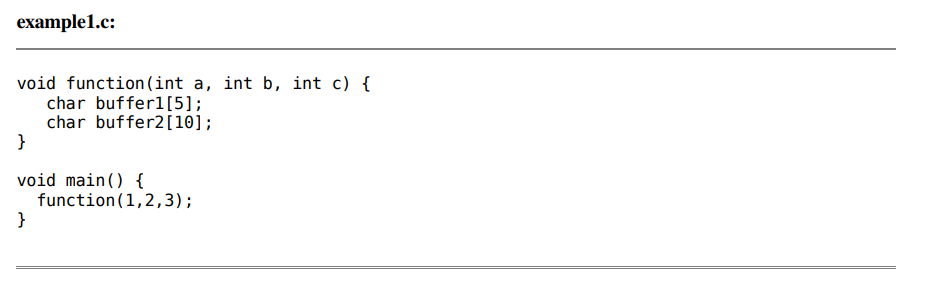
**Example1:**

$ 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**

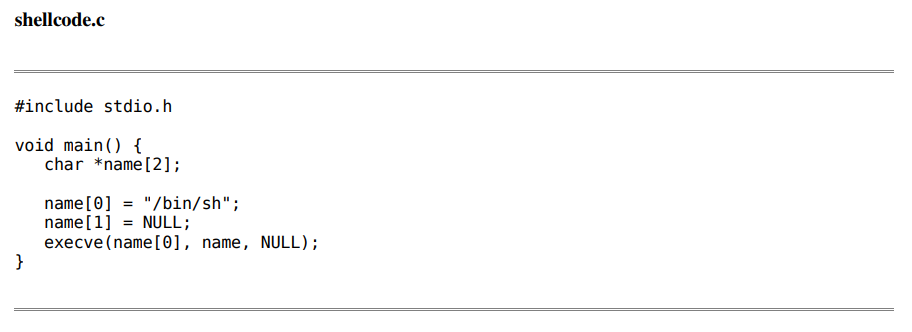
**Text

Description automatically generated**

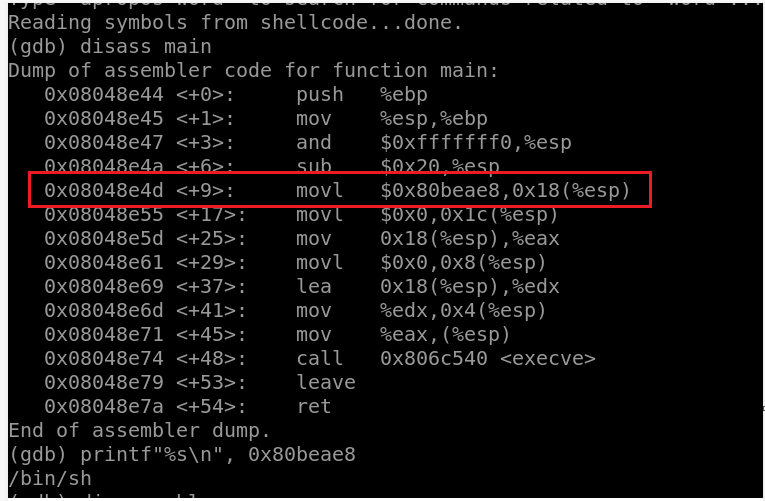
**Text

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

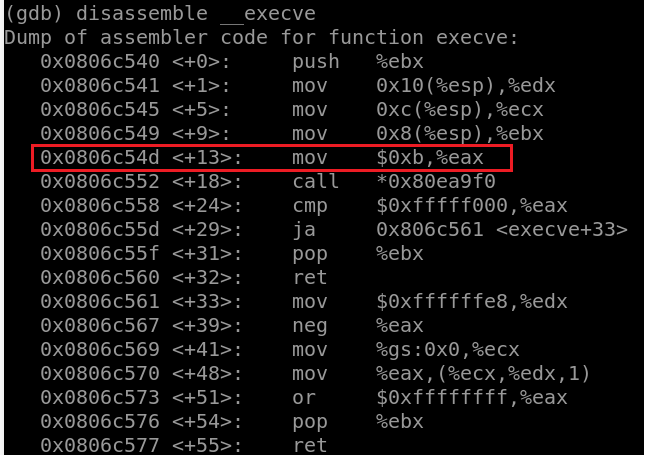
Description automatically generated with medium confidence

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

Text

Description automatically generated

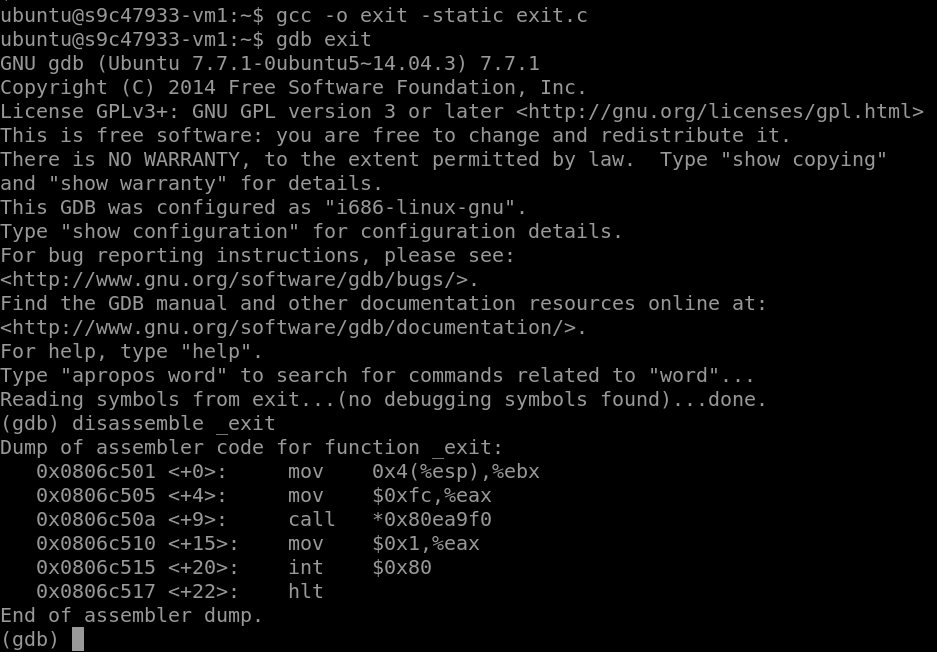
Copy 0xb (11 decimal) onto the stack. This is the index into the syscall table. 11 is execve.



**Exit**

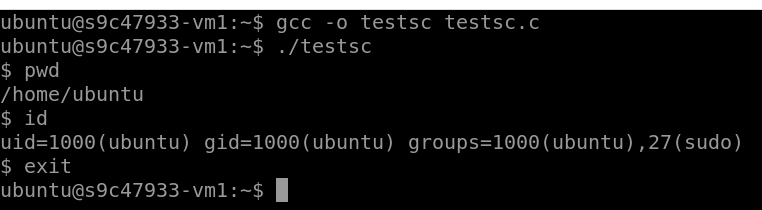
Text

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**

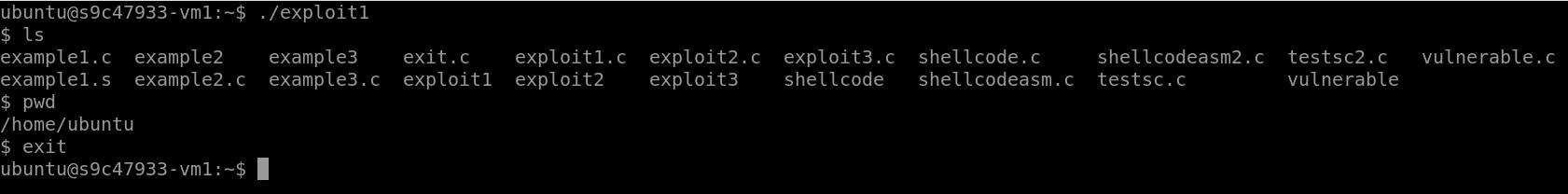
**Writing an Exploit**

**overflow1**

**Graphical user interface, text, application, email

Description automatically generated**

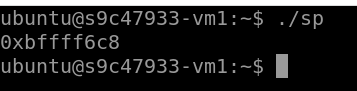
Đoạn code lợi dụng lỗ hổng của hàm **strcpy** là không kiểm tra độ dài chuỗi khi copy, nên nó sẽ ghi đè đoạn shellcode của chúng ta lên buffer, dẫn đến chương trình bị overflow

****

**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

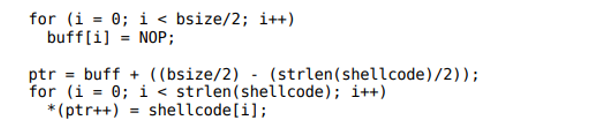
Description automatically generated

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.

