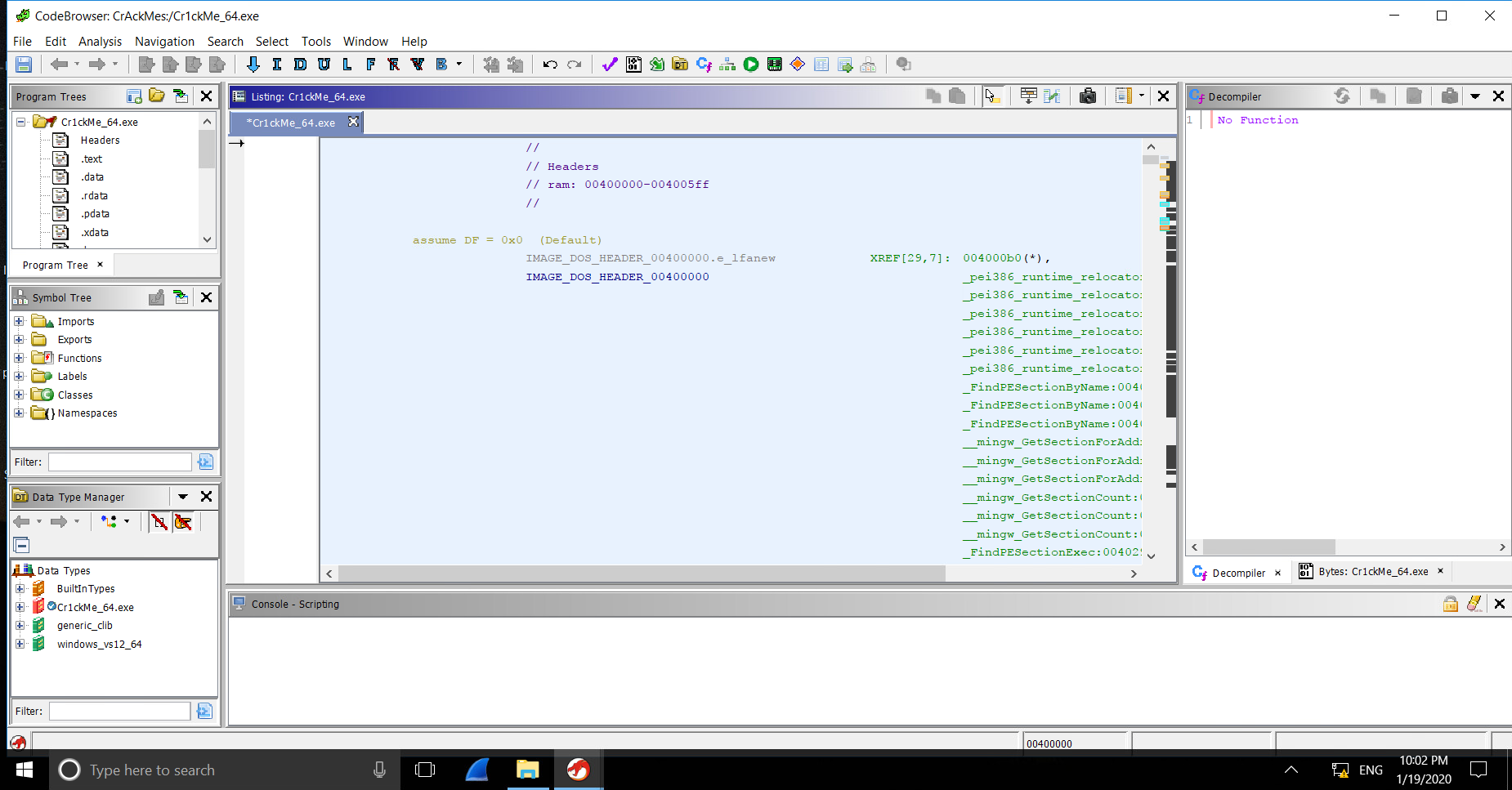
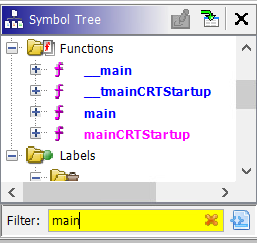
Week 15 – Cr1ckMe Solution

Cr1ckMe (Level One)

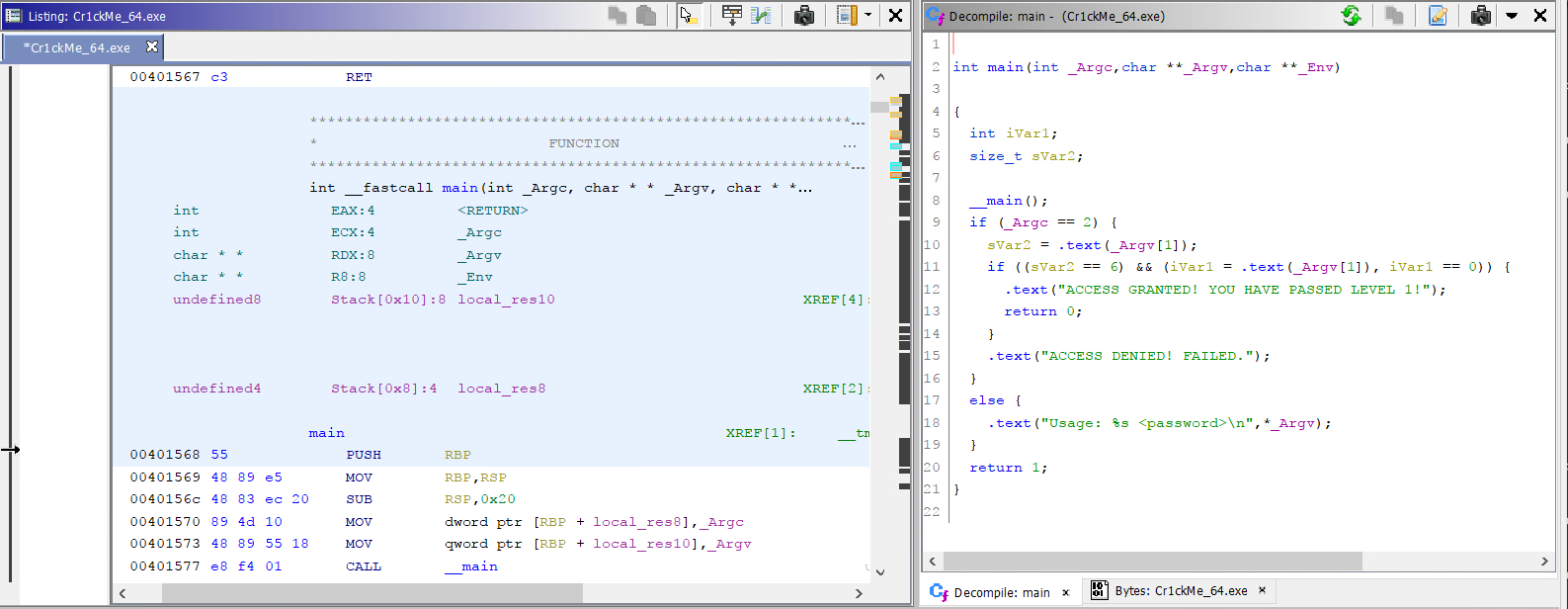
Using Ghidra

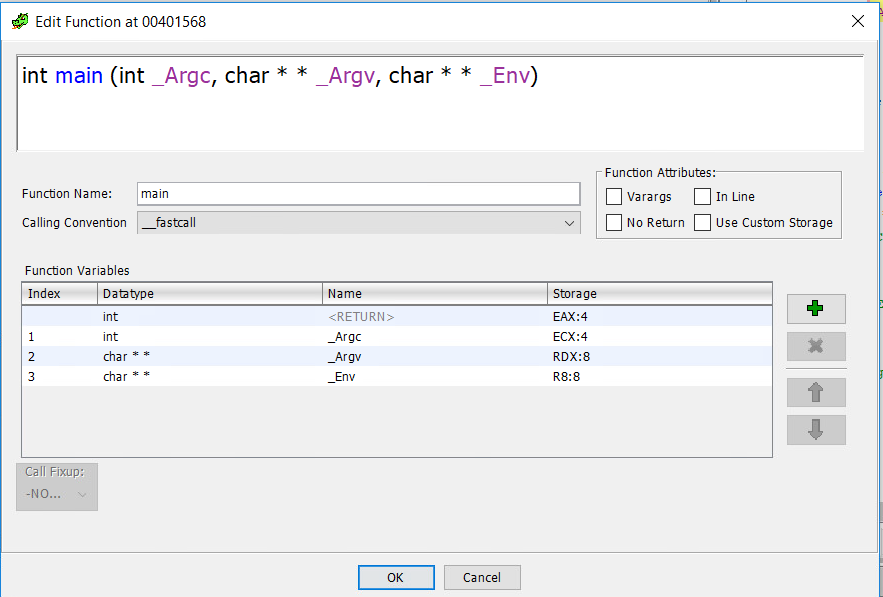




Finding the main function

Using the Symbol Tree window, we can search for the main function of the program or rather, the entry point of the program.

Working from the main function

Once the main function is found, the decompile window will show Ghidra’s decompilation of the Cr1ckMe. Looking at the first line in the decompile window:

int main(int \_Argc,char \*\*\_Argv,char \*\*\_Env)

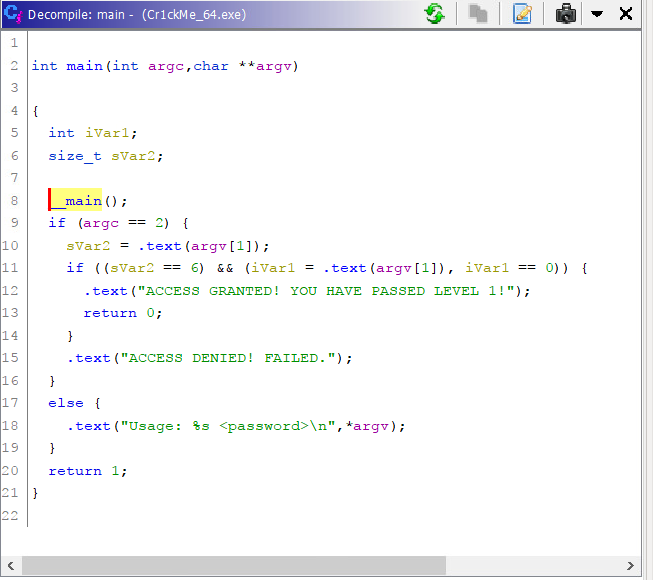
This very much resembles a typical C program, other than the \_Env variable that is not used. We can make it cleaner by editing the function’s signature.  
(done by right clicking the first line and selecting “Edit function signature” from the menu)

From here, we simply input the C standard:

After doing so, our decompilation is cleaner with the only issue being the function names and variable names. However, looking at the code we can already identify the conditional statement that allows us to pass the Cr1ckMe.

int main(int argc,char\*\* argv)

Identifying End Goal & Conditional Statement

From the code, it is easy to identify our end goal, it is to get into the part of the code that shows success.

In this code, it is in the form of a string saying “ACCESS GRANTED! YOU HAVE PASSED LEVEL 1!”.

Right before this line of code, there is an IF statement that is our conditional statement for entering this section of code.

In a nutshell, if we can make this IF statement true, we can crack the file.

Breaking the Conditional Statement

**Conditional Statement Breakdown**

if ((sVar2 == 6) && (iVar1 = .text(argv[1]), iVar1 == 0))

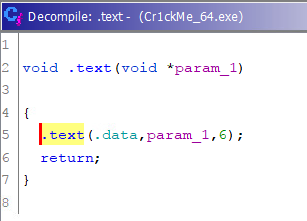
There are two factors that have to be True for this IF statement to be active:

* sVar2 == 6
* iVar1 = .text(argv[1]), iVar1 == 0

For the first factor, it tells us that the size of the password is 6 characters, since sVar2 is a size\_t type.

For the second factor, we need .text(argv[1]) to be equal to 0. To do that, we need to know what .text does to argv[1] to get its value. So, we can double click on the .text function in the decompiler.

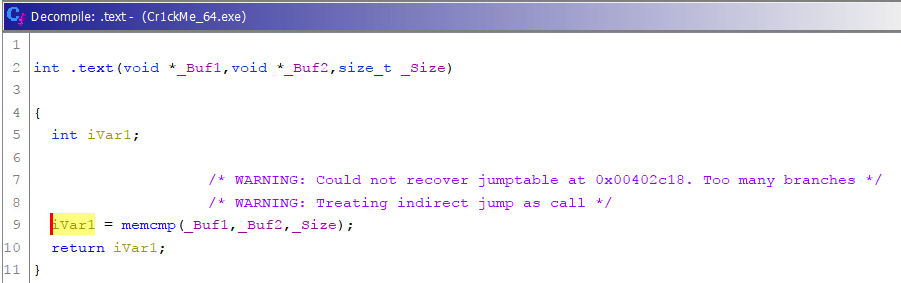
**.text #1 Function Breakdown**



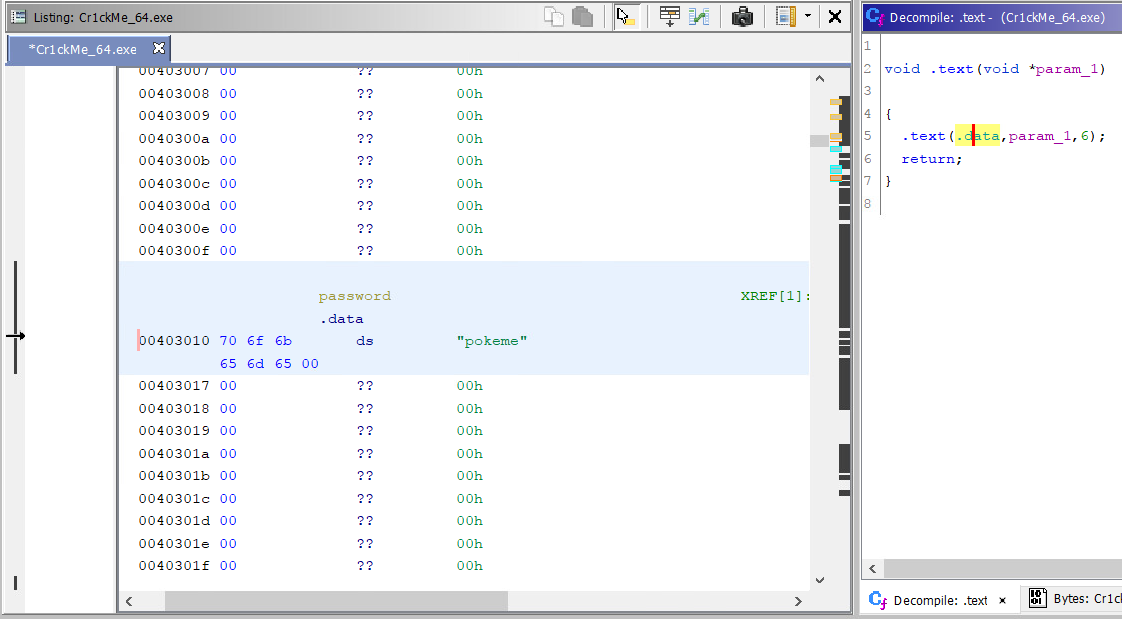
It seems to be comparing the value of “.data” to argv[1] which is input as param\_1. We can also see the value 6 being put in the next .text function.

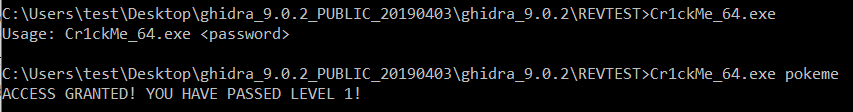
Although this is most likely a comparing function, we can double check by looking into the .text function once again.

**.text #2 Function Breakdown**

Looking at this function, it is already identified as a memcmp function so we were right about the comparison function. Now we can go back to the previous .text function and look at the data being compared against our input (argv[1]), which is most likely the password for Cr1ckMe.

**Password Found**

As shown above, the password (under the variable named password) is “pokeme”. Since it is hard-coded with no encryption or obfuscation, we can see it in clear ASCII.

**Solved**

**Notes**

A shortcut to this would be searching the entire executable for common password variable names such as “password”, “pword”, “pass”, etc. however this is not a good practice in coding and is rarely used in proper code.