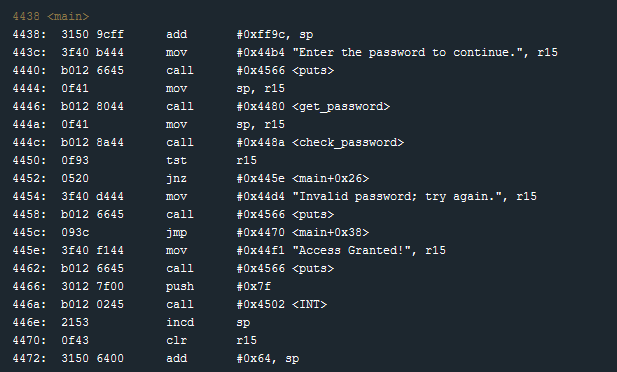
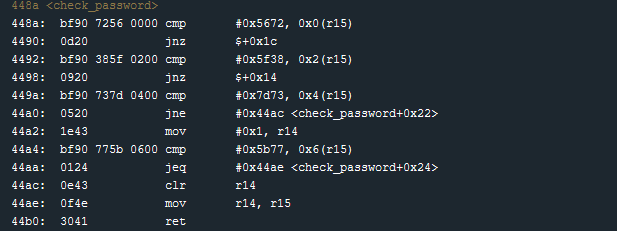
Micro Corruption - Sydney

When starting this challenge, the first thing I do again is to read the Manual. Unfortunately, near the end I see a sentence that lets me know it won’t be as simple checking memory for the password. “We have received reports that the prior version of the lock was bypassable without knowing the password. We have fixed this and removed the password from memory.” I can already tell this challenge will take a bit more thought than the last one.

Because I know the password isn’t going to be stored in memory, I decide to look through the object dump first this time.



At line 444a I see that it looks like the password entered by the user is returned from get\_password and then placed into register r15. After obtaining the password, the program checks that to confirm that it is valid. With that in mind, I head down to the check\_password function to investigate.



It looks like I was close with my understanding of what was in register r15. It doesn’t have the password stored in it (pretty obvious on second thought) but it must instead be a memory address that holds the provided password. Inside of the function, we appear to be checking what is stored at that memory address two bytes at a time. Any time the comparison fails, the function jumps to 44ac to clear the value stored in register r14. Interestingly, it seems if the program makes it to 44a2, the value in r14 is set to one. The last thing done by the function is move the value we have in r14 (either 0 or something) into r15. So we overwrite the memory address with what looks like a bit flag. This must be used by main to determine if the password was valid.

Looking back at the call to check\_password in the main function, and the code after it, it looks like the program checks if r15 is zero. If it’s not zero, a jump is made to 445e that begins the code to let the user know access has been granted. If r15 was zero, then the user is told they provided the wrong password and the program jumps near the end of main.

With all this in mind, I decide to do a hex to string conversion on the values compared against the memory address stored in r15.

0x5672 - Vr

0x5f38 - \_8

0x7d73 - }s

0x5b77 – [w

Putting this all together gives us the password “Vr\_8}s[w”.

I debug the program and provide this password and… invalid password. That’s strange. Let’s step through the code during check\_password and see what’s going wrong. I place a breakpoint at the beginning of the function so I can see where in the comparisons the password is failing.

Stepping through the comparison showed that 5672 were the first two bytes stored in the memory address, but they were failing the comparison against 0x5672. This was a little confusing to me and it took me a lot of thinking to discover why this was the case. After reviewing the manual, I learn that values on this architecture are actually stored in little endian. This means I’ll have to flip around my password to validate properly. Flipping the order of each 2byte pair gives me the password “rV8\_s}w[“. I enter that into the program and success! On to the next one.