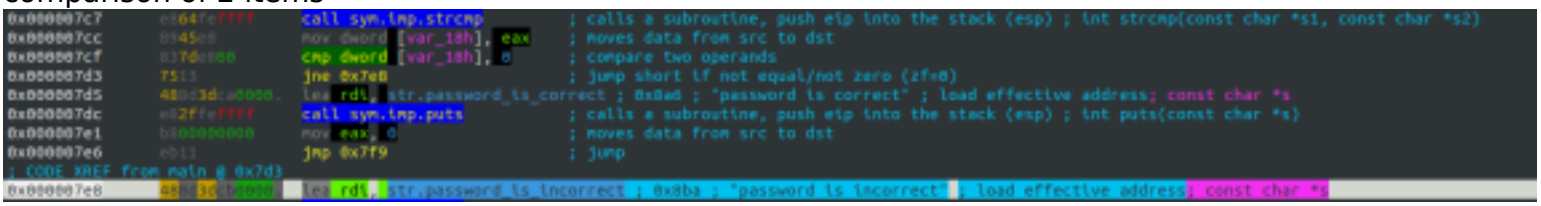
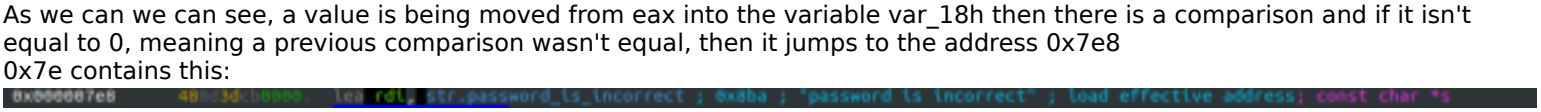
RE Challenge

File 1:

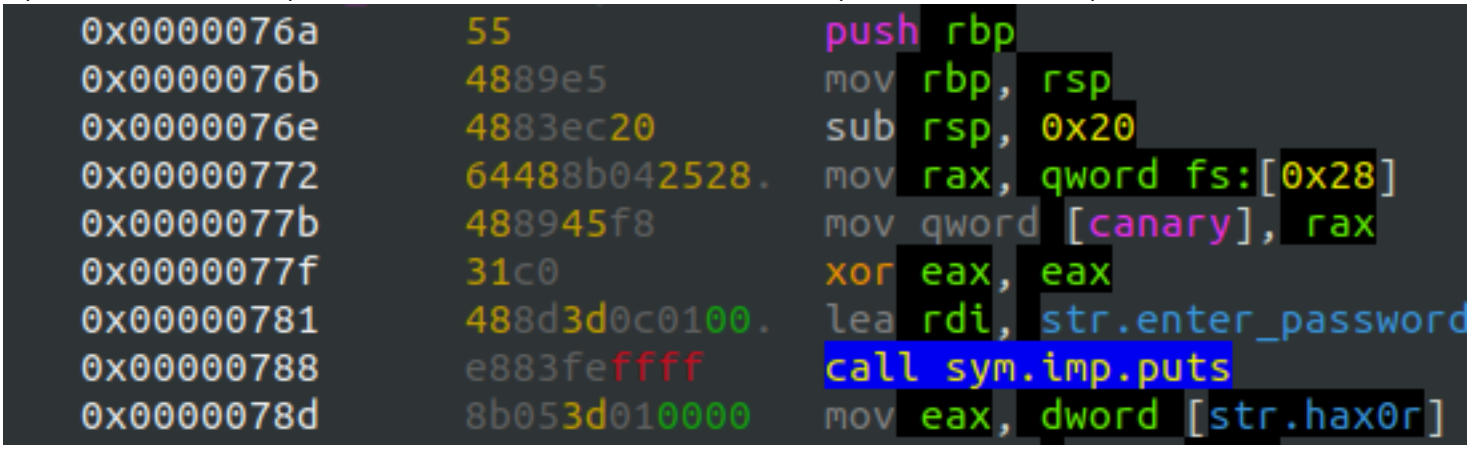
Given the fact that this is a crackme-style binary, my eyes first navigate to anything involving the comparison of 2 items



As we can we can see, a value is being moved from eax into the variable var_18h then there is a comparison and if it isn't equal to 0, meaning a previous comparison wasn't equal, then it jumps to the address 0x7e8
0x7e contains this:



So, being an Assembly noob, my mind goes to thinking that any strings mentioned before may be what was compared to our input then another comparison (between eax & 0 to make sure our input was what was expected)



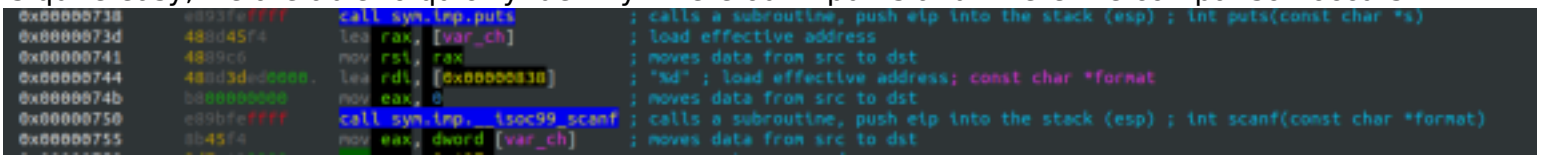
Within the first few lines of main, we have the string hax0r, so let's test this as a password:

```
[reing] ./crackme1.bin
enter password
hax0r
password is correct
```

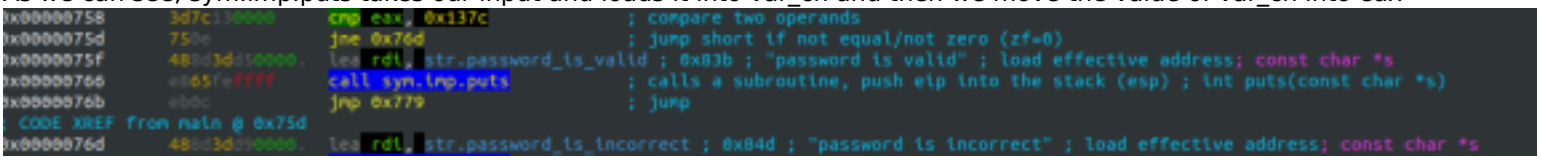
Boom! First challenge solved

File 2:

With this second file, symbols are still have the luxury of symbols like we did in the first so navigating it is quite easy, we are able to quickly identify where our input is and where the comparison occurs



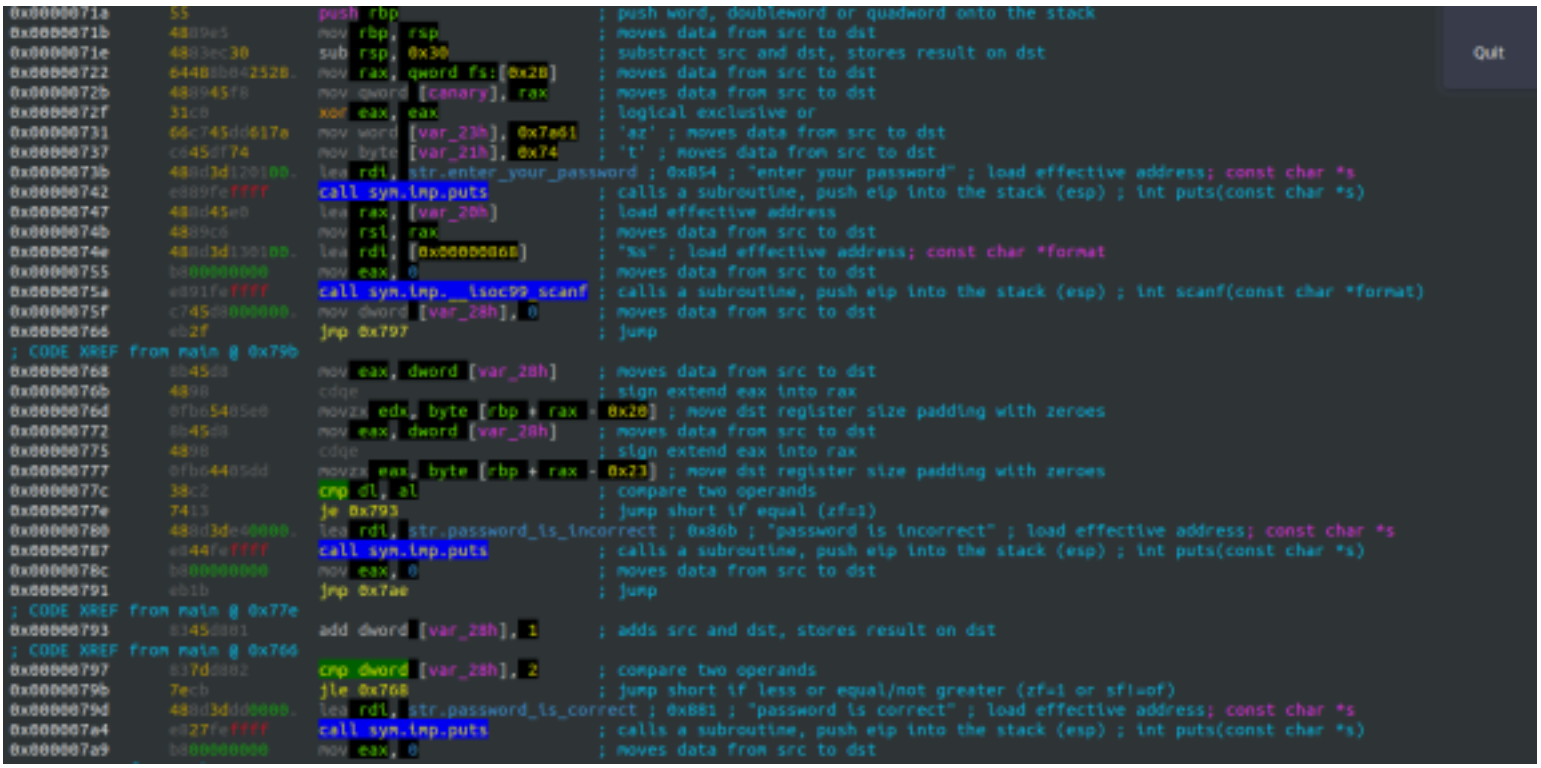
As we can see, sym.imp.puts takes our input and loads it into var_ch and then we move the value of var_ch into eax



Then our input is compared against **0x137c** which is 4988 in decimal, and if the comparison isn't equal, we get a **password is incorrect** but if the comparison is equal, we get a **password is valid**.
And as we can see, that's our password!

```
[reing] ./crackme2.bin
enter your password
4988
password is valid
```

File 3:



Now this one is more complex, in terms of the assembly.
In terms of the programming logic, it's simple
All it does (to my knowledge) is load the values az & t into variables, loop through & add them into one variable, then compare our input to that variable
Testing this theory...

```
[reing] ./crackme3.bin
enter your password
azt
password is correct
```