Crackme#01

Filovirid
filovirid@protonmail.com

November 8, 2019

1 Question

You can find the file related to the question at crackme_01.zip.
Answer the following questions:

1. Find the flag.
2. Patch the binary.
3. There is a bug in printing the flag. What is it and patch it!!!

2 Objectives

The learning objective of this *crackme* is to learn how to work with IDA and x64dbg. To solve this problem just learn how to open a file in a debugger and read assembly codes.

3 Answer

The first step to solve this crackme is to download IDA or x64dbg. After downloading and installing the debugger of your choice, now it's time to open the file in one of the mentioned debugger. I am going to use both debuggers for this challenge. I use IDA to answer the first and third question and x64dbg to answer the second one.

3.1 Finding the flag

After openning the .exe file in IDA, you will see the default IDA window like Figure 1. If yours is different, you can set it to default by using menu '*windows->reset desktop*'. In this way, IDA layout will be reset to default.

In the left panel, you can see the '*functions*' window which has the list of functions used in the code. Some of them are import functions (what is import function??) and some of them are export functions (more about export functions). To show this panel in IDA, you can use the shortcut key '*shift+F3*'. In short, import functions are the functions which imported to the binary files from other resources like .DLL files. Export functions are the functions which are written by programmer and shared with other modules.

The function we are interested here is the '*main*' function which is the entry point of the code. The entry point of the module is different from the entry point of the code. The entry point of the module also known as 'EP', points to the first byte of the code which is executed by operating systems, but the entry point of the code is the main function and the start point of the code written by the programmer. For example, consider the following code:

```
#include <stdio.h>

int main(int argc, char ** argv){
    printf("Hi there!!");
    return 0;
}
```

The entry point of the above code is the '**main**' function but if you compile the code, you will see that the entry point of the module is another function. The reason is because of the compiler generated codes. Compilers typically add some functions at the beginning of the code to take care of stack operations, preparing all the arguments that '**main**' function expects (like argc, argv and envp). IDA has this advantages that can identify (not all the time though) the '*main*' function for you (like this sample). We will discuss more in details about compiler generated codes in further crackmes.

So after finding the '*main*' function, you will see something like Figure 2. Looking at the assembly code of the '*main*' function, you see a comparison at address 00401642 and then a jump to address 0040164A. The jump will always happen which means that it will never call the **__Z4flagv** function

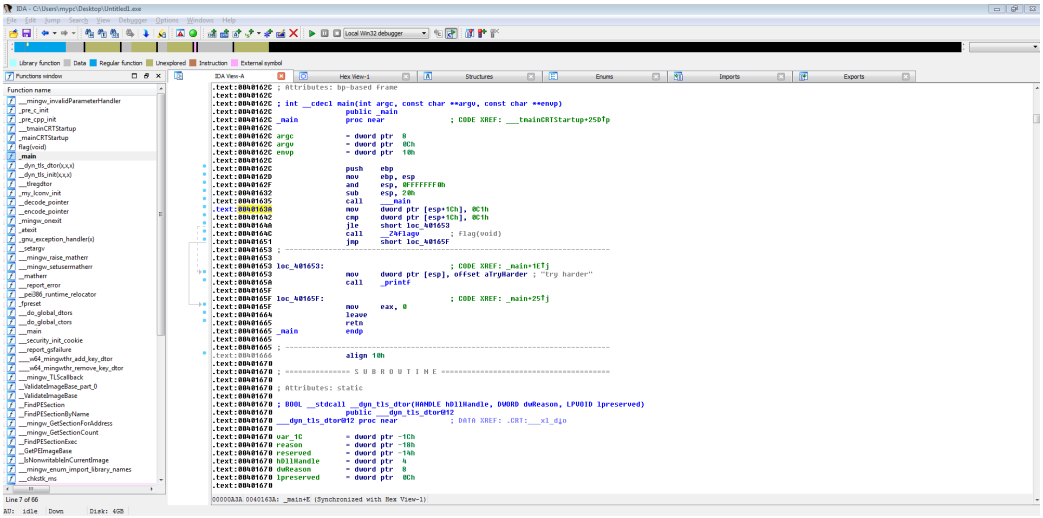

Figure 1: Default IDA window

to print the answer. In line 0040163A, *mov* instruction, put a value (0xC1) in the stack. Then, in the next line, *cmp* instruction compares the stack with the same value and based on comparison, the zero flag (ZF) changes to 1 (ZF =1). In the next line, *JLE* instruction will jump if zero flag is set (which is the case). In order to see the flag:

1. Put a breakpoint on line 0040164A (by clicking on the line and pressing F2 button).

2. Press F9 to run the program until it reaches the breakpoint.

3. On the *general registers* window (on the right side of the IDA), click on ZF and press *space* to change its value from 1 to 0.

4. Now press F8 again and you will see the flag on the screen.

3.2 Patching the binary

In this section we are going to see how we can patch this binary so that always see the flag instead of '*try harder*' string. Open the file in x32dbg (since the binary is 32-bit). Press CTRL+G and type 00401642 and press 'enter' to go to the *cmp* instruction as before. Now press space (or double click) on the instruction and change the line:

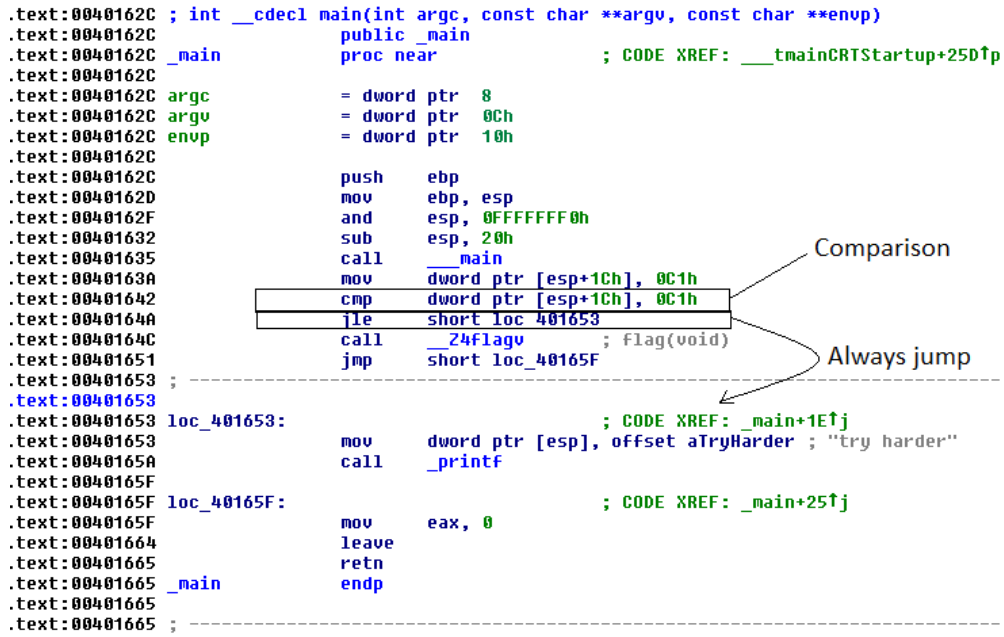

Figure 2: Default IDA window

`cmp dword ptr ss:[esp+0x1c], C1`

to

`cmp dword ptr ss:[esp+0x1c], C0`

Why we changed the value to c0? Why not c2? To answer this question, read the documentation of the *cmp* and *jle* instructions. Now from the '*file*' menu, choose '*patch file*' option and save the result by clicking '*patch file*' button.

That's it. We just patched the file!!!

3.3 Bug in printing the flag

When you see the flag on the screen, you also see some junk characters on the screen. What are these characters if they are not part of the answer? Looking at the function that prints the flag on the screen, you see that the flag string does not end with \0 (null) character. The responsible function for printing the flag is printf which prints all the characters until it reaches \0 (null) character. To fix the problem, Go to the function that is responsible

to print the flag and change the last character (at 00401614) from \A (new line character) to \0 (null character). Patch the binary in the same way you did before.