George, Levi

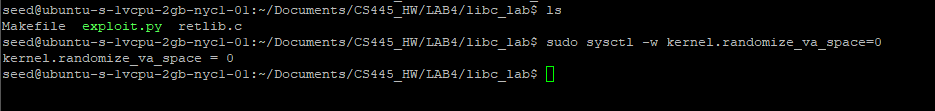
CS44500 – Computer Security

02/15/23

Lab 4 – Return to Lib C

Env Setup.

Turning off ASLR



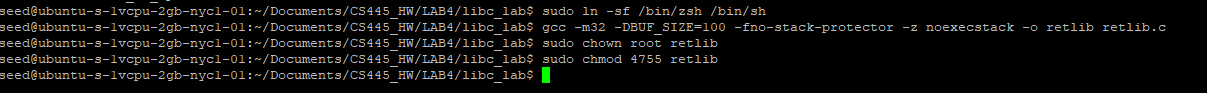
Changing from Dash to Zsh



It appears that the ret\_lib.c has a vulnerability due to using strcpy in the bof function.

Additionally I see that we read in about 1000 characters from our badfile, this may affect how I construct my payload.

Compilation with no stack protector nor executable stack. (With N being 100 for the DBUF\_SIZE)



TASK 1. Getting our libc function’s address in memory

Touch badfile and run gdb

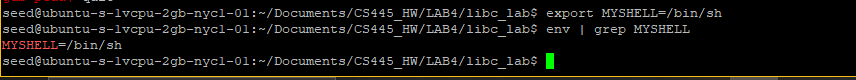


Getting my libc function addresses

Text

Description automatically generated with medium confidence

TASK 2. Putting shell string in the memory



Store my shell within a env variable (I can access my env vars from within my program) to use later.

I create my printenv program



Which has this code

Text

Description automatically generated

I get this for my /bin/sh address

Text

Description automatically generated

However, this is a 64 bit address,



I recompile for 32-bit using -m32 and get the following address



TASK 3. Launching my attack

I insert my function addresses into my python code.

Text

Description automatically generated

I generate a gdb version of the vulnerable code via gcc using deactivated countermeasures, setting the code to 32 bit and with the buffer size as 100.



I set a breakpoint a bof (gdb command b bof, then r)

I find the address of $ebp and buffer then the difference between them.

Text

Description automatically generated

Using this as my basis, I set my insertion points as such.

Text

Description automatically generated

However, this failed. I got a segfault.

Text

Description automatically generated

I checked my handout to see how the order was in the class demo and realized that I mis-ordered my commands. I set them to this:

Text

Description automatically generated

This arrangement awarded me the root shell. (I opened my exploit to take the shot from above)

Text

Description automatically generated

Interestingly, I did attempt to use the 64-bit address of my shell command.

A screenshot of a computer

Description automatically generated with medium confidence

But uninterestingly, it just didn’t give me a new shell. (I suspect that my system and exit command ran, but since I wasn’t getting the right address for the shell command, I didn’t get any shell.)

A screenshot of a computer

Description automatically generated with medium confidence

TASK 3b. Variation 1 – No Exit()

A screenshot of a computer

Description automatically generated with medium confidence

I set my exit command as the above and got a segmentation fault.

Text

Description automatically generated

This must be caused by us not exiting bof when we execute our attack.

Task 3c. Variation 2 – change program name

I changed my programs name and I also returned my shell address to the original 32-bit address

This won’t succeed due to my program’s name influencing the addressing of my shell address in my env variables. Which we see below.

Graphical user interface, text

Description automatically generated