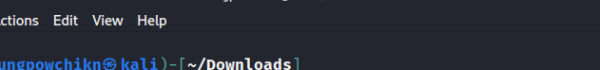


First I used *file authenticator.docx* to show information of the file:

- 
- ```
kungpowchikn@kali: ~/Downloads
File Actions Edit View Help

(kungpowchikn@kali)-[~/Downloads]
$ file authenticator.docx
authenticator.docx: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV
), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32
, BuildID[sha1]=c83c0e2b299560482a8bda62857e07175b441a79, not stripped

(kungpowchikn@kali)-[~/Downloads]
$
```

[illegible]

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

```
kungpowchikn@kali: ~/Downloads
File Actions Edit View Help
(No debugging symbols found in authenticator.docx)
gdb-peda$ disass main
Dump of assembler code for function main:
0x08048506 <+0>: lea ecx,[esp+0x4]
0x0804850a <+4>: and esp,0xffffffff
0x0804850d <+7>: push DWORD PTR [ecx-0x4]
0x08048510 <+10>: push ebp
0x08048511 <+11>: mov ebp,esp
0x08048513 <+13>: push ebx
0x08048514 <+14>: push ecx
0x08048515 <+15>: sub esp,0x10
0x08048518 <+18>: mov ebx,ecx
0x0804851a <+20>: cmp DWORD PTR [ebx],0x1
0x0804851d <+23>: jg 0x0804854c <main+70>
0x0804851f <+25>: sub esp,0xc
0x08048522 <+28>: push 0x08048642
0x08048527 <+33>: call 0x08048370 <puts@plt>
0x0804852c <+38>: add esp,0x10
0x0804852f <+41>: mov eax,DWORD PTR [ebx+0x4]
0x08048532 <+44>: mov eax,DWORD PTR [eax]
0x08048534 <+46>: sub esp,0x8
0x08048537 <+49>: push eax
0x08048538 <+50>: push 0x08048652
0x0804853d <+55>: call 0x08048350 <printf@plt>
0x08048542 <+60>: add esp,0x10
0x08048545 <+63>: mov eax,0x1
0x0804854a <+68>: jmp 0x080485a0 <main+154>
0x0804854c <+70>: mov eax,DWORD PTR [ebx+0x4]
0x0804854f <+73>: add eax,0x4
0x08048552 <+76>: mov eax,DWORD PTR [eax]
0x08048554 <+78>: sub esp,0xc
0x08048557 <+81>: push eax
0x08048558 <+82>: call 0x080484ab <authenticate>
0x0804855d <+87>: add esp,0x10
0x08048560 <+90>: mov DWORD PTR [ebp-0xc],eax
0x08048563 <+93>: cmp DWORD PTR [ebp-0xc],0x0
0x08048567 <+97>: je 0x0804858b <main+133>
0x08048569 <+99>: sub esp,0xc
0x0804856c <+102>: push 0x08048667
0x08048571 <+107>: call 0x08048380 <system@plt>
0x08048576 <+112>: add esp,0x10
```

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*

```
(kungpowchikn@kali)-[~/Downloads]
$ gdb -q authenticator.docx
Reading symbols from authenticator.docx ...
(No debugging symbols found in authenticator.docx)
gdb-peda$
gdb-peda$
gdb-peda$ break main
Breakpoint 1 at 0x08048515
gdb-peda$ run aaaaaaaaaaaaaaaaaaaaaaa
```

```
kungpowchikn@kali: ~/Downloads
File Actions Edit View Help
—]
EAX: 0xffffc411 ('a' <repeats 23 times>)
EBX: 0xffffc0c0 → 0x2
ECX: 0xffffc0c0 → 0x2
EDX: 0xffffc0f4 → 0x0
ESI: 0x2
EDI: 0x80483b0 (<_start>:      xor    ebp,ebp)
EBP: 0xffffc0a8 → 0x0
ESP: 0xffffc080 → 0xffffc411 ('a' <repeats 23 times>)
EIP: 0x8048558 (<main+82>:      call   0x80484ab <authenticate>)
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overf
low)
[—————code—————]
—]
0x8048552 <main+76>: mov     eax,DWORD PTR [eax]
0x8048554 <main+78>: sub     esp,0xc
0x8048557 <main+81>: push    eax
⇒ 0x8048558 <main+82>: call   0x80484ab <authenticate>
0x804855d <main+87>: add     esp,0x10
0x8048560 <main+90>: mov     DWORD PTR [ebp-0xc],eax
0x8048563 <main+93>: cmp     DWORD PTR [ebp-0xc],0x0
0x8048567 <main+97>: je      0x804858b <main+133>
Guessed arguments:
arg[0]: 0xffffc411 ('a' <repeats 23 times>)
[—————stack—————]
—]
0000| 0xffffc080 → 0xffffc411 ('a' <repeats 23 times>)
0004| 0xffffc084 → 0xffffffff
0008| 0xffffc088 → 0x80483b0 (<_start>:      xor    ebp,ebp)
0012| 0xffffc08c → 0x80485fb (<__libc_csu_init+75>:  add    edi,0x1)
0016| 0xffffc090 → 0x2
0020| 0xffffc094 → 0xffffc164 → 0xffffc3e9 ("/home/kali/Downloads/authen
nticator.docx")
0024| 0xffffc098 → 0xffffc170 → 0xffffc429 ("COLORFGBG=15;0")
0028| 0xffffc09c → 0x80485d3 (<__libc_csu_init+35>:  lea    eax,[ebx-0xf
8])
[—————]
—]
Legend: code, data, rodata, value
0x8048558 in main ()
gdb-peda$
```

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

```

[-----] code
0x080484cd <authenticate+34>: push    0x8048630
0x080484d2 <authenticate+39>: lea     eax,[ebp-0x20]
0x080484d5 <authenticate+42>: push    eax
=> 0x080484d6 <authenticate+43>: call    0x8048340 <strcmp@plt>
0x080484db <authenticate+48>: add     esp,0x10
0x080484de <authenticate+51>: test    eax,eax
0x080484e0 <authenticate+53>: je      0x80484fa <authenticate+79>
0x080484e2 <authenticate+55>: sub     esp,0x8
Guessed arguments:
arg[0]: 0xffffc058 ('a' <repeats 23 times>)
arg[1]: 0x8048630 ("0xabc123")
[-----] stack
0000| 0xffffc040 -> 0xffffc058 ('a' <repeats 23 times>)
0004| 0xffffc044 -> 0x8048630 ("0xabc123")
0008| 0xffffc048 -> 0x0
0012| 0xffffc04c -> 0x0
0016| 0xffffc050 -> 0x0
0020| 0xffffc054 -> 0x0
0024| 0xffffc058 ('a' <repeats 23 times>)
0028| 0xffffc05c ('a' <repeats 19 times>)
[-----]
Legend: code, data, rodata, value
0x080484d6 in authenticate ()
gdb-peda$

```

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

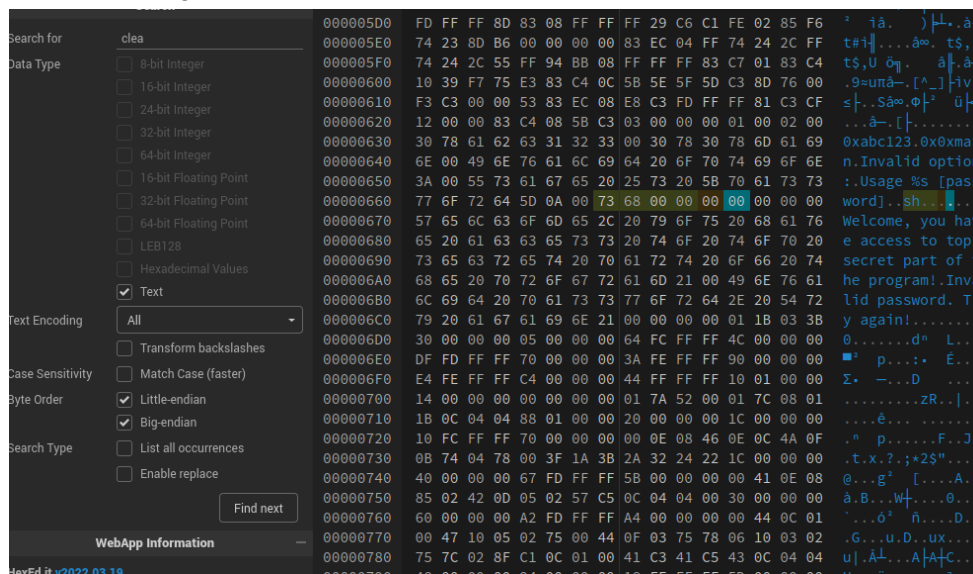
```
(kungpowchikn@kali)-[~/Downloads]
$ ./authenticator.docx 0xabc123
```

```
File Actions Edit View Help
Welcome, you have access to top secret part of the program!

(kungpowchikn@kali)-[~/Downloads]
$
```

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

```
(kungpowchikn@kali)-[~/Downloads]
$ chmod +x *

(kungpowchikn@kali)-[~/Downloads]
$ ./authenticator\1\).docx 0xabc123
$ echo this is a shell
this is a shell
$
```

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.