BÁO CÁO THỰC HÀNH LAB 4

**Môn học: Cơ chế hoạt động của mã độc**

**Tên chủ đề: Simple Worm**

*GVHD: Nguyễn Hữu Quyền*

1. **THÔNG TIN CHUNG:**

*(Liệt kê tất cả các thành viên trong nhóm)*

Lớp: NT230.N21.ATCL

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Họ và tên | MSSV | Email |
| 1 | Bạch Văn Xuân Thông | 20521978 | 20521978@gm.uit.edu.vn |
| 2 | Trần Tấn Tài | 20521862 | 20521862@gm.uit.edu.vn |
| 3 | Phạm Bá Tín | 20522016 | 20522016@gm.uit.edu.vn |

1. **NỘI DUNG THỰC HIỆN:[[1]](#footnote-0)**

|  |  |  |
| --- | --- | --- |
| **STT** | **Công việc** | **Kết quả tự đánh giá** |
| 1 | Yêu cầu 1 | 100% |
| 2 | Yêu cầu 2 | 10% |

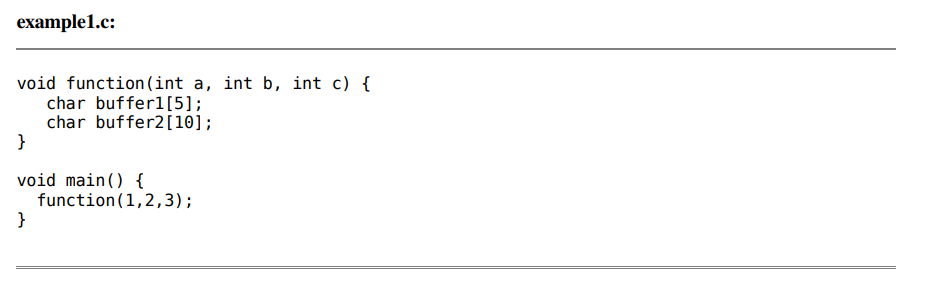
**Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.**

**BÁO CÁO CHI TIẾT**

**Yêu cầu 1:**

**Đọc cẩn thận tập tin “stack\_smashing”. Chạy từng ví dụ để hiểu cơ chế khai thác lỗi buffer overflow**

**Example1:**

$ gcc -S -o example1.s example1.c

Text

Description automatically generated

Text

Description automatically generated with medium confidence

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**Buffer Overflows**

**Example2:**

**Graphical user interface, text

Description automatically generated with medium confidence**

Lỗi tràn bộ đệm ở **strcpy(buffer,str);** do buffer chỉ có 16 byte nhưng chuỗi mà nó copy vào lại chứa 256 ký tự A (0x41), điều này làm cho bộ đệm trong ngăn xếp phía sau bị ghi đè. Nó ghi đè lên SFP, RET, và thậm chí là \*str!. Điều đó có nghĩa là return address hiện là 0x41414141.

Text

Description automatically generated

Text

Description automatically generated with medium confidence

**Example3**

**Graphical user interface, text, application

Description automatically generated**

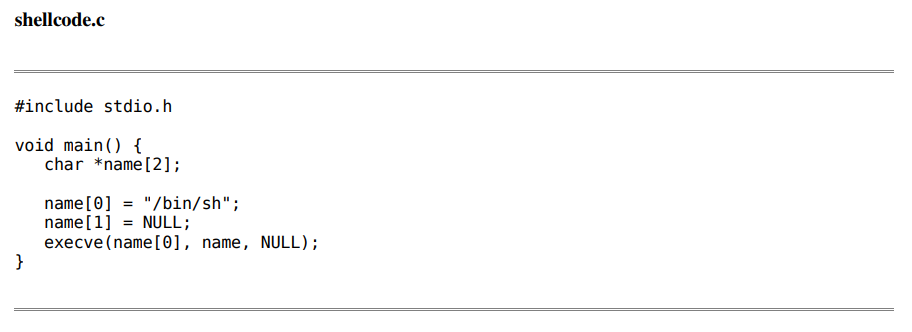
**Text

Description automatically generated**

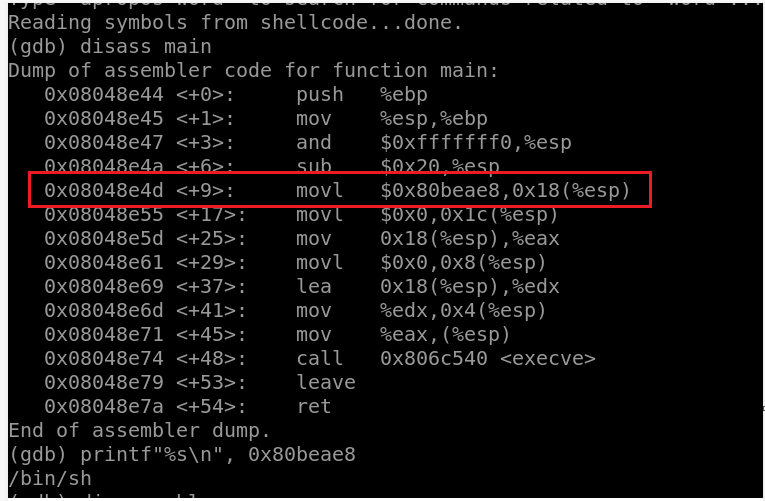
**Text

Description automatically generated**

**Shell Code**



Copy giá trị 0x80beae8 (địa chỉ của chuỗi "/bin/sh") vào vị trí đầu tiên của con trỏ name[].



Đưa giá trị NULL vào name[1].

Text

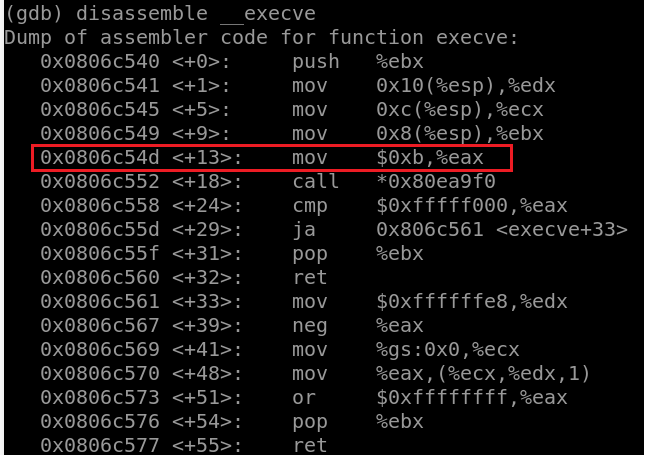
Description automatically generated with medium confidence

Các lệnh tiếp theo sẽ lần được giá trị vào và gọi hàm (**execve(name[0], name, NULL);** )

Text

Description automatically generated

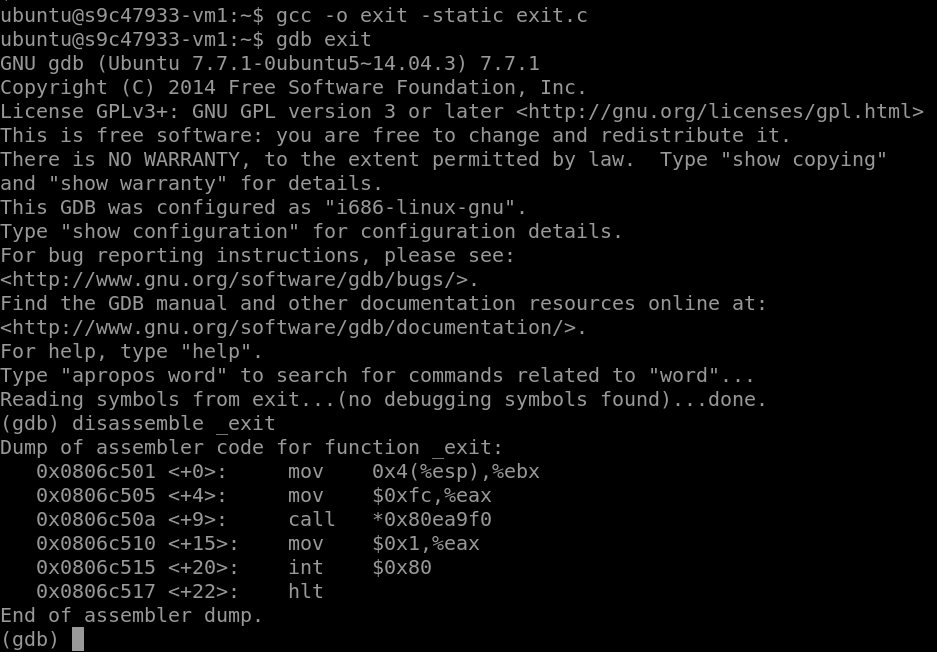
Copy 0xb (11 decimal) onto the stack. This is the index into the syscall table. 11 is execve.



**Exit**

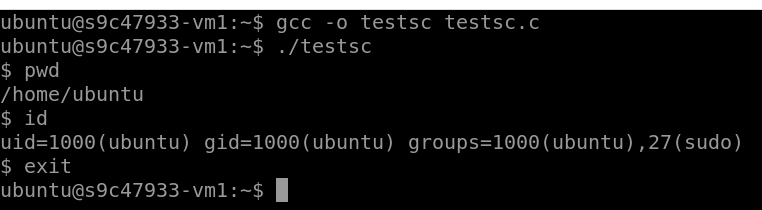
Text

Description automatically generated with medium confidence

****

**Graphical user interface, text, application

Description automatically generated**

****

**Graphical user interface, text, application

Description automatically generated**

**Text

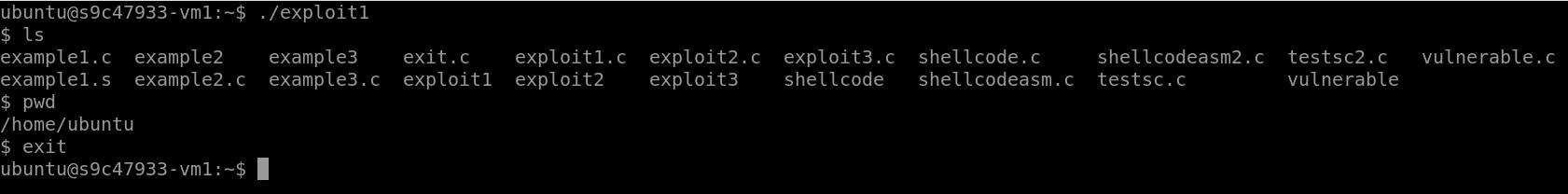
Description automatically generated**

**Writing an Exploit overflow1**

**Graphical user interface, text, application, email

Description automatically generated**

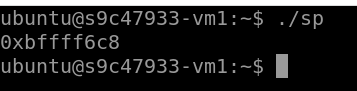
Đoạn code lợi dụng lỗ hổng của hàm **strcpy** là không kiểm tra độ dài chuỗi khi copy, nên nó sẽ ghi đè đoạn shellcode của chúng ta lên buffer, dẫn đến chương trình bị overflow

****

**Graphical user interface

Description automatically generated with medium confidence**

Đây là chương trình để in con trỏ ngăn xếp



**Exploit2**

**Text

Description automatically generated**

Đoạn code đầu tiên định nghĩa giá trị mặc định cho offset và buffer size sử dụng trong chương trình. Tiếp theo là khai báo đoạn shellcode được sử dụng.

hàm get\_sp() để lấy địa chỉ của stack pointer (ESP) hiện tại.

Tiếp theo nó sẽ điều chỉnh bsize và offset tùy thuộc vào đối số đầu vào và dùng hàm malloc để cấp phát bộ nhớ.

Chương trình lấy địa chỉ của shellcode bằng cách lấy địa chỉ của ESP hiện tại và trừ đi offset.

Tiếp theo là dùng vòng lặp 4 byte để ghi đè shellcode.

Lưu giá trị của buff vào biến môi trường EGG sao đó sử dụng hàm system để thực thi shellcode.

Text

Description automatically generated

Theo tác giả thì tấn công shellcode theo cách này thì cần ít nhất 100 và tệ nhất là vài nghìn lần thử 🡪 Để không cần thử nhiều như thế ta sử dụng NOP theo cách exploit3 dưới đây.

**Exploit3**

Đây là đoạn code nâng cấp dựa trên exploit2

A picture containing text

Description automatically generated

Vòng lặp này sẽ ghi đè địa chỉ trở về (return address) trong bộ đệm bằng cách thay thế giá trị trước đó bằng địa chỉ mới được tính toán từ hàm get\_sp() - offset

Text

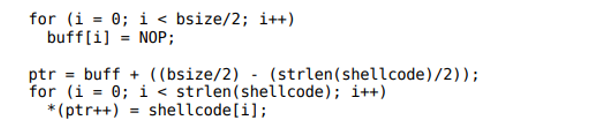
Description automatically generated

Đầu tiên, một vòng lặp được sử dụng để đặt một loạt các lệnh **NOP** vào nửa đầu của chuỗi **buff**. Điều này là để đảm bảo rằng **shellcode** sẽ được đặt ở vị trí có độ dài chính xác.

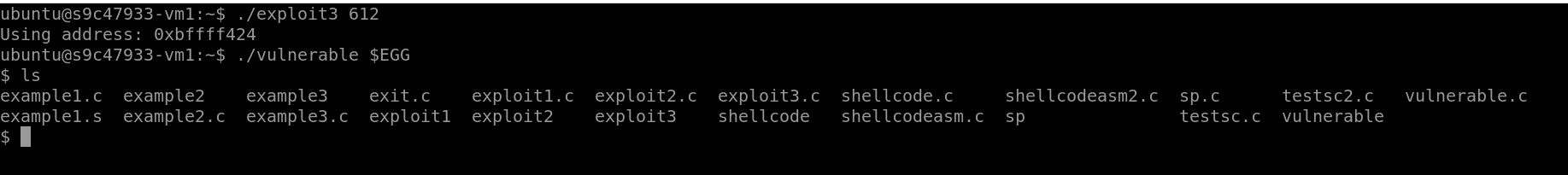
Sau đó, biến **ptr** được khởi tạo để trỏ đến giữa của chuỗi buff (tức là vị trí **bsize/2**).

Vị trí cuối cùng để chèn **shellcode** được tính toán bằng cách lấy giá trị trung bình của vị trí ptr và độ dài của **shellcode**, và sau đó trừ đi một lượng byte làm tròn xuống (**/2**). Điều này đảm bảo rằng **shellcode** được đặt ở vị trí giữa của chuỗi **buff** và sẽ không tràn qua phần đệm **NOP** được đặt ở nửa đầu của chuỗi

Cuối cùng, vòng lặp được sử dụng để chèn **shellcode** vào trong chuỗi buff bằng cách sao chép từng byte của **shellcode** vào các vị trí liên tiếp trong bộ đệm bắt đầu từ vị trí được tính toán ở trên

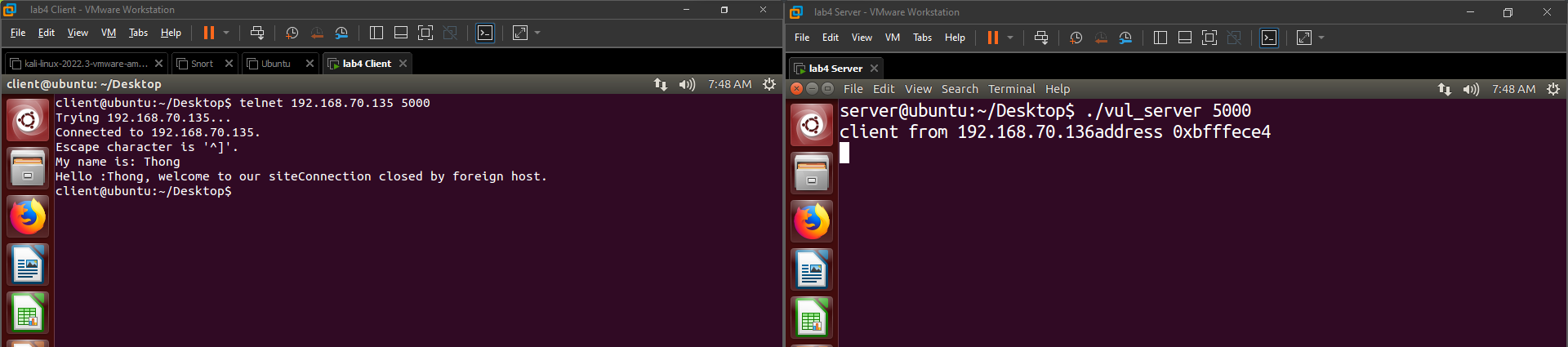


Một lựa chọn tốt cho kích thước bộ đệm của chúng tôi là khoảng 100 byte so với kích thước của bộ đệm mà chúng tôi đang cố gắng tràn. Bộ đệm mà chúng tôi đang cố gắng tràn dài 512 byte, vì vậy chúng tôi sẽ sử dụng 612.

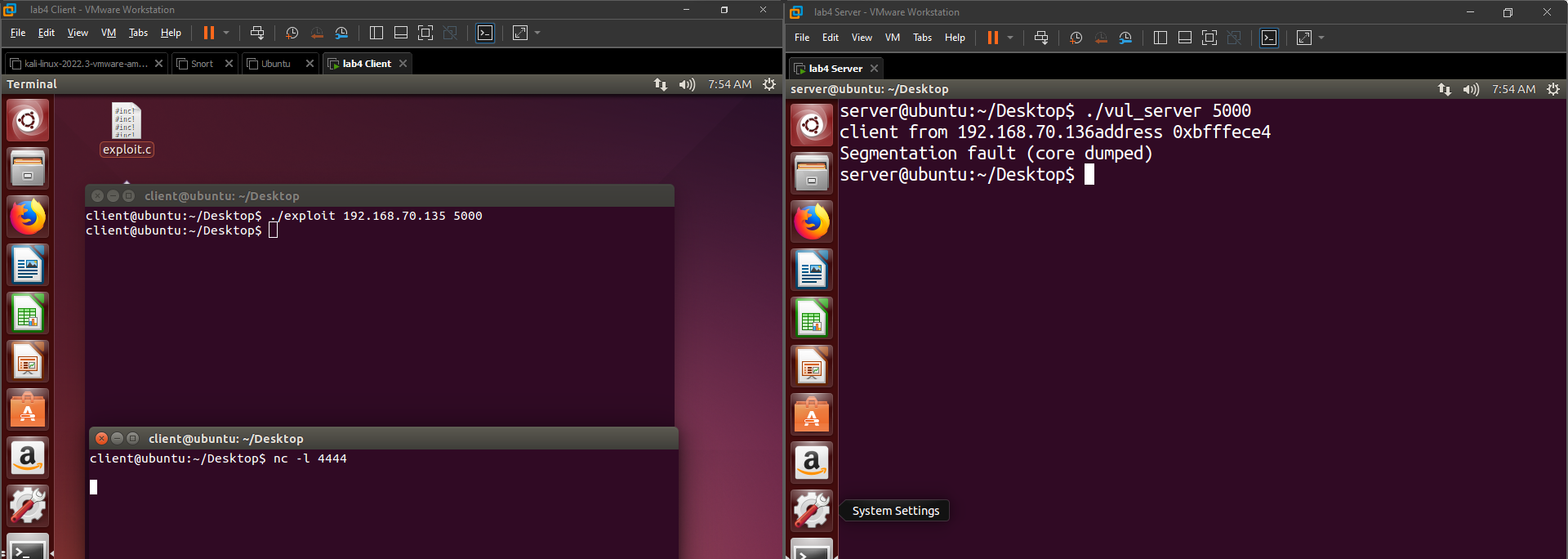


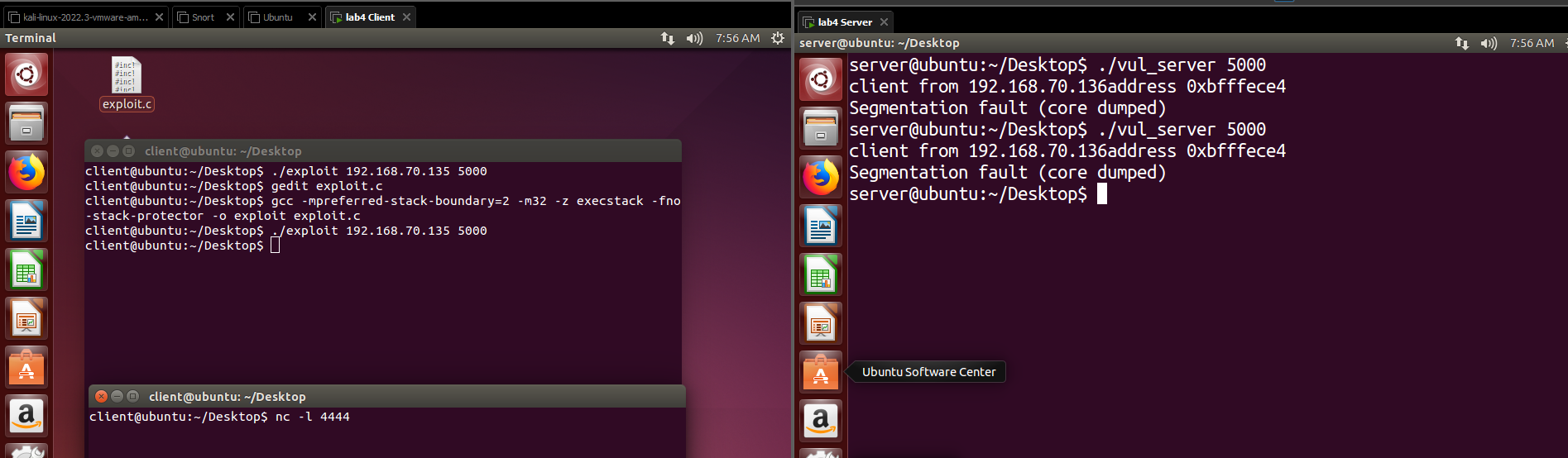
**Remote Buffer Overflow**

Thực hiện telnet thành công, server trả về client chuỗi text còn server sẽ hiển thị địa chỉ RET



Sau khi thực thi code theo bài lab sẽ xuất hiện lỗi segmentation fault

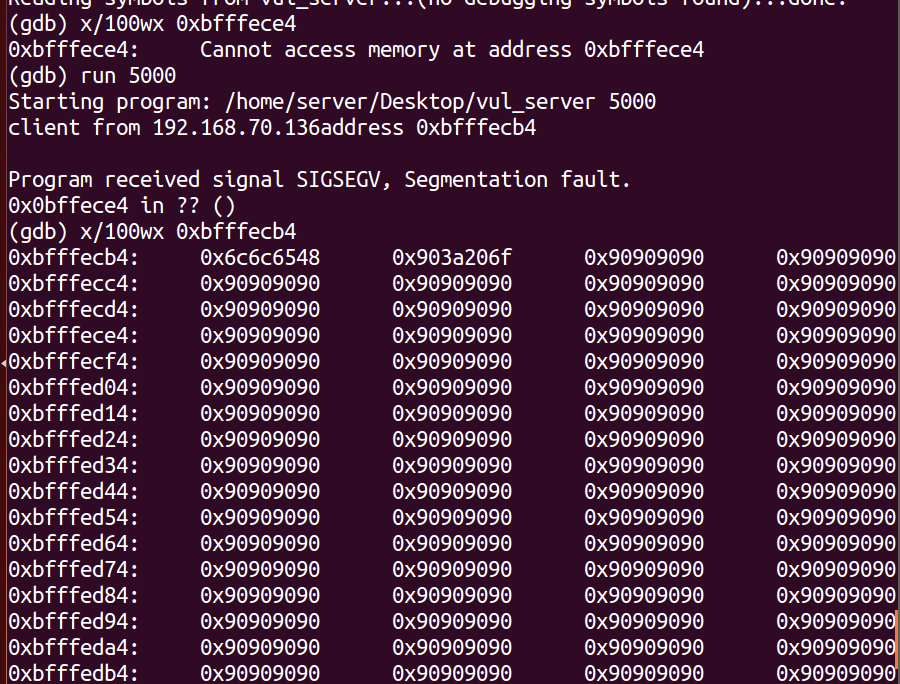


Có thể địa chỉ RET trong code cung cấp đã sai, ta thử tìm lại địa chỉ RET mới. Sau khi thực thi telnet ta sẽ thấy server cho ta một địa chỉ RET thử thay địa chỉ này vào xe

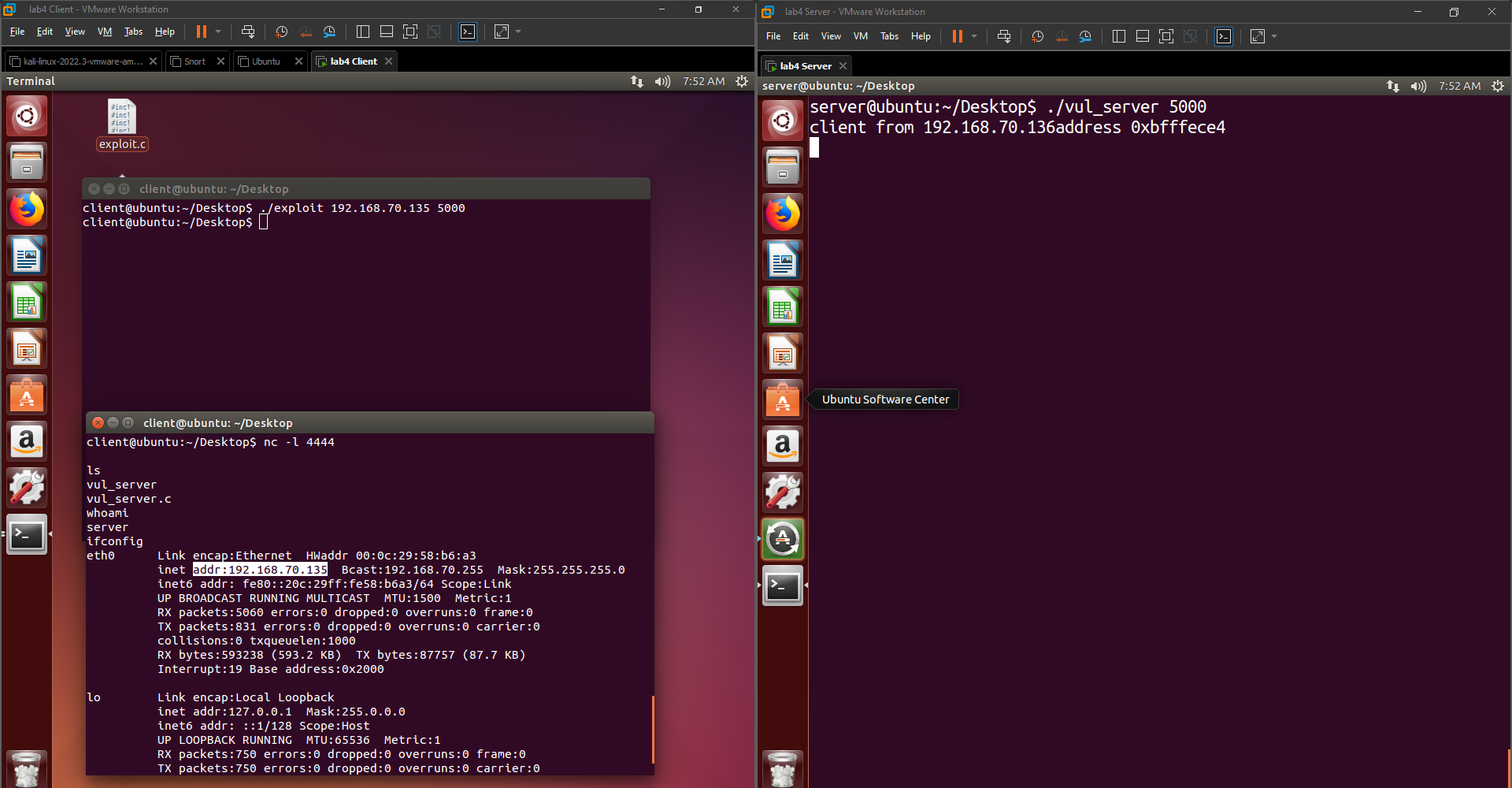
Vẫn không thành công, ta thử debug chương trình chứa lỗi bên server xem



Không thể truy cập vào địa chỉ đó thử lấy địa chỉ khác bằng lệnh run 5000. Khi xem xét địa chỉ mới ta có không thấy đoạn shellcode ta cần chèn vào ở đâu, nên có thể đây là địa chỉ sai. Vì vậy ta sẽ lấy một địa chỉ tùy ý ở đây làm địa chỉ RET nhằm mục đích trỏ vào các lệnh NOP (lưu ý: +5 bytes)



Thực hiện exploit thành công



**\*Câu hỏi thêm**

**- Có thể thay đổi port 4444 thành port khác được không ?**

Port xxxx sẽ được phân tích thành y\*256+z (Hệ thống lưu dạng hex nên chỉ số thứ 3 là bội số 16^2=256). Do đó, y và z được tính bằng cách lấy xxxx chia lấy dư cho 256, y là thương và z là số dư. shellcode[39] = y và shellcode[40] = z, áp dụng tương tự đối với các port khác tùy ý

A picture containing text

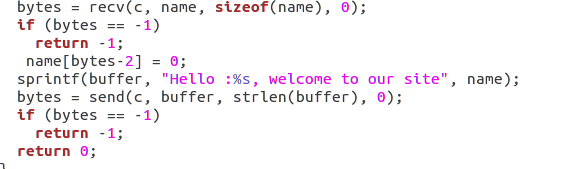
Description automatically generated

Text

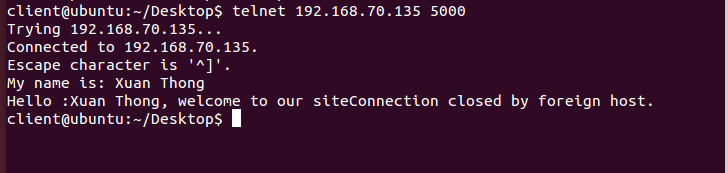
Description automatically generated

**- Tại sao phải cộng thêm 5 bytes vào địa chỉ RET ?**

Lỗi buffer overflow xảy ra khi ta copy giá trị từ biến name vào buffer mà không cần biết biến name có vượt quá kích thước của buffer hay không



Sau khi telnet thì server yêu cầu giá trị biến name để nhập vào mà không cần kiểm tra độ dài của biến name. Sau đó sever sẽ xuất hiện giá trị của RET để thay đổi RET của code exploit. Tuy nhiên ta thấy đoạn text “welcome…” cũng được đưa vào buffer, Do đó ta phải thêm 5 bytes vào RET nhận được để RET trỏ về đoạn code cần thực thi.



**Yêu cầu 2:**

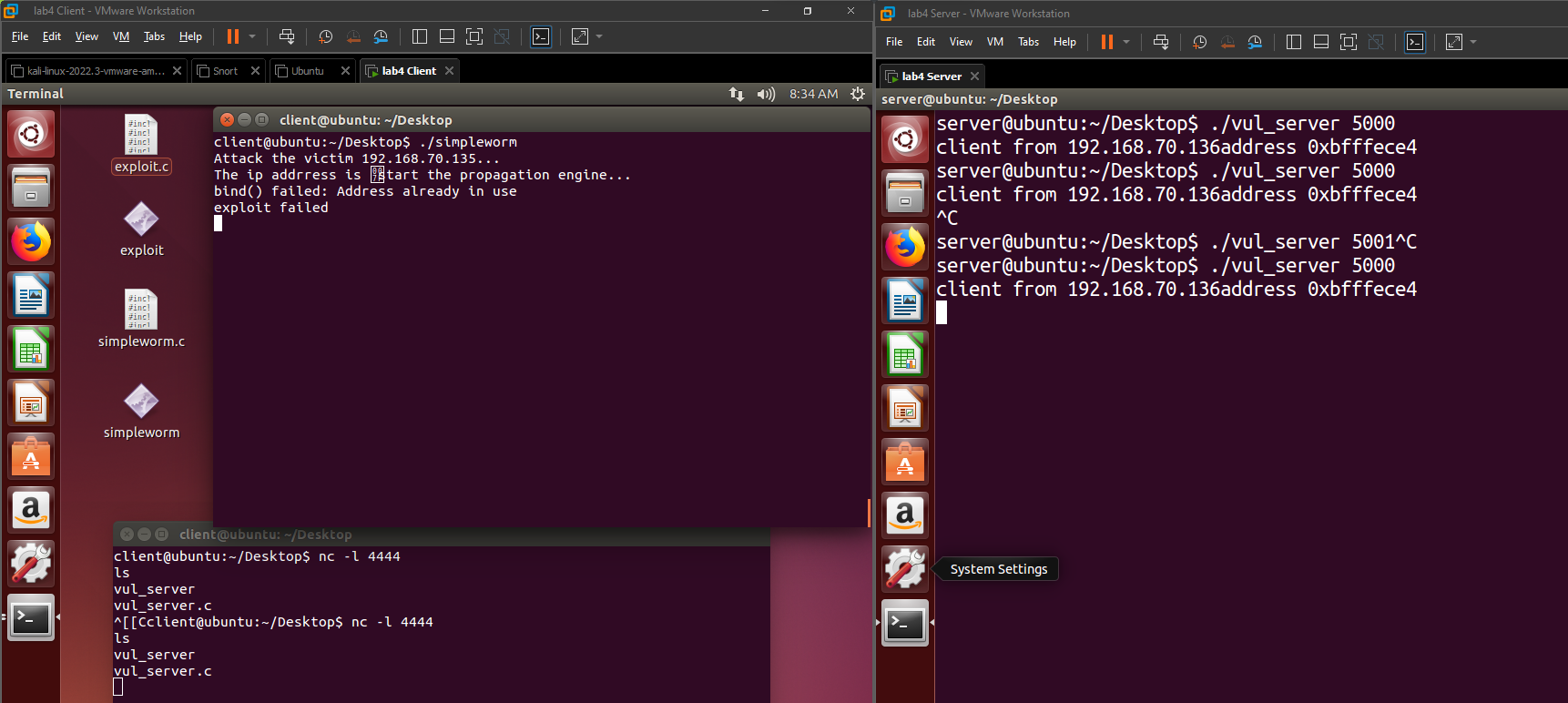
**Nhiệm vụ cuối cùng trong bài lab này là sau khi Worm đã lây nhiễm thành công trên máy chủ thứ 1 và attacker khai thác thành công thì Worm sẽ tự động đây lây nhiễm sang máy chủ khác và tự động thực thi ./vulner\_server 5000.**

Ý tưởng của bài sẽ cải tiến code exploit của remote buffer overflow để lây lan worm. Simpleworm kết nối đến server qua port 5000, sTuy nerver sẽ kết nối ngược tới client qua port 4444. Tiếp theo simpleworm sẽ thực hiện lệnh nc để mở port 10000 và ghi dữ liệu từ file simpleworm từ client sang máy chủ và thực thi file simpleworm trên server

Code thực hiện như sau



Tuy nhiên nhóm không thể hoàn thành nhiệm vụ của simpleworm là lây lan sang máy khác và thực hiện code bị lỗi



Code tham khảo nằm trong file đính simpleworm.c kèm.

1. Ghi nội dung công việc, các kịch bản trong bài Thực hành [↑](#footnote-ref-0)