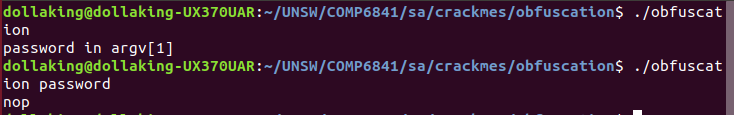
Obfuscation1 Write Up

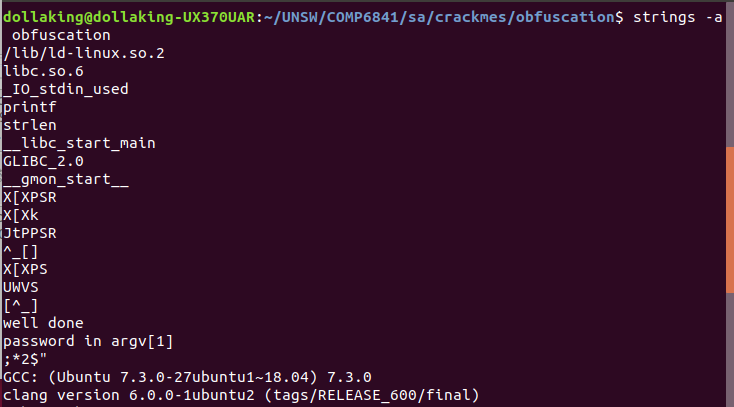
Link: <https://crackmes.one/crackme/5bd1d1bb33c5d4110a29b31e>

I ran the program by typing ./obfuscation and the output was “password in argv[1]”. I guess this means that I have to put the password as an argument with my command.

I typed in ./obfuscation password and the output of that command is nop.



First thing that popped in my head to start trying is strings. I could just use the string command “strings -a obfuscation”



There are quite a few strings, I start to go through the strings that look like a password and try them. After trying all of the strings there are none that work.

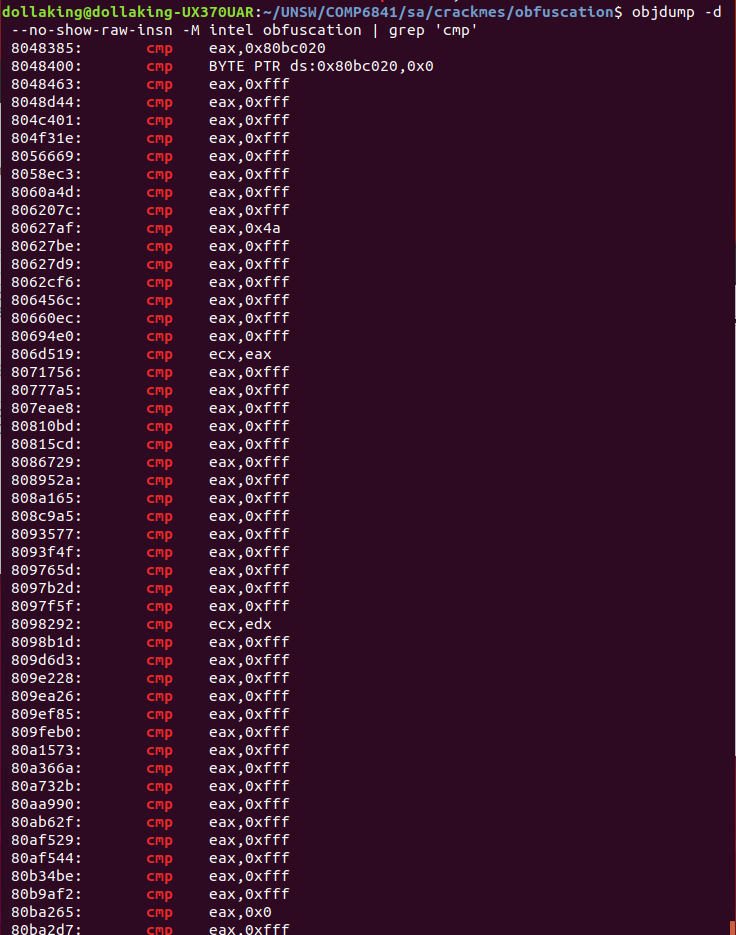
The next thing I tried is using “objdump -d –no-show-raw-insn -M intel obfuscation” and it gave me a huge list of commands where I can’t even see the beginning of the output of the commands even after scrolling my terminal to the top.

The next thing I used is a program called Binary Ninja that maps out the jumps into a graph making it very easy to analyse.

Even after using Binary Ninja, the graph is very huge and to be honest at the point I hit a wall on what I need to do to solve this.

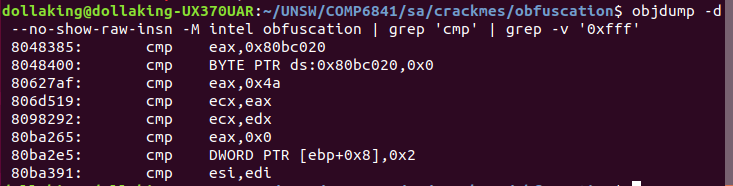
After sleeping on this problem and reading a guide; <https://idafchev.github.io/writeups/2017/06/22/basic_reverse_engineering_part_1.html>, I realised that in order to check if the password is correct it must compare. Since this is a obfuscation a lot of the functions are useless so I should filter out everything but the “cmp” commands.

To do this I must use objdump and use grep to filter everything out. “objdump -d –no-show-raw-insn -M intel obfuscation | grep ‘cmp’”



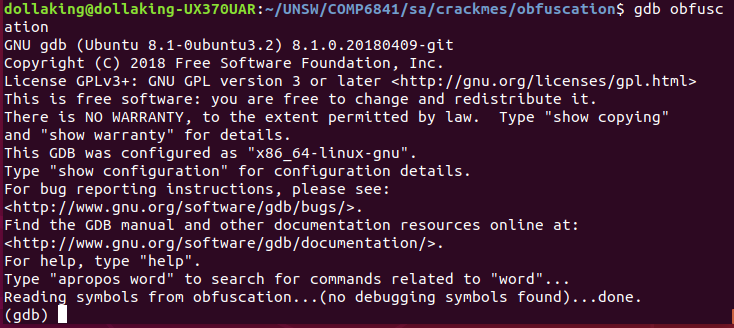
This made the list so much more compact, but there still a lot of cmp to go through. There I made a gamble and decided to filter out everything that is comparing to 0xfff since there is a lot of them. Also this is obfuscation so there will be a lot of functions that will be useless.

So I filtered out everything that contained 0xfff : “objdump -d –no-show-raw-insn -M intel obfuscation | grep ‘cmp’ | grep -v ‘0xfff’”

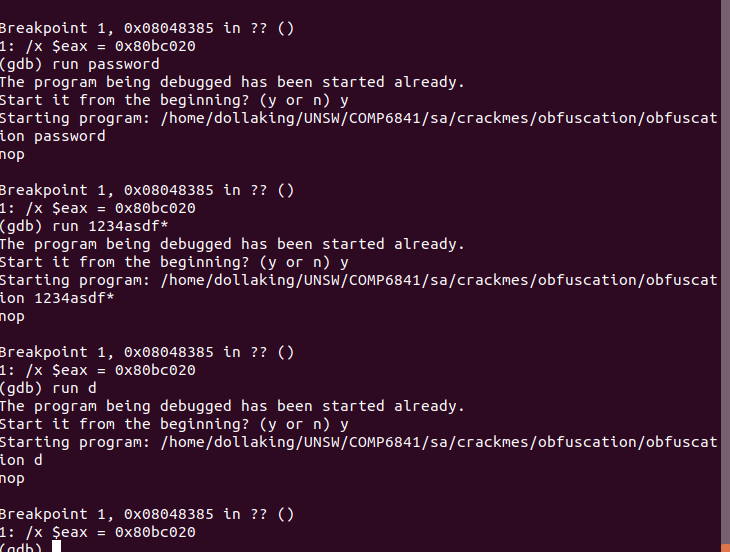


Now I am using gdb to put in different passwords and see how the cmp behave by looking at the registers.

So do achieve this first I ran gdb with the functions. I typed “gdb obfuscation”.



Then I want to put it to stop at the cmp function. At the left hand side of the objdump there will be an address, that is the address that we want to jump to. To do this, just need to type “break \*0x8048385”. Next we want to see what the register contains and we can use it to cmp ourselves what it is different when we put in different input. To do this you need to type “display/x $eax”. Display basically shows the value , the /x means that you want to see the value in hexadecimal and $eax is the register that we want to see. I chose eax because that is the register they used for that specific cmp.



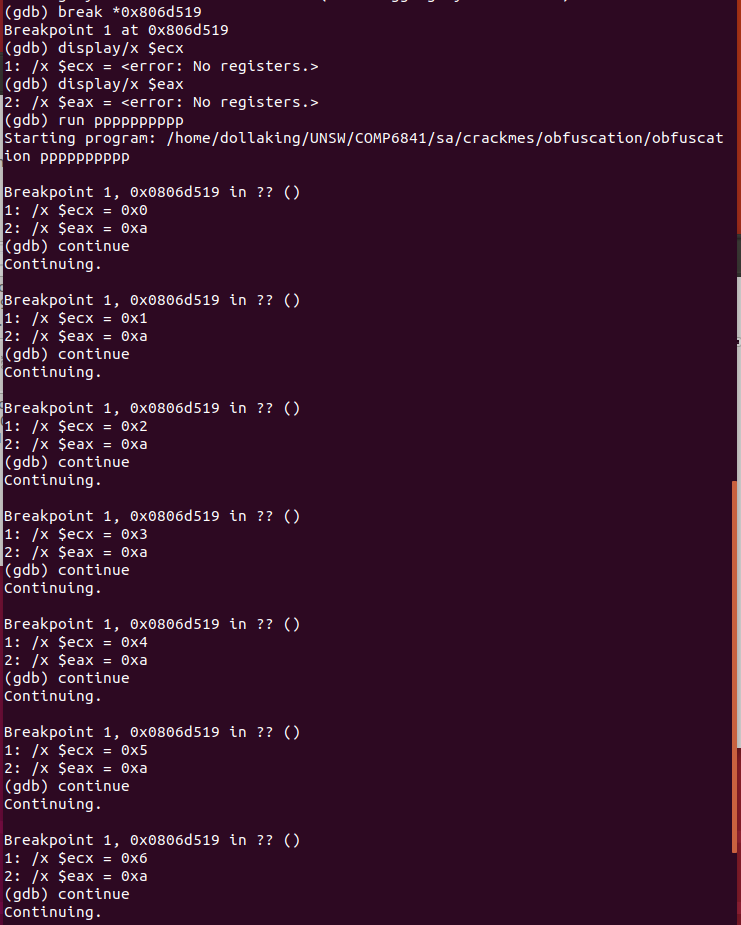
These are the passwords that I typed in “hello”, “password”, “1234asdf\*”, “d”. However no matter what password I put the register shows the exact same value : 0x80bc020 which translate to 134987808 in decimals. So it is very safe to assume that the input has no effect on this cmp so I will move on to the next cmp.

The next tried the next cmp. Everything is the same but I changed my breakpoint to 0x80627af.  
This time the numbers are different.

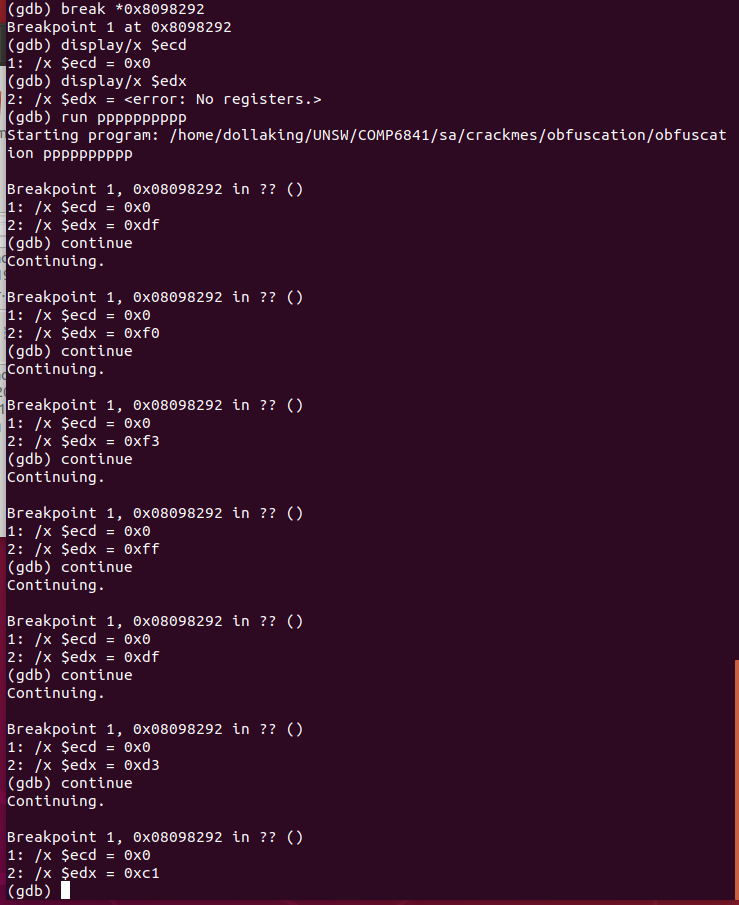
When typed “dollaking” the result was 0x43, when I typed “hello” it showed 0x27. When I typed “a” it showed oxb, when I ran “I” it showed 0xb. This is interesting because the result is the same, which could mean two things, this was a coincidence or this is a comparison of string length.

So I ran it again with “u” and it also showed 0xb. Now this confirms that this is checking the length. Now we need find out how many length will get it to 0x4a, because the cmp was trying to compare it with 0x4a. “dollaking” gave the result 0x43 which is 67, “hello” gave 0x27 which is 39 and “a” gave 11. This showed that the number increases as the length of the input increases. 0x4a is is 74 which means the length of the word is longer than “dollaking”. “dollaking” has 9 letters. So I am going to brute force this and try 10 letters so I ran “pppppppppp”. Luckily this gave 0x4a which means the password is 10 length.

Now to the next cmp, everything is the same but the breakpoint which is 806d519, and you would need to display both the ecx and eax registers since that’s what they are comparing.



I ran “run pppppppppp” in which the first break point gave me $ecx = 0x0 = 0 and $eax = 0xa = 10. I continued the program which gave me $ecx = 0x1 = 1 and $eax = 0xa = 10. I kept continuing and $ecx kept incremented at 1 while $eax remained at 10. Incrementation stops at 10. This suggests that this is a loop checking through each letter. This cmp is a counter to check if they have reached the end of the password.



I ran the next cmp which is the same process as before but the break point is \*0x8098292 and the display registers are $ecd and $edx.

I ran “run pppppppppp”

I proceeded with continue 10 times and these are the results I got.

Ecx = 0xc2, edx 0xdf

Ecx = 0xc2, edx 0xf0

Ecx = 0xc2, edx 0xf3

Ecx = 0xc2, edx 0xff

Ecx = 0xc2, edx 0xdf

Ecx = 0xc2, edx 0xd3

Ecx = 0xc2, edx 0xc1

Ecx = 0xc2, edx 0xb6

Ecx = 0xc2, edx 0xc7

Ecx 0xc2, edx 0xc0

From this there is a high chance that the characters are being compared here because I placed all the same letters as the password and every ecx is the same.

So here we know that 0xc2 is p but we don’t know the rest. So I brute forced all the alphabet, numbers and symbols, by typing the them and running this cmp.

After brute forcing I find that the password is MBA-Master.



Summary:

So basically filter out all the commands to cmp.  
Then filter out the useless cmp in this case there were a lot of cmp of 0xfff, so filter those out. For this I just filtered it out because 0xfff seems like there isn’t any significance and if the filtered cmp didn’t give me an answer I would go through the 0xfff cmp. It would be much harder if useful information was actually in one of the 0xfff cmp.  
Then go through all the cmp there and see their behaviour.  
The 3rd cmp in showed us that the password is of length 10.  
The 4th cmp showed us there is a loop checking each character.  
The next cmp compared each character but it was encoded.