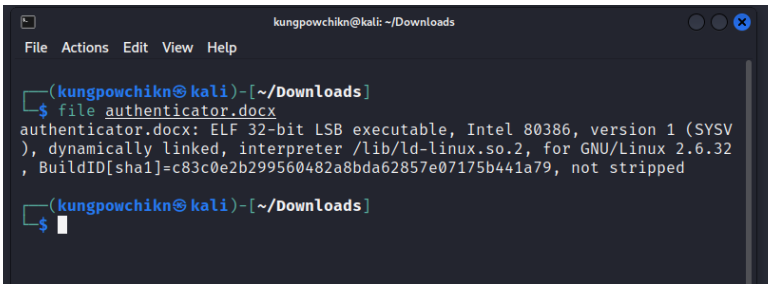
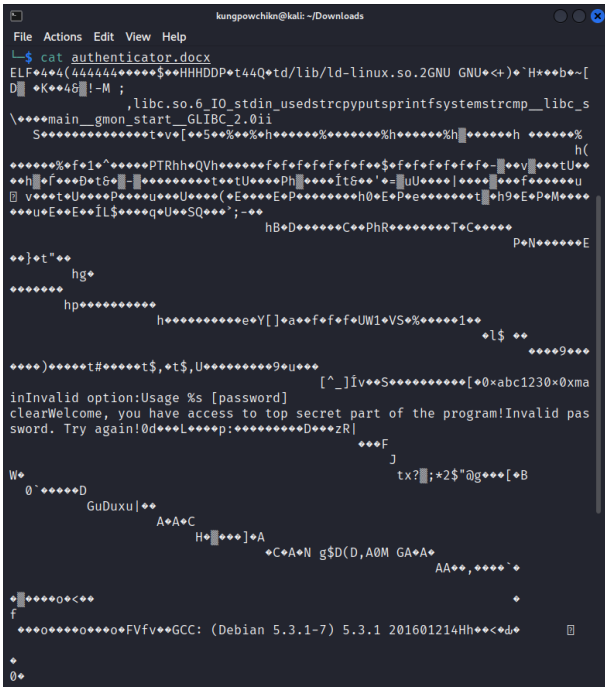
Report of the Reverse Engineering the Authenticator.docx program Assignment.

First I used *file authenticator.docx* to show information of the file:

- ELF 32 bit
- Sha hashed
- And its executable file

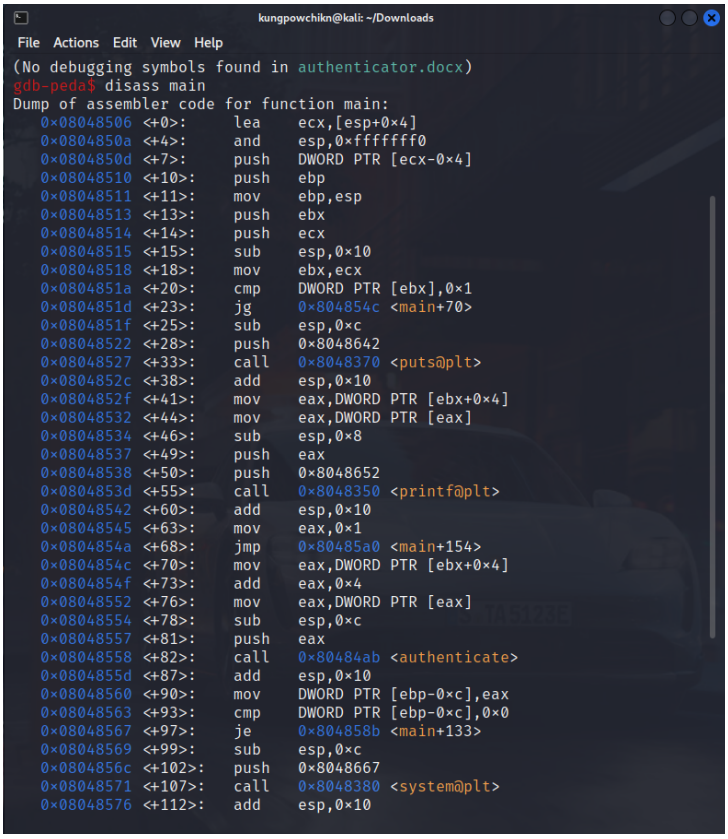


Cat'ing the file confirmed that it was in fact hashed.



To begin, I started with running GDB with peda as the hexdump took too much interpretation and I had been practicing reverse engineering/password cracking with it. So this proved to be a good opportunity to put what I learned to the test.

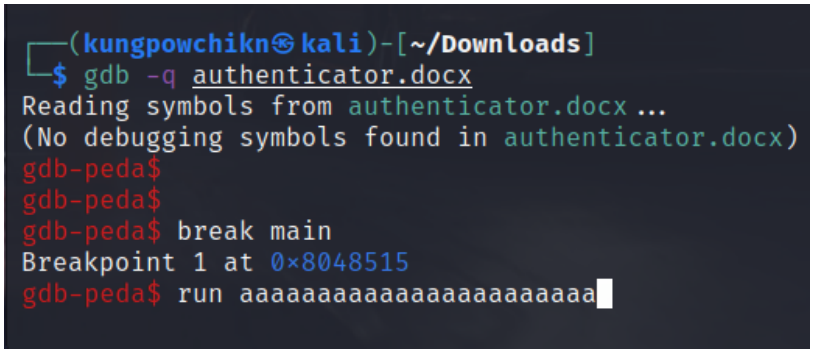
First I used the *disass main* command to see what is happening inside the code…

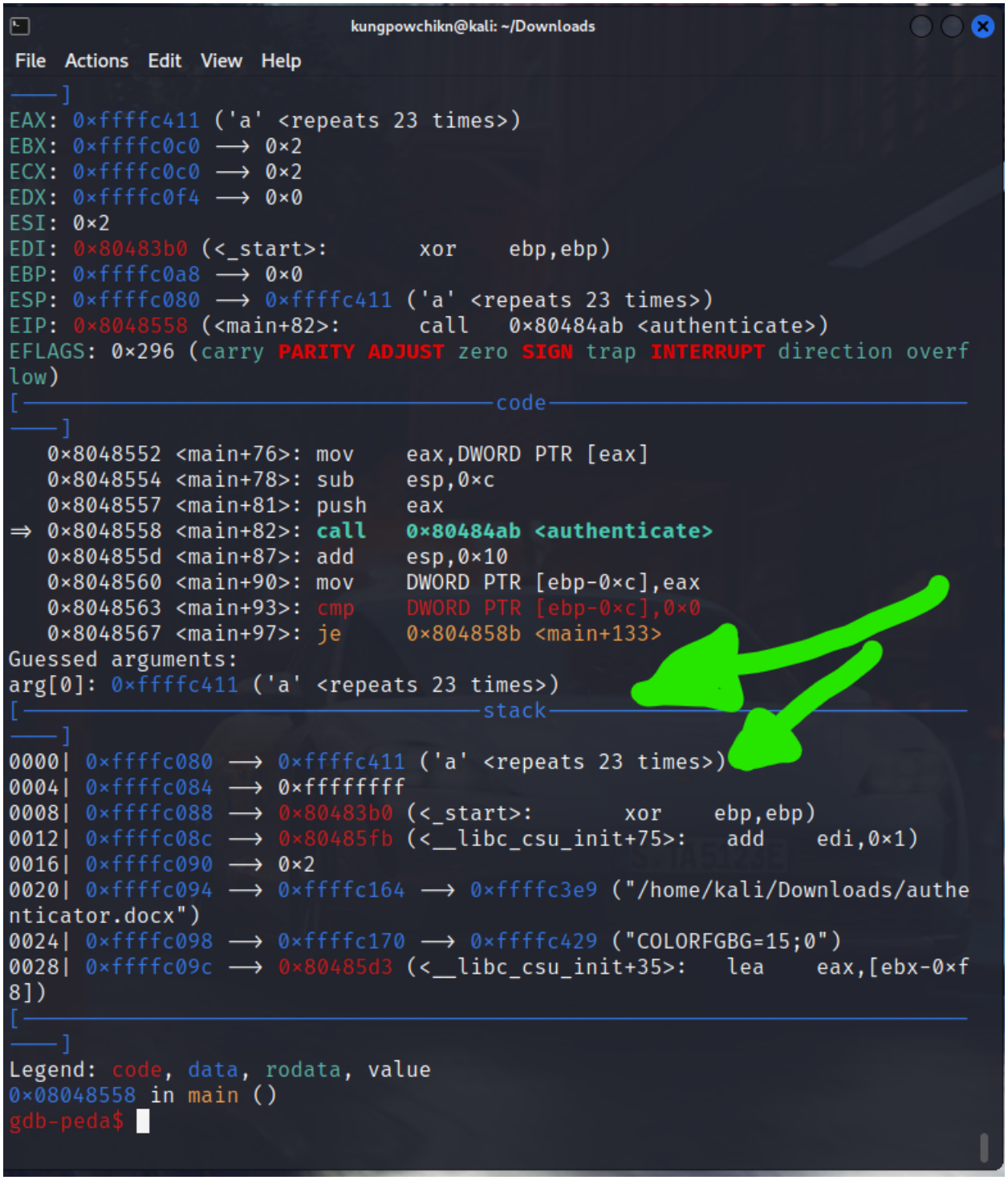


Notable things we can see:

- In main we see a printf function
    - This identifies it as a C/C++ program
- A function call to a function “authenticate”

So I run the program with a bunch of a’s and set a breakpoint at main. I’m planned to just step through the code the *next* command until I reach the call to *authenticate*



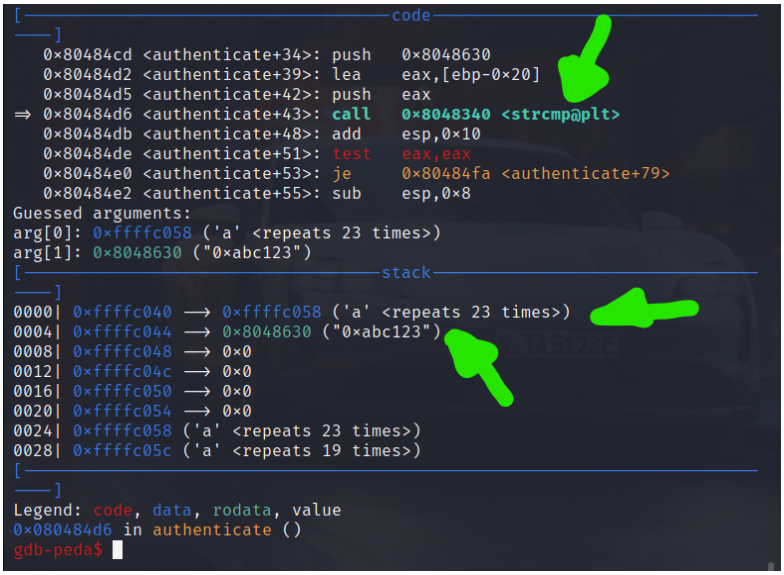


Above shows that I have reached the function call, and that my input (a string of a's) has been pushed onto the stack. Now I use the *si* command to *step* into the function.

Now I use the *disass authenticate* to show what is happening inside of the function and look for key information that makes my search easier…

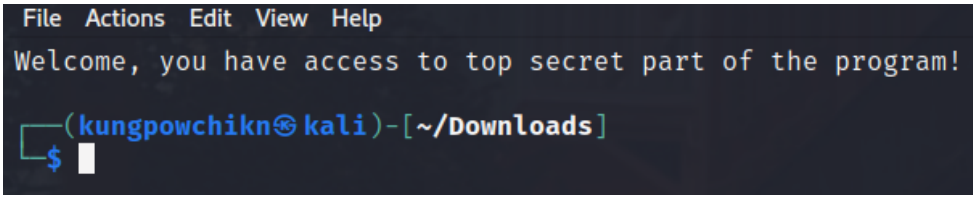
```
gdb-peda$ disass authenticate
Dump of assembler code for function authenticate:
⇒ 0x080484ab <+0>:     push   ebp
   0x080484ac <+1>:     mov    ebp,esp
   0x080484ae <+3>:     sub    esp,0x28
   0x080484b1 <+6>:     mov    DWORD PTR [ebp-0xc],0x0
   0x080484b8 <+13>:    sub    esp,0x8
   0x080484bb <+16>:    push   DWORD PTR [ebp+0x8]
   0x080484be <+19>:    lea    eax,[ebp-0x20]
   0x080484c1 <+22>:    push   eax
   0x080484c2 <+23>:    call   0x8048360 <strcpy@plt>
   0x080484c7 <+28>:    add    esp,0x10
   0x080484ca <+31>:    sub    esp,0x8
   0x080484cd <+34>:    push   0x8048630
   0x080484d2 <+39>:    lea    eax,[ebp-0x20]
   0x080484d5 <+42>:    push   eax
   0x080484d6 <+43>:    call   0x8048340 <strcmp@plt>
   0x080484db <+48>:    add    esp,0x10
   0x080484de <+51>:    test   eax,eax
   0x080484e0 <+53>:    je     0x80484fa <authenticate+79>
   0x080484e2 <+55>:    sub    esp,0x8
   0x080484e5 <+58>:    push   0x8048639
   0x080484ea <+63>:    lea    eax,[ebp-0x20]
   0x080484ed <+66>:    push   eax
   0x080484ee <+67>:    call   0x8048340 <strcmp@plt>
   0x080484f3 <+72>:    add    esp,0x10
   0x080484f6 <+75>:    test   eax,eax
   0x080484f8 <+77>:    jne    0x8048501 <authenticate+86>
   0x080484fa <+79>:    mov    DWORD PTR [ebp-0xc],0x1
   0x08048501 <+86>:    mov    eax,DWORD PTR [ebp-0xc]
   0x08048504 <+89>:    leave
   0x08048505 <+90>:    ret
End of assembler dump.
```

As you can see above there is a call to string compare. As the program is hashed, I'm hoping that the correct password is nested in the program, so again I'm going to step through the program until the *strcmp* function call



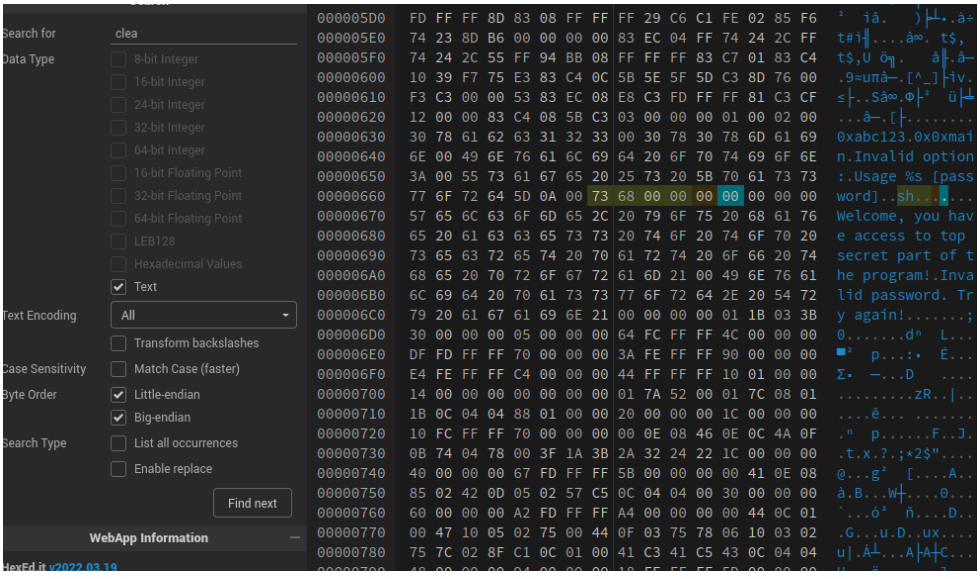
Due to how function calls work, and that strcmp only takes two parameters I can see my initial string of a's has been pushed onto the stack AND another string has been pushed onto stack, "0xabc123". I'm assuming that this is the correct passphrase.

So now I'm just checking that my assumption is correct. If not I would have stepped through carefully, and maybe even had to step into the strcmp function

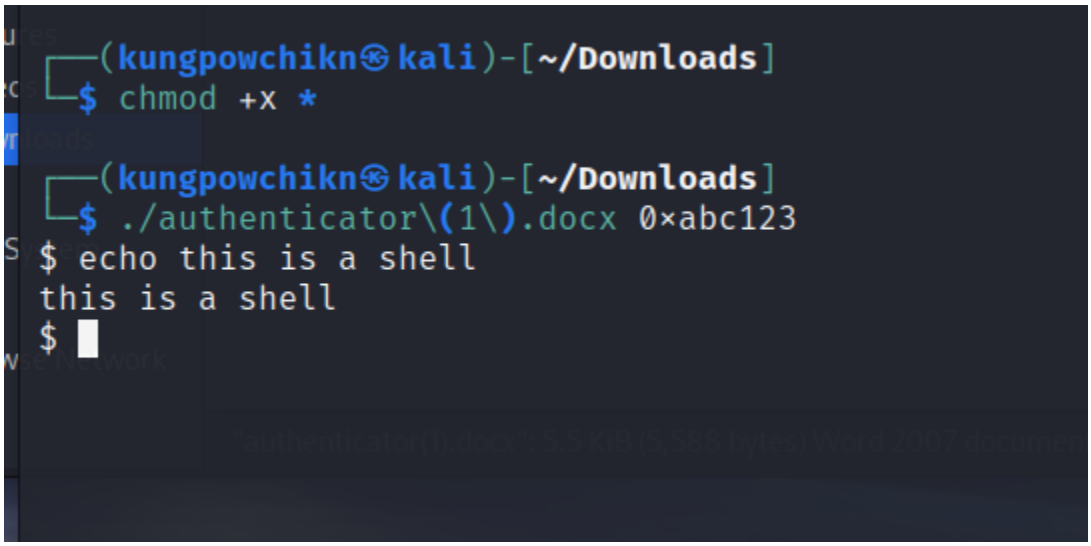


Success!

So now to open a shell upon entry to the program. I used Hexedit to see/edit the binary of the program. Upon running the program with the correct passphrase (at this point I noticed there is a second passphrase that would've worked "0x0xmain") my command line was cleared. That means there exists a clear command that executes when the password is entered. So I searched for the word clear to find an appropriate location to change the binary. Changing the corresponding hex to "73 68 00 00 0" replaces the clear command with "sh"



Upon downloading the edited program again, all one has to do is change the permissions such that the user can execute the program, and then run it with the password



Done.

I'm aware that it would've been an easier task if I had just used the hex editor for the entirety of the assignment, however I found GDB-peda pairing to be an extremely versatile tool in my toolbox, so I wanted to take the opportunity to enhance my skills with it.