CSCI 420

MIA WEBER

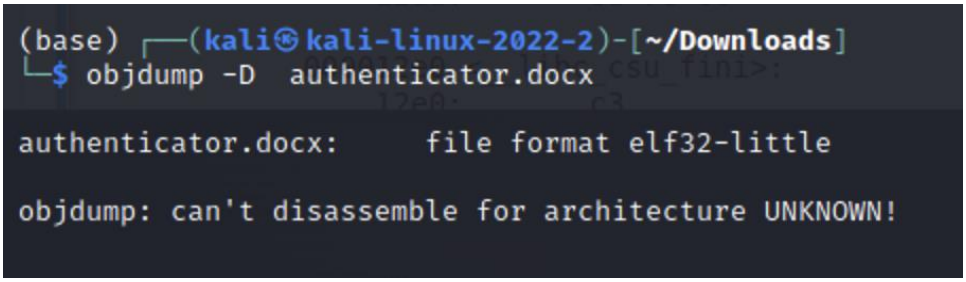
04/26/2023

A5 – Reverse Engineering

i. Determine the type/format of the file

The first thing we can do is run the ***file*** command the results of which can be seen below:

Because I am running an ARM based Kali Linux Machine (as well as my ARM based host environment) I ran into many problems related to the unknown state of the architecture that prevented the file from being disassembled and executed. This prevented me from finding the password until I was able to borrow someone else's computer in order to complete the assignment. Because of this major setback, I was unable to complete this assignment in full or on time.



https://unix.stackexchange.com/questions/119318/how-to-install-an-intel-binary-file-on-arm

ii. Reverse engineer the file to find the default passwords (backdoors) hidden

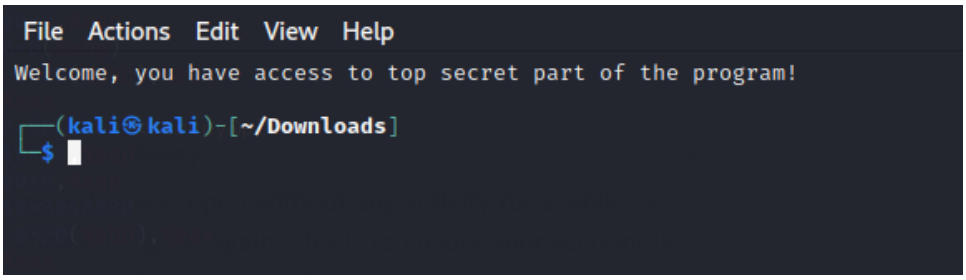
In order to find the default password in the file I used gdb to disassemble the binary and then set breakpoints and step through the assembly code lines while checking the values that are stored in each register. The following screenshots document this process:
The first breakpoint that I set was at the authenticate function using the gdb command ***break authenticate***
The second breakpoint that I set was at the instruction line 0x080484d6 which calls the string compare function. This breakpoint can be seen below. When checking the values stored in some of the relevant registers, we can see that the password that was provided (1234) is stored in the eax register.

```
Breakpoint 12, 0×080484d6 in authenticate ()
(gdb) disass authenticate
Dump of assembler code for function authenticate:
   0×080484ab <+0>:     push   %ebp
   0×080484ac <+1>:     mov    %esp,%ebp
   0×080484ae <+3>:     sub    $0×28,%esp
   0×080484b1 <+6>:     movl   $0×0,-0×c(%ebp)
   0×080484b8 <+13>:    sub    $0×8,%esp
   0×080484bb <+16>:    push   0×8(%ebp)
   0×080484be <+19>:    lea    -0×20(%ebp),%eax
   0×080484c1 <+22>:    push   %eax
   0×080484c2 <+23>:    call   0×8048360 <strcpy@plt>
   0×080484c7 <+28>:    add    $0×10,%esp
   0×080484ca <+31>:    sub    $0×8,%esp
   0×080484cd <+34>:    push   $0×8048630
   0×080484d2 <+39>:    lea    -0×20(%ebp),%eax
   0×080484d5 <+42>:    push   %eax
⇒  0×080484d6 <+43>:    call   0×8048340 <strcmp@plt>
   0×080484db <+48>:    add    $0×10,%esp
   0×080484de <+51>:    test   %eax,%eax
   0×080484e0 <+53>:    je     0×80484fa <authenticate+79>
   0×080484e2 <+55>:    sub    $0×8,%esp
   0×080484e5 <+58>:    push   $0×8048639
   0×080484ea <+63>:    lea    -0×20(%ebp),%eax
   0×080484ed <+66>:    push   %eax
   0×080484ee <+67>:    call   0×8048340 <strcmp@plt>
   0×080484f3 <+72>:    add    $0×10,%esp
   0×080484f6 <+75>:    test   %eax,%eax
   0×080484f8 <+77>:    jne    0×8048501 <authenticate+86>
   0×080484fa <+79>:    movl   $0×1,-0×c(%ebp)
   0×08048501 <+86>:    mov    -0×c(%ebp),%eax
   0×08048504 <+89>:    leave
   0×08048505 <+90>:    ret
End of assembler dump.
(gdb) x/s $eax
0×ffffcf88:     "1234"
(gdb) x/s $esp
0×ffffcf70:     "\210\317\377\3770\206\004\b@\332\377", <incomplete sequence \367>
(gdb)
```

The third breakpoint that I set was at the instruction line 0x080484ee which is the second call to the string compare function. This breakpoint can be seen below. We can see that the value of eax is still 1234 and the value that is pushed in the register 0x8048639 is “0x0xmain” which looked like a password to me.

```
Dump of assembler code for function authenticate:
   0×080484ab <+0>:     push   %ebp
   0×080484ac <+1>:     mov    %esp,%ebp
   0×080484ae <+3>:     sub    $0×28,%esp
   0×080484b1 <+6>:     movl   $0×0,-0×c(%ebp)
   0×080484b8 <+13>:    sub    $0×8,%esp
   0×080484bb <+16>:    push   0×8(%ebp)
   0×080484be <+19>:    lea    -0×20(%ebp),%eax
   0×080484c1 <+22>:    push   %eax
   0×080484c2 <+23>:    call   0×8048360 <strcpy@plt>
   0×080484c7 <+28>:    add    $0×10,%esp
   0×080484ca <+31>:    sub    $0×8,%esp
   0×080484cd <+34>:    push   $0×8048630
   0×080484d2 <+39>:    lea    -0×20(%ebp),%eax
   0×080484d5 <+42>:    push   %eax
   0×080484d6 <+43>:    call   0×8048340 <strcmp@plt>
   0×080484db <+48>:    add    $0×10,%esp
   0×080484de <+51>:    test   %eax,%eax
   0×080484e0 <+53>:    je     0×80484fa <authenticate+79>
   0×080484e2 <+55>:    sub    $0×8,%esp
   0×080484e5 <+58>:    push   $0×8048639
   0×080484ea <+63>:    lea    -0×20(%ebp),%eax
   0×080484ed <+66>:    push   %eax
⇒  0×080484ee <+67>:    call   0×8048340 <strcmp@plt>
   0×080484f3 <+72>:    add    $0×10,%esp
   0×080484f6 <+75>:    test   %eax,%eax
   0×080484f8 <+77>:    jne    0×8048501 <authenticate+86>
   0×080484fa <+79>:    movl   $0×1,-0×c(%ebp)
   0×08048501 <+86>:    mov    -0×c(%ebp),%eax
   0×08048504 <+89>:    leave
   0×08048505 <+90>:    ret
End of assembler dump.
(gdb) x/s $eax
0×ffffcf88:     "1234"
(gdb) x/s $ebp
0×ffffcfa8:     "\330\317\377\377]\205\004\b\215\322\377\377˜\375\367/\312\301\367\240\024\374", <incomplete sequence \3
67>
(gdb) x/s $esp
0×ffffcf70:     "\210\317\377\3779\206\004\b@\332\377", <incomplete sequence \367>
(gdb) x/s 0×8048340
0×8048340 <strcmp@plt>: "\377%\370\230\004\bh"
(gdb) x/s 0×80484fa
0×80484fa <authenticate+79>:    "\307E\364\001"
(gdb) x/s 0×8
0×8:    <error: Cannot access memory at address 0×8>
(gdb) x/s 0×8048639
0×8048639:      "0×0xmain"
(gdb)
```

Sure enough, when we execute the file with the password “0x0xmain” we gain access to the secret part of the program. This can be seen in the screenshot below.



iii. **Modify the binary to execute the */bin/sh* shell program when the user successfully authenticates**

This is the part of the assignment that I was unable to complete. Below is a compilation of links to resources that I found and was using.

https://stackoverflow.com/questions/67993603/how-does-a-linux-c-c-system-command-work

https://stackoverflow.com/questions/14827894/launch-shell-with-inline-assembly

We get the byte sequence of 2f62696e2f2f7368, which in ASCII is equal to /bin//sh

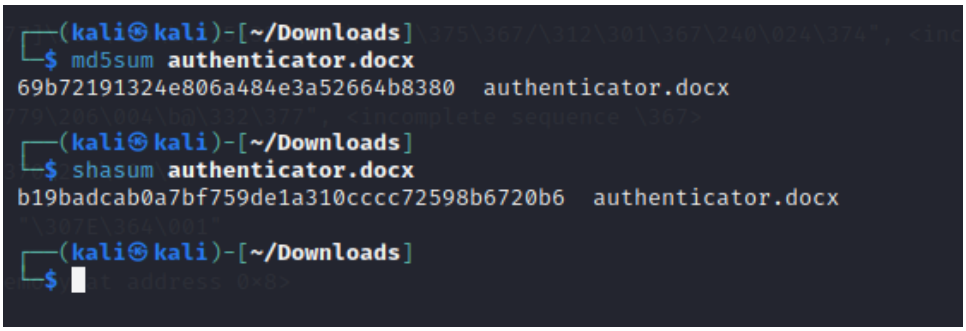
https://axcheron.github.io/linux-shellcode-101-from-hell-to-shell/

https://stackoverflow.com/questions/65766170/assembly-code-to-shell-code-section-data-and-section-text-in-which-order

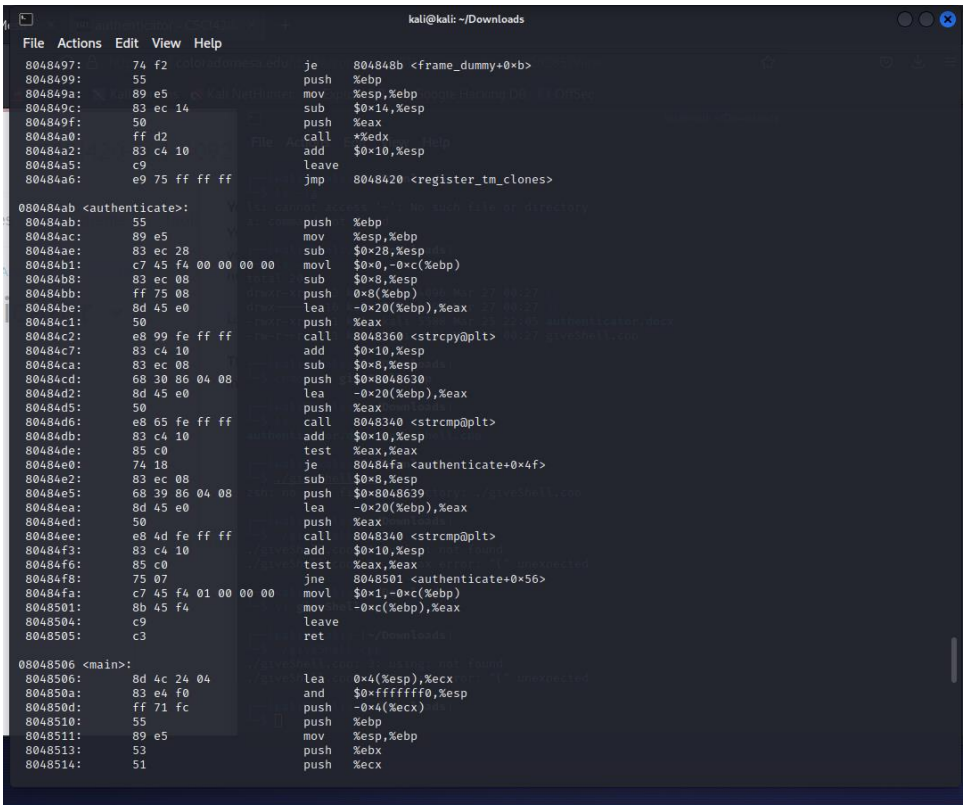
iv. **Briefly explain checksums and calculate md5 and sha1 of the original and modified binaries and put those in the report**

Checksums are generated from running a cryptographic hash function on a file for the purpose of detecting errors that may have happened during its transmission or storage. Checksums are frequently used to verify data integrity but are not relied on to verify the authenticity of the data.

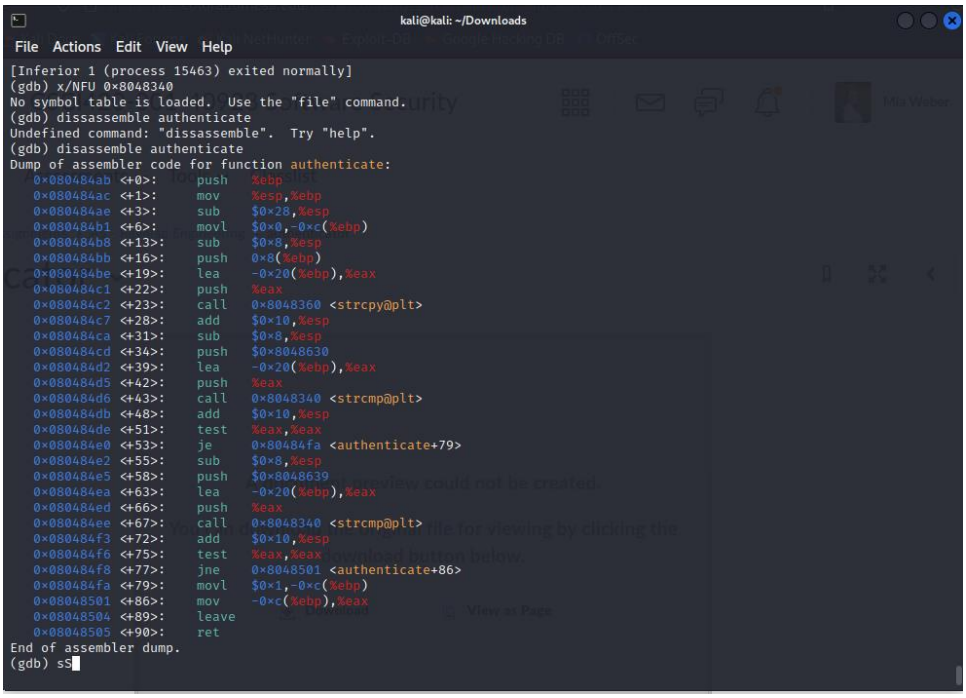
Below is a screenshot of the md5 and sha1 checksums of the original binary:



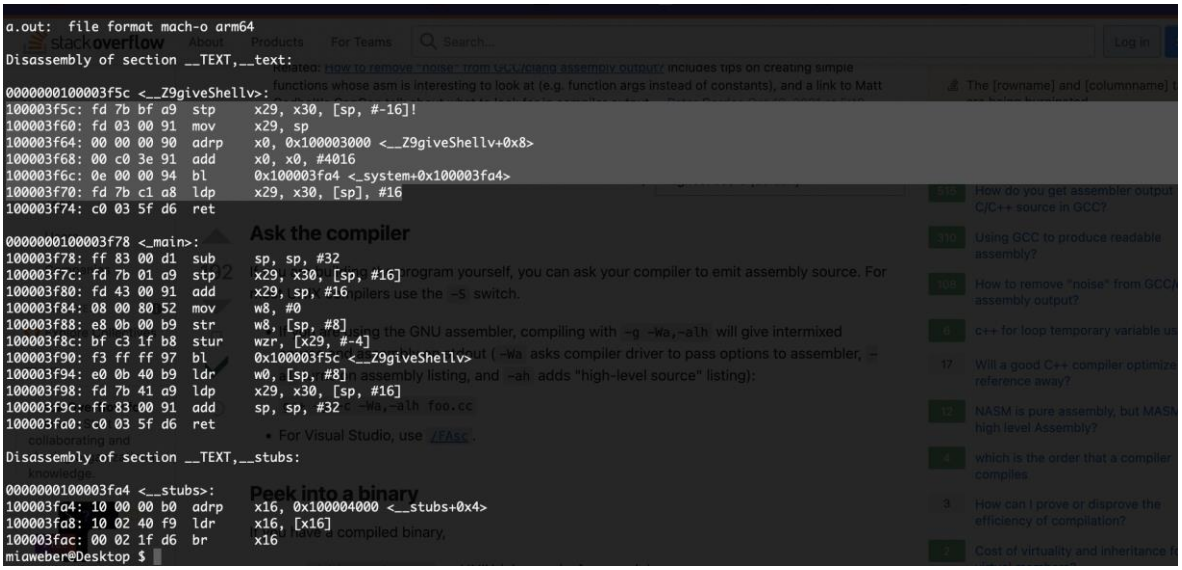
BELOW IS WORK:
Below is a screenshot of the assembly code for the authenticate function:



Below is a screenshot of the assembly code of the authenticate function from inside gdb:



Below is a screenshot of the assembly code for the c++ function that spawns a shell: (this is what I have to integrate into the binary file to spawn a shell after the user authenticates)


a.out: file format mach-o arm64
Disassembly of section __TEXT,__text:
0000000100003f5c <__Z9giveShellv>:
100003f5c: fd 7b bf a9 stp   x29, x30, [sp, #-16]!
100003f60: fd 03 00 91 mov   x29, sp
100003f64: 00 00 00 90 adrp   x0, 0x100003000 <__Z9giveShellv+0x8>
100003f68: 00 c0 3e 91 add   x0, x0, #4016
100003f6c: 0e 00 00 94 bl   0x100003fa4 <_system+0x100003fa4>
100003f70: fd 7b c1 a8 ldp   x29, x30, [sp], #16
100003f74: c0 03 5f d6 ret
0000000100003f78 <_main>:
100003f78: ff 83 00 d1 sub   sp, sp, #32
100003f7c: fd 7b 01 a9 stp   x29, x30, [sp, #16]
100003f80: fd 43 00 91 add   x29, sp, #16
100003f84: 08 00 80 52 mov   w8, #0
100003f88: e8 0b 00 b9 str   w8, [sp, #8]
100003f8c: bf c3 1f b8 stur   wzr, [x29, #-4]
100003f90: f3 ff ff 97 bl   0x100003f5c <__Z9giveShellv>
100003f94: e0 0b 40 b9 ldr   w0, [sp, #8]
100003f98: fd 7b 41 a9 ldp   x29, x30, [sp, #16]
100003f9c: ff 83 00 91 add   sp, sp, #32
100003fa0: c0 03 5f d6 ret
Disassembly of section __TEXT,__stubs:
0000000100003fa4 <__stubs>:
100003fa4: 10 00 00 b0 adrp   x16, 0x100004000 <__stubs+0x4>
100003fa8: 10 02 40 f9 ldr   x16, [x16]
100003fac: 00 02 1f d6 br   x16
miaweber@Desktop $