CSCI 420

MIA WEBER

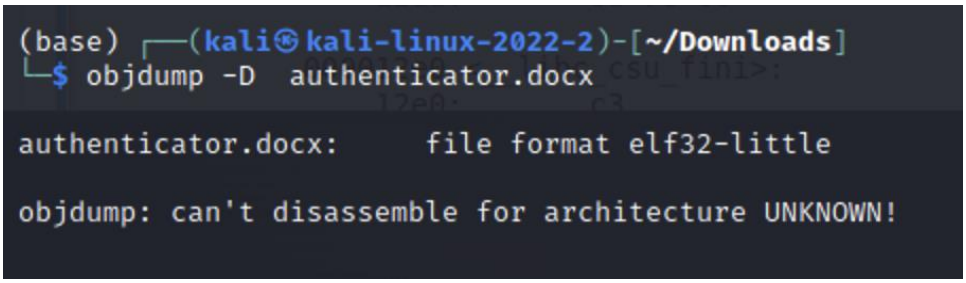
04/03/2023

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. Below is the result of attempting to disassemble the authenticator.docx file on my ARM based Kali Linux machine.



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, 0x080484d6 in authenticate ()
(gdb) disass authenticate
Dump of assembler code for function authenticate:
   0x080484ab <+0>:     push   %ebp
   0x080484ac <+1>:     mov    %esp,%ebp
   0x080484ae <+3>:     sub    $0x28,%esp
   0x080484b1 <+6>:     movl   $0x0,-0xc(%ebp)
   0x080484b8 <+13>:    sub    $0x8,%esp
   0x080484bb <+16>:    push   0x8(%ebp)
   0x080484be <+19>:    lea    -0x20(%ebp),%eax
   0x080484c1 <+22>:    push   %eax
   0x080484c2 <+23>:    call   0x8048360 <strcpy@plt>
   0x080484c7 <+28>:    add    $0x10,%esp
   0x080484ca <+31>:    sub    $0x8,%esp
   0x080484cd <+34>:    push   $0x8048630
   0x080484d2 <+39>:    lea    -0x20(%ebp),%eax
   0x080484d5 <+42>:    push   %eax
=> 0x080484d6 <+43>:    call   0x8048340 <strcmp@plt>
   0x080484db <+48>:    add    $0x10,%esp
   0x080484de <+51>:    test   %eax,%eax
   0x080484e0 <+53>:    je     0x80484fa <authenticate+79>
   0x080484e2 <+55>:    sub    $0x8,%esp
   0x080484e5 <+58>:    push   $0x8048639
   0x080484ea <+63>:    lea    -0x20(%ebp),%eax
   0x080484ed <+66>:    push   %eax
   0x080484ee <+67>:    call   0x8048340 <strcmp@plt>
   0x080484f3 <+72>:    add    $0x10,%esp
   0x080484f6 <+75>:    test   %eax,%eax
   0x080484f8 <+77>:    jne    0x8048501 <authenticate+86>
   0x080484fa <+79>:    movl   $0x1,-0xc(%ebp)
   0x08048501 <+86>:    mov    -0xc(%ebp),%eax
   0x08048504 <+89>:    leave
   0x08048505 <+90>:    ret
End of assembler dump.
(gdb) x/s $eax
0xffffcf88:     "1234"
(gdb) x/s $esp
0xffffcf70:     "\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:
   0x080484ab <+0>:     push   %ebp
   0x080484ac <+1>:     mov    %esp,%ebp
   0x080484ae <+3>:     sub    $0x28,%esp
   0x080484b1 <+6>:     movl   $0x0,-0xc(%ebp)
   0x080484b8 <+13>:    sub    $0x8,%esp
   0x080484bb <+16>:    push   0x8(%ebp)
   0x080484be <+19>:    lea    -0x20(%ebp),%eax
   0x080484c1 <+22>:    push   %eax
   0x080484c2 <+23>:    call   0x8048360 <strcpy@plt>
   0x080484c7 <+28>:    add    $0x10,%esp
   0x080484ca <+31>:    sub    $0x8,%esp
   0x080484cd <+34>:    push   $0x8048630
   0x080484d2 <+39>:    lea    -0x20(%ebp),%eax
   0x080484d5 <+42>:    push   %eax
   0x080484d6 <+43>:    call   0x8048340 <strcmp@plt>
   0x080484db <+48>:    add    $0x10,%esp
   0x080484de <+51>:    test   %eax,%eax
   0x080484e0 <+53>:    je     0x80484fa <authenticate+79>
   0x080484e2 <+55>:    sub    $0x8,%esp
   0x080484e5 <+58>:    push   $0x8048639
   0x080484ea <+63>:    lea    -0x20(%ebp),%eax
   0x080484ed <+66>:    push   %eax
=> 0x080484ee <+67>:    call   0x8048340 <strcmp@plt>
   0x080484f3 <+72>:    add    $0x10,%esp
   0x080484f6 <+75>:    test   %eax,%eax
   0x080484f8 <+77>:    jne    0x8048501 <authenticate+86>
   0x080484fa <+79>:    movl   $0x1,-0xc(%ebp)
   0x08048501 <+86>:    mov    -0xc(%ebp),%eax
   0x08048504 <+89>:    leave
   0x08048505 <+90>:    ret
End of assembler dump.
(gdb) x/s $eax
0xffffcf88:     "1234"
(gdb) x/s $ebp
0xffffcfa8:     "\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
0xffffcf70:     "\210\317\377\3779\206\004\b@\332\377", <incomplete sequence \367>
(gdb) x/s 0x8048340
0x8048340 <strcmp@plt>: "\377%\370\230\004\bh"
(gdb) x/s 0x80484fa
0x80484fa <authenticate+79>:    "\307E\364\001"
(gdb) x/s 0x8
0x8:    <error: Cannot access memory at address 0x8>
(gdb) x/s 0x8048639
0x8048639:      "0x0xmain"
(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.

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

┌──(kali㉿kali)-[~/Downloads]
└─$
```

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

In order to modify the binary authenticator.docx I used the tool hexedit and an ASCII to hexadecimal converter in order to determine what the correct hex digits are. When we use hexdump of the original authenticator.docx binary we can see that there is a system call to "clear" which clears the screen after the user authenticates. This can be seen in the screenshot below.

```
00000580  08 e8 ea fd ff ff 83 c4  10 eb 10 83 ec 0c 68 ac  |..............h.|
00000590  86 04 08 e8 d8 fd ff ff  83 c4 10 b8 00 00 00 00  |................|
000005a0  8d 65 f8 59 5b 5d 8d 61  fc c3 66 90 66 90 66 90  |.e.Y[].a..f.f.f.|
000005b0  55 57 31 ff 56 53 e8 25  fe ff ff 81 c3 31 13 00  |UW1.VS.%.....1..|
000005c0  00 83 ec 0c 8b 6c 24 20  8d b3 0c ff ff ff e8 39  |.....l$ .......9|
000005d0  fd ff ff 8d 83 08 ff ff  ff 29 c6 c1 fe 02 85 f6  |.........)......|
000005e0  74 23 8d b6 00 00 00 00  83 ec 04 ff 74 24 2c ff  |t#..........t$,.|
000005f0  74 24 2c 55 ff 94 bb 08  ff ff ff 83 c7 01 83 c4  |t$,U............|
00000600  10 39 f7 75 e3 83 c4 0c  5b 5e 5f 5d c3 8d 76 00  |.9.u....[^_]..v.|
00000610  f3 c3 00 00 53 83 ec 08  e8 c3 fd ff ff 81 c3 cf  |....S...........|
00000620  12 00 00 83 c4 08 5b c3  03 00 00 00 01 00 02 00  |......[.........|
00000630  30 78 61 62 63 31 32 33  00 30 78 30 78 6d 61 69  |0xabc123.0x0xmai|
00000640  6e 00 49 6e 76 61 6c 69  64 20 6f 70 74 69 6f 6e  |n.Invalid option|
00000650  3a 00 55 73 61 67 65 20  25 73 20 5b 70 61 73 73  |:.Usage %s [pass|
00000660  77 6f 72 64 5d 0a 00 63  6c 65 61 72 00 00 00 00  |word]..clear....|
00000670  57 65 6c 63 6f 6d 65 2c  20 79 6f 75 20 68 61 76  |Welcome, you hav|
00000680  65 20 61 63 63 65 73 73  20 74 6f 20 74 6f 70 20  |e access to top |
00000690  73 65 63 72 65 74 20 70  61 72 74 20 6f 66 20 74  |secret part of t|
000006a0  68 65 20 70 72 6f 67 72  61 6d 21 00 49 6e 76 61  |he program!.Inva|
000006b0  6c 69 64 20 70 61 73 73  77 6f 72 64 2e 20 54 72  |lid password. Tr|
000006c0  79 20 61 67 61 69 6e 21  00 00 00 00 01 1b 03 3b  |y again!.......;|
000006d0  30 00 00 00 05 00 00 00  64 fc ff ff 4c 00 00 00  |0.......d...L...|
000006e0  df fd ff ff 70 00 00 00  3a fe ff ff 90 00 00 00  |....p...:.......|
000006f0  e4 fe ff ff c4 00 00 00  44 ff ff ff 10 01 00 00  |........D.......|
00000700  14 00 00 00 00 00 00 00  01 7a 52 00 01 7c 08 01  |.........zR..|..|
00000710  1b 0c 04 04 88 01 00 00  20 00 00 00 1c 00 00 00  |........ .......|
00000720  10 fc ff ff 70 00 00 00  00 0e 08 46 0e 0c 4a 0f  |....p......F..J.|
00000730  0b 74 04 78 00 3f 1a 3b  2a 32 24 22 1c 00 00 00  |.t.x.?.;*2$"....|
00000740  40 00 00 00 67 fd ff ff  5b 00 00 00 00 41 0e 08  |@...g...[....A..|
00000750  85 02 42 0d 05 02 57 c5  0c 04 04 00 30 00 00 00  |..B...W.....0...|
00000760  60 00 00 00 a2 fd ff ff  a4 00 00 00 00 44 0c 01  |`............D..|
00000770  00 47 10 05 02 75 00 44  0f 03 75 78 06 10 03 02  |.G...u.D..ux....|
00000780  75 7c 02 8f c1 0c 01 00  41 c3 41 c5 43 0c 04 04  |u|......A.A.C...|
00000790  48 00 00 00 94 00 00 00  18 fe ff ff 5d 00 00 00  |H...........]...|
```

Because adding more digits will alter the binary in unintentional ways, there are only five digits to work with so "/bin/sh" won't work. Just "sh" will though so we can use hexedit in order to change the hex bytes to represent sh as seen below.

```
00000620  12 00 00 83 c4 08 5b c3  03 00 00 00 01 00 02 00  |......[.........|
00000630  30 78 61 62 63 31 32 33  00 30 78 30 78 6d 61 69  |0×abc123.0×0xmai|
00000640  6e 00 49 6e 76 61 6c 69  64 20 6f 70 74 69 6f 6e  |n.Invalid option|
00000650  3a 00 55 73 61 67 65 20  25 73 20 5b 70 61 73 73  |:.Usage %s [pass|
00000660  77 6f 72 64 5d 0a 00 73  68 00 00 00 00 00 00 00  |word]..sh.......|
00000670  57 65 6c 63 6f 6d 65 2c  20 79 6f 75 20 68 61 76  |Welcome, you hav|
00000680  65 20 61 63 63 65 73 73  20 74 6f 20 74 6f 70 20  |e access to top |
00000690  73 65 63 72 65 74 20 70  61 72 74 20 6f 66 20 74  |secret part of t|
000006a0  68 65 20 70 72 6f 67 72  61 6d 21 00 49 6e 76 61  |he program!.Inva|
000006b0  6c 69 64 20 70 61 73 73  77 6f 72 64 2e 20 54 72  |lid password. Tr|
000006c0  79 20 61 67 61 69 6e 21  00 00 00 00 01 1b 03 3b  |y again!.......;|
000006d0  30 00 00 00 05 00 00 00  64 fc ff ff 4c 00 00 00  |0.......d...L...|
000006e0  df fd ff ff 70 00 00 00  3a fe ff ff 90 00 00 00  |....p...:.......|
000006f0  e4 fe ff ff c4 00 00 00  44 ff ff ff 10 01 00 00  |........D.......|
00000700  14 00 00 00 00 00 00 00  01 7a 52 00 01 7c 08 01  |.........zR..|..|
00000710  1b 0c 04 04 88 01 00 00  20 00 00 00 1c 00 00 00  |........ .......|
```

Now we can execute the binary again and see if the shell is spawned after the user authenticates, which it does. This can be seen in the image below.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ ./authenticator.docx 0×0xmain
$ ls
authenticator.docx  giveShell.cpp  hope.docx  littleE.docx  modifed2.docx  modified3.docx  modified.docx  newauth.docx
$ whoami
kali
$ exit
Welcome, you have access to top secret part of the program!
```

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.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ md5sum authenticator.docx
69b72191324e806a484e3a52664b8380  authenticator.docx
┌──(kali㉿kali)-[~/Downloads]
└─$ shasum authenticator.docx
b19badcab0a7bf759de1a310cccc72598b6720b6  authenticator.docx
┌──(kali㉿kali)-[~/Downloads]
└─$
```

Below is a screenshot of the md5 and sha1 checksums of the modified binary.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ md5sum authenticator.docx
ff10a16a1b04809bac96fb4f8cf8d62e  authenticator.docx
┌──(kali㉿kali)-[~/Downloads]
└─$ shasum authenticator.docx
892cd439972cf48cab8d5574e4bea469c2756342  authenticator.docx
┌──(kali㉿kali)-[~/Downloads]
└─$
```