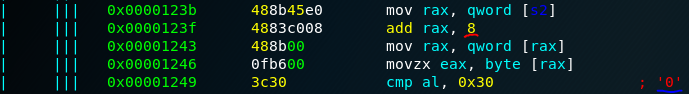
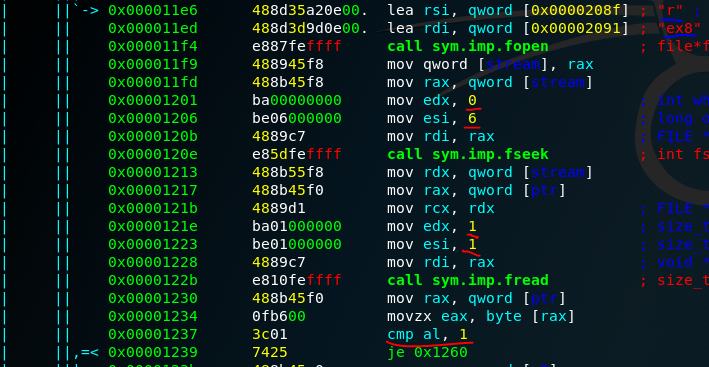
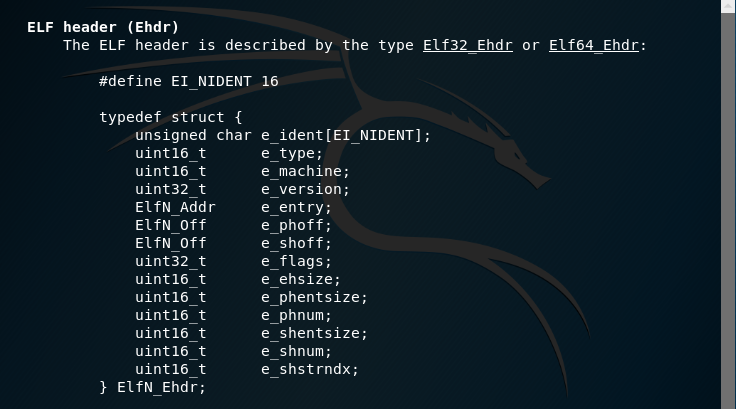
***Ex8***

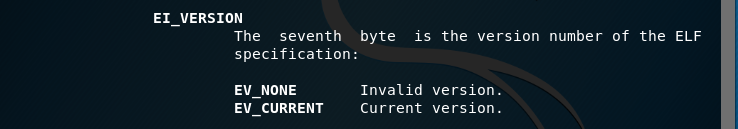
This challenge’s input is simple and easy figure out considering what we have had to do so far but there is a catch. Even with the correct input the program will not give you a score.

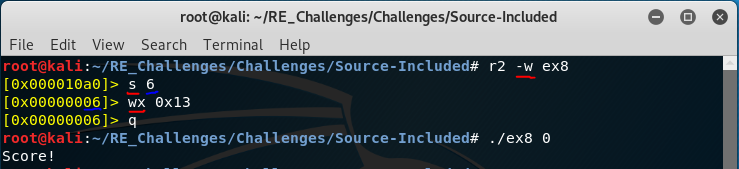
To begin lets just focus on this input. Notice that the pointer to our argv is moved into the local s2.

Further down we can see that the address of the pointer is moved into rax and incremented by 8. This is because the size of a pointer on our architecture is 8 bytes or 64 bits. Remember how the first argument to a program is its own name types into the terminal then all preceding parameters are stored in the argv array? Arrays are just sequential chunks of memory of the same type so by knowing that this array is an array of char\*’s or an array of character arrays we must move 8 bytes from the beginning of this array to get the 2nd array of characters. From here we load the address that holds the 2nd arrays 1st element then move the first byte, character, into eax with zero extension, bits not used are set to 0. Then compare it to 0x30 or ascii 0. Well looks like the input check is just checking if the first character of our input string is 0 and doesn’t care what the rest of the string is if more characters even exist.

If you tried to run a string into the program with this knowledge of how it is checking you will be surprised to see it failed! That is because there is another check done, it checks the elf version number in the e\_ident array of the program’s elf header! With all files there is a special structure to them that gives the 1’s and 0’s that they consist of meaning to the programs that read them. Before we get into these details lets give an overview of what the 3 “f”, file, functions are doing. fopen opens a file and returns a pointer called a file descriptor that other parts of the program can use as a reference to this file to be able to read and/or write to it. If we look at the parameters passed to it, underlined in red, we can see it is loading ex8 itself and in right mode, hence the “r”. After this we call the function fseek with its parameters being the file descriptor returned from fopen and two numbers, 0 and 6. These two number represent where to start in the file and how many bytes to “seek” or move forward into the file. So we can say that we are moving to the 6th byte or offset 6 into the program file on disk, note this is not program running in ram but the file itself saved on your hard drive. Now we call the fread function with a variable to store data read in and two numbers 1 and 1. These two numbers represent the size to be read from the file, 1 byte, and how many of this size to read, so just 1 byte. Finally, we compare this read value to 1 and jump accordingly.

If you run the command, “man elf”, you can read in detail about the “executable linkable format”. Notice that is is described as a struct with its first field being the “e\_ident” array of size 16. So since we are reading into just the 6th byte we can deduce the value being checked is here.

Scrolling down you will see that the 6th offset is for EI\_VERSION, which is the field that stores the version number of elf. These values represent 0 for none and 1 for the current and only version. However, this byte can take any number between 0 and 255, 0x00 and 0xff. So with the value we are reading being compared to 1 we can tell that the program is checking if the elf file is stating it is not using the current and only version of elf. Now knowing this and that it can assume a totally of 256 values from 0-255 we can rewrite this offset with whatever we want and run the program with the correct input to get a Score!

We will use yet again radare2 but this time in “write” mode, hence the red underlined “-w”. Since we are not going to need analysis performed just to write a value at a specific offset we do not need the “a” commands like before. Now we need to get to this 6th byte offset, so we seek there with “s” and state which byte into the file we need to go to. Now you should see the yellow colored prompt is the number 0x06 indicating we properly seek to the 6th offset. Now we just need to write a number into it! This is simple, we are wanting to write a hex value so we use “wx”, write hex, and any non 0x01 value then “q”, quit.