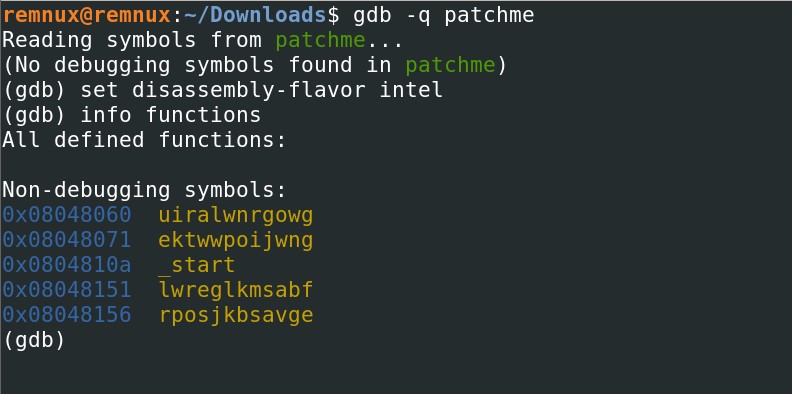
Name: Rithik Sarvesh

UID: 120395246

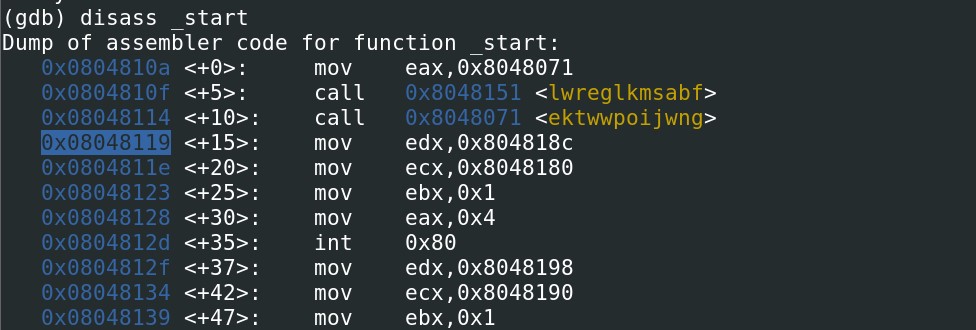
Task: In-class exercise 5

Course: ENPM696

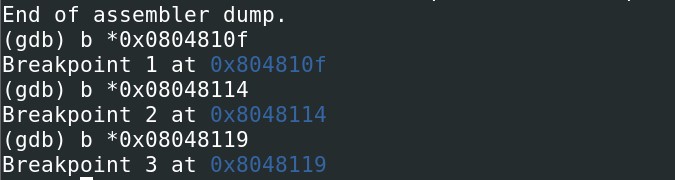
1. We saw our professor executing the binary and it printed “Remove me!” followed by “Hello” in the second line. The objective was to stop the binary from printing the “Remove me!” line. In accordance to that, let’s disassemble the binary using GDB.



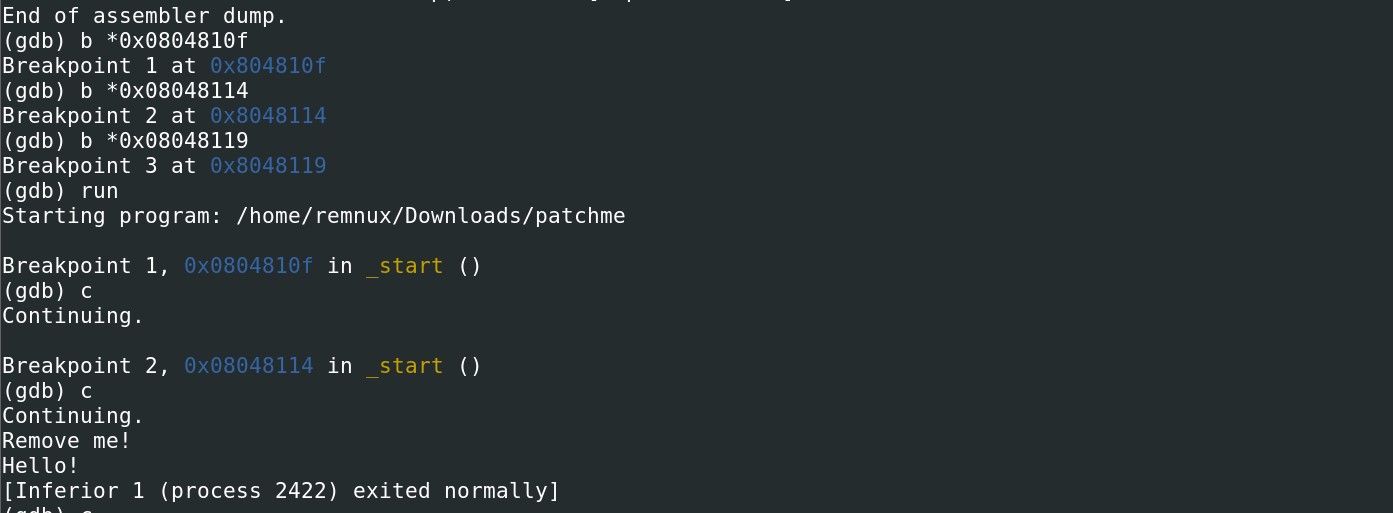
In the start function, two functions “lwreglkmsabf” and “ektwwpoijwng” were called and followed by a series of instruction. No standard c library functions were called so, it’s a handwritten binary/assembly . It was inferred as “int 0x80” command was present.



Three breakpoints were created to find when the print statement is done/called. First one is at the start of the function “lwreglkmsabf”. Second one is set at the start of the execution of the next function “ektwwpoijwng”. Third one is set at the end of the latter function.

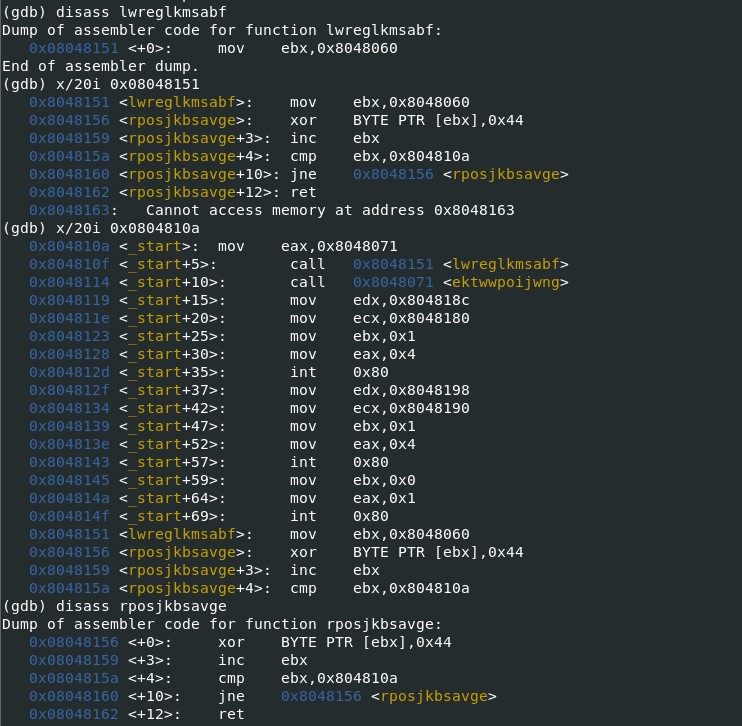


After execution, we can see that, the program exits after printing the two statements before reaching the breakpoint 3.

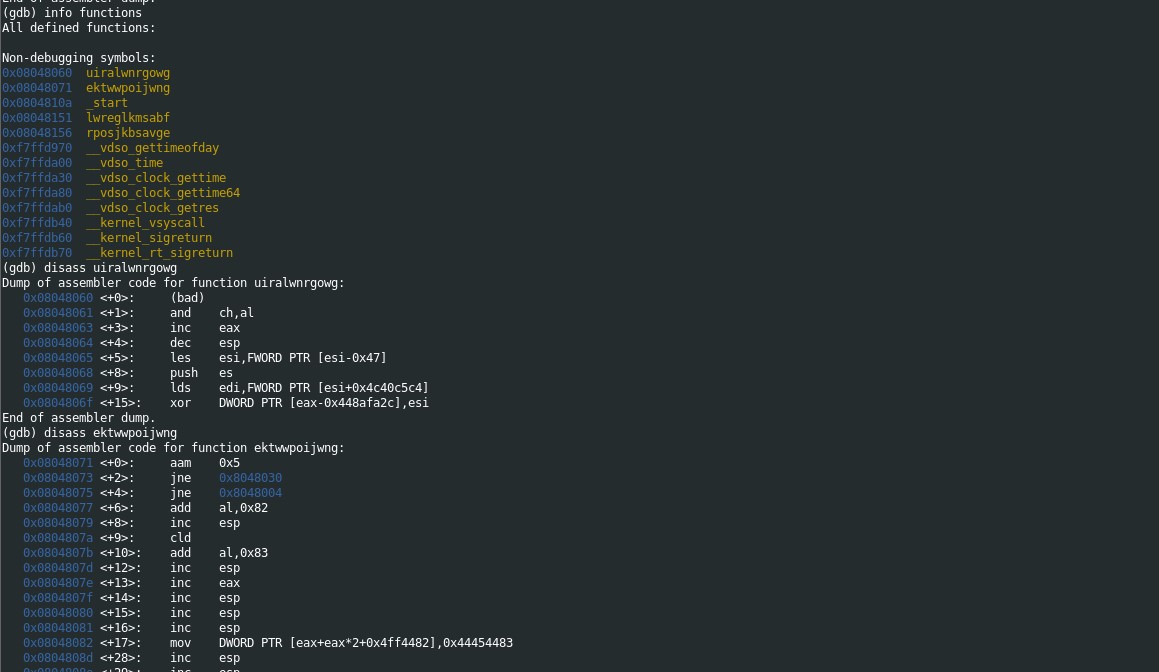


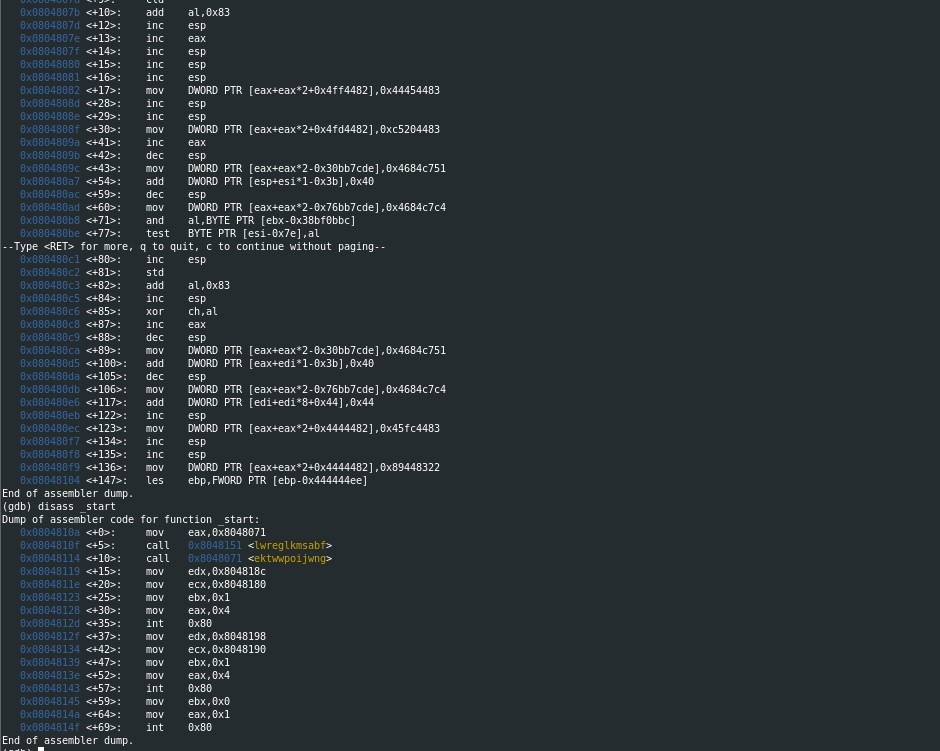
While disassembling the function “lwreglkmsabf”, we can see that there is no RET statement. Thus “x/20i 0x08048151” is done to print the next 20 instructions after the given address, As there was no RET statement in the function “lwreglkmsabf”, it keeps on executing the next instructions which is part of the function “rposjkbsavge”. At the address 0x08048151, an address (0x08048060) is fed to the EBX and it is XORed to 0x44. It runs on a loop until it reaches the address 0x0804810a.

When we used “x/20i 0x0804810a” to find the next 20 instructions, we came to know that the start function started at 0x0804810a. Thus, until that address, all the instructed were XORed already. That’s why it didn’t make sense at first. During the run-time, it was XORed again to get the intended instructions.

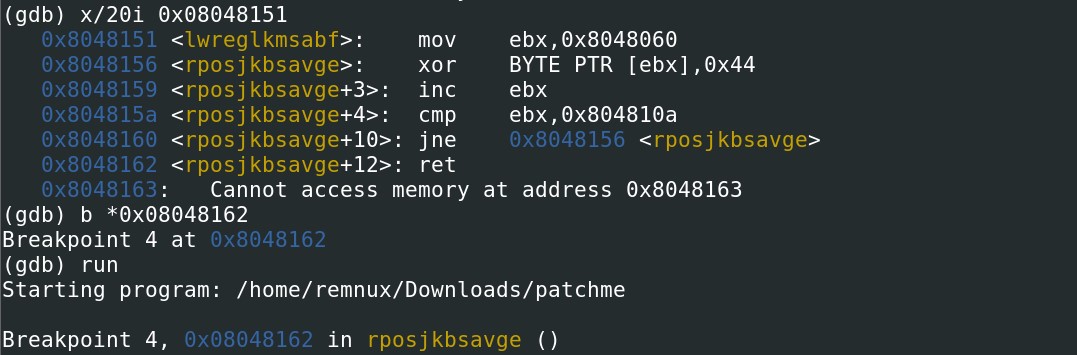


The function “uiralwnrgowg” starts at the address 0x08048060 and ends at the address 0x08048104. This is the part of the binary which is deobfuscated by the XOR operation in runtime.

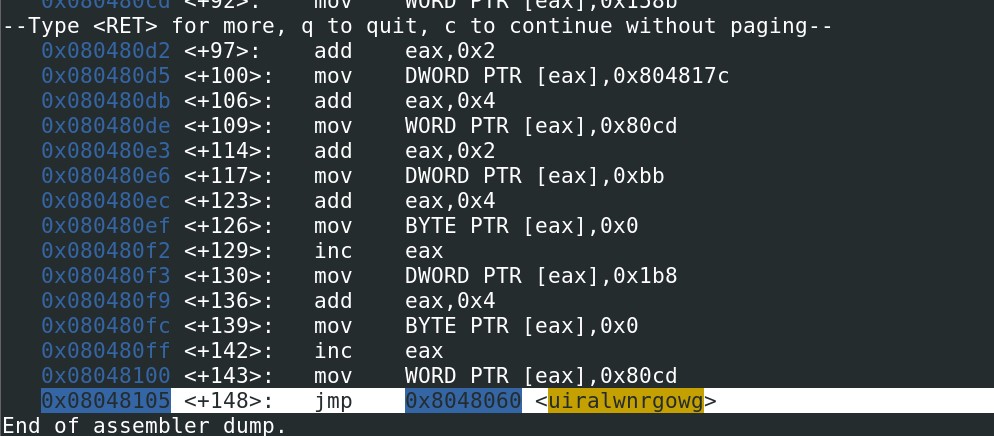




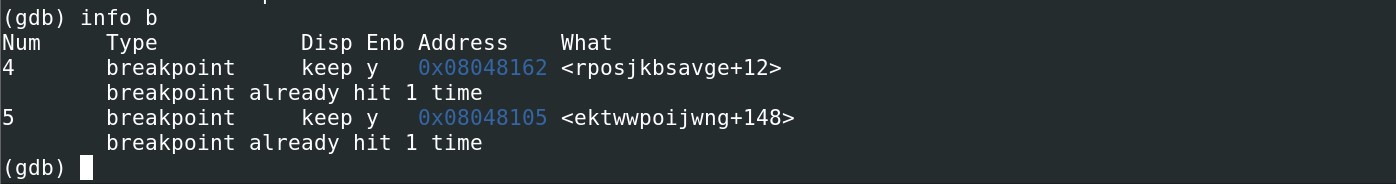
A new breakpoint is set at the return statement (0x08048162) to have a look at the deobfuscated instructions.



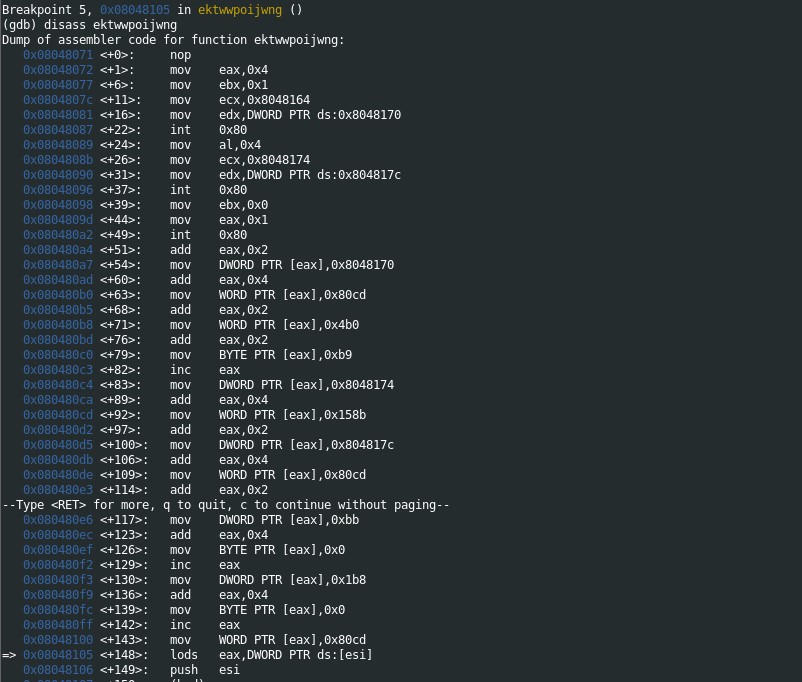
After deobfuscation, all the ‘i’ were dotted and ‘t’ were crossed. The function “uiralwnrgowg” was being called at the end of the function “ektwwpoijwng”.



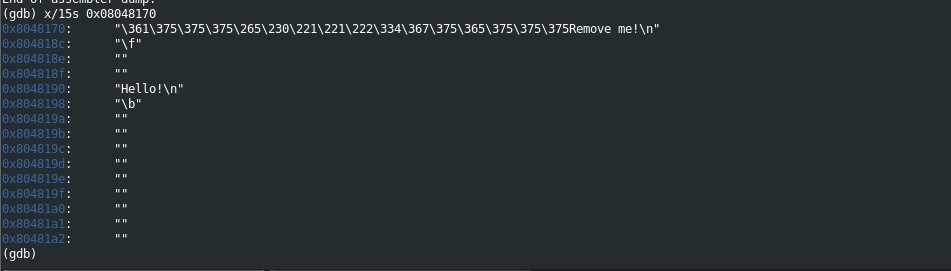
Another breakpoint is set before calling the function “uiralwnrgowg” to deobfuscate the function “ektwwpoijwng”.



Once the breakpoint was hit, the function “ektwwpoijwng” made all the senses. It was referring a series of addresses which might have the string we were looking for.



When we printed the first 15 lines of string from the address “0x08048170”. We found the string we were looking for.



B.

There were 6 “int 0x80” instructions were found. As part of the hint, 3 of those instructions present in the start function were neglected. Rest of the three were found in the function “ektwwpoijwng”.

First occurrence – 0x08048087

Arguments

EAX - 0x4

EBX – 0x1

ECX – 0x08048164

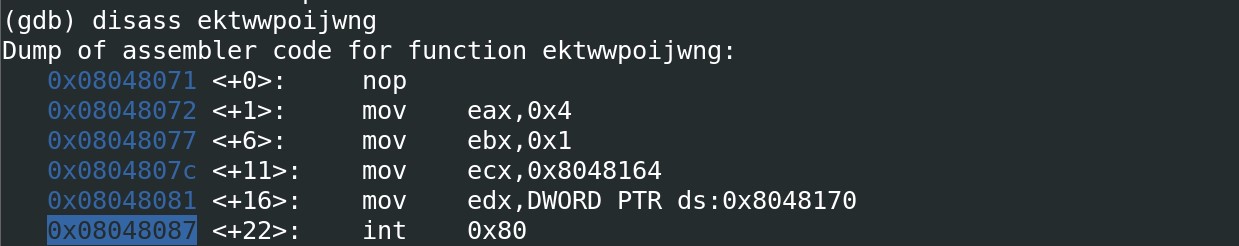
EDX – Size of the string

Possible behavior

Prints a string of size EDX from the given address at ECX.

Result

It printed “Remove me!”



Second occurrence – 0x08048096

Arguments

EAX – 0x4

EBX – 0x1 (It was already set by the first occurence)

ECX – 0x08048174

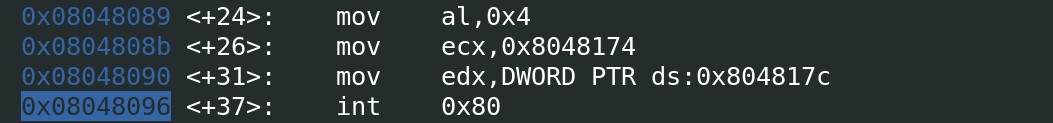
EDX – Size of the string

Possible behavior

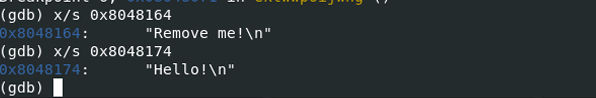
Prints a string of size EDX from the given address at ECX.

Result

It printed “Hello!”



Screenshot of the strings present in the addresses.



Third occurrence – 0x080480a2

Arguments

EAX – 0x1

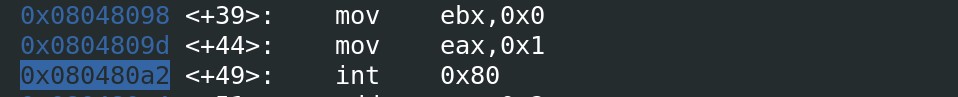
EBX – 0x0

Possible behavior

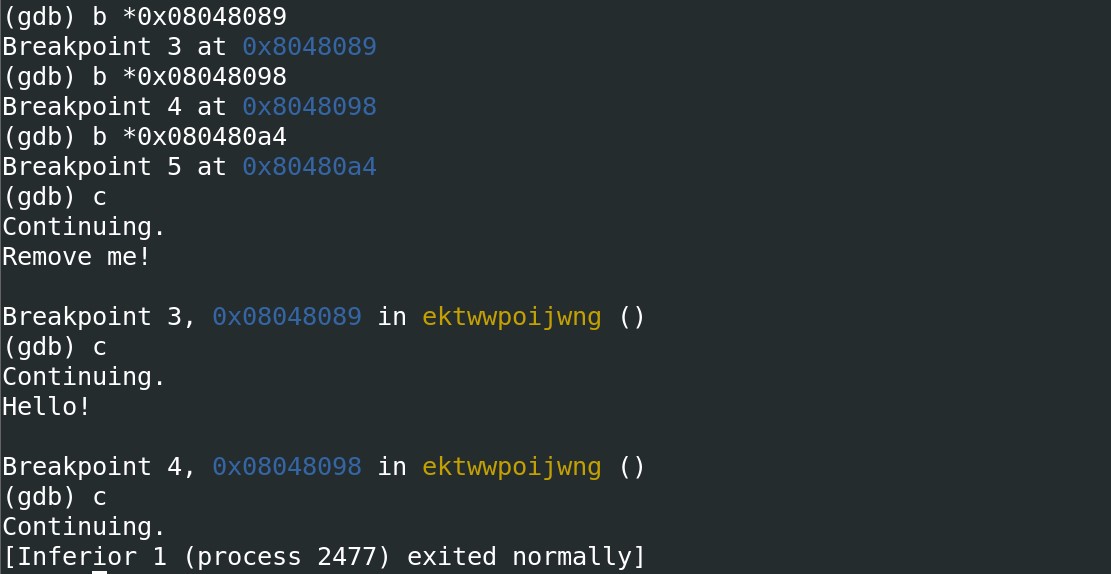
The code exits after executing this.

Result

The code exited. That’s why the rest of the start function were never executed.

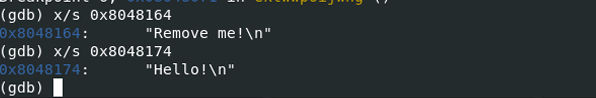


Screenshot of the result after setting breakpoints after each “int 0x80”.

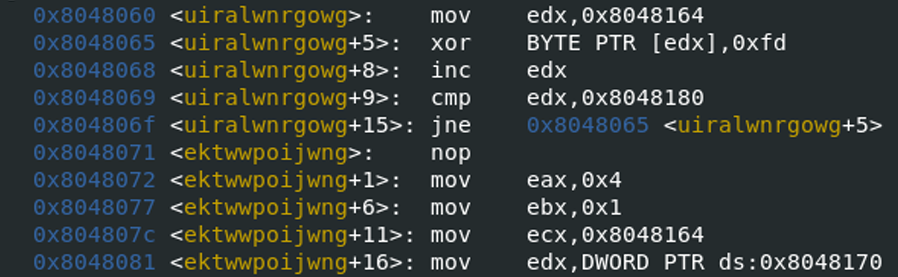


1. Erv

During our analysis, we found the location of the “Remove me!” string. The address is 0x08048164. One way to stop the code from printing the statement is by replacing this string with a value which gives 0 while being XORed.



Based on the previous analysis, we know already that the string was XORed by 0xfd before being used. Thus, 0xfd XOR oxfd returns 0 as they are same. So, by replacing the string with the hex 0xfd, we can stop the code from printing the statement “Remove me!”.



As the string starts at 0x08048164, or at the 0x164 from the start of the program, we can say that it starts at 356th byte. The size of the string(“Remove me!”) we need to replace is 10. It’s of block size 1 as it is continuous. Setting conv parameter to notrunc (to prevent file truncation.). As part of the hint, we know how to use echo to replace the string, thus the final command to make this happen is “echo -ne “\xfd\xfd\xfd\xfd\xfd\xfd\xfd\xfd\xfd\xfd” | dd of=patchme bs=1 seek=356 count=10 conv=notrunc”. When we executed the binary again, no “Remove me!” was found.

