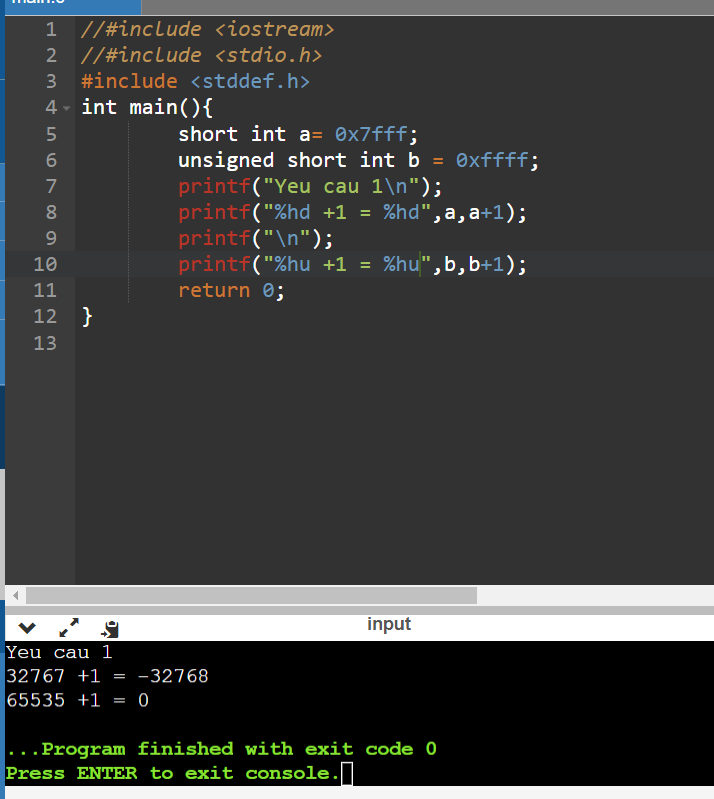
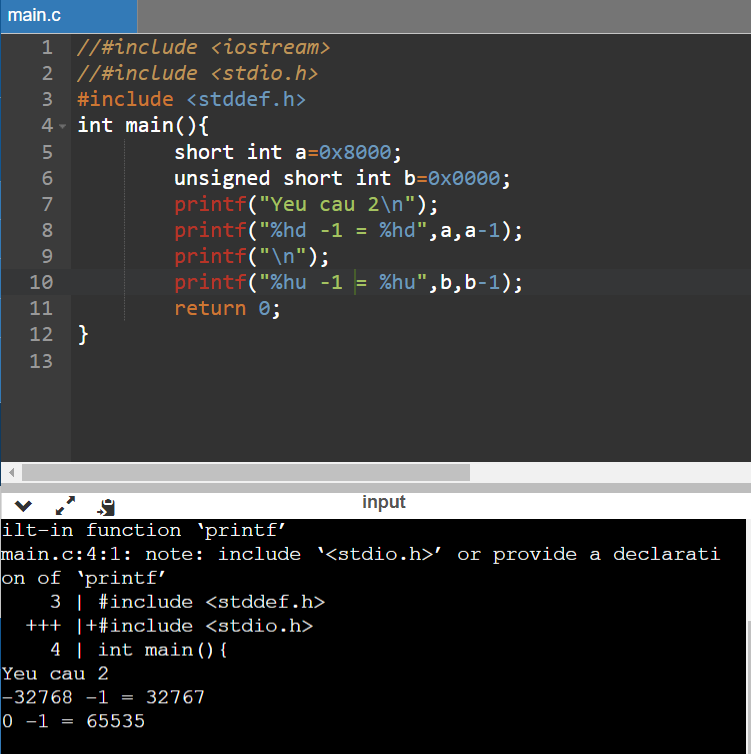
Yêu cầu 1. Sinh viên giải thích kết quả thực hiện, vì sao ta có được những kết quả

như hình trên? Khi nào xảy ra tràn trên?

* Khi vượt ngưỡng cho phép của số bit của 1 con số sẽ xảy ra hiện tượng tràn trên và tương tự với lỗi tràn dưới





Yêu cầu 2. Sinh viên giải thích kết quả thực hiện, vì sao ta có được những kết quả như hình trên? Khi nào xảy ra tràn dưới?

* Các tập lệnh assembly của máy tính không phân biệt giữa só có dấu và không dấu, dữ liệu liệu tồn tại và tính toán ở dạng nhị phân
* Ví dụ phép cộng 0x7fff+1 các toán hạng được chuyển sang dạng nhị phân để tính toán

0111 1111 1111 1111 + 1 = 1000 0000 0000 0000 = 0x8000(32768)

* Vì max của short int chỉ là 32767 nên khi vượt quá giới hạn nó sẽ quay lại giá gị ban đầu là -32768
* Và tương tự với tràn dưới thì vì giá trị nhỏ nhất của short int là -32768 nên khi – 1 thì nó sẽ quay lại giá trị lớn nhất là 32767

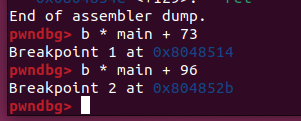
Khai thác tràn số nguyên: Ví dụ 1

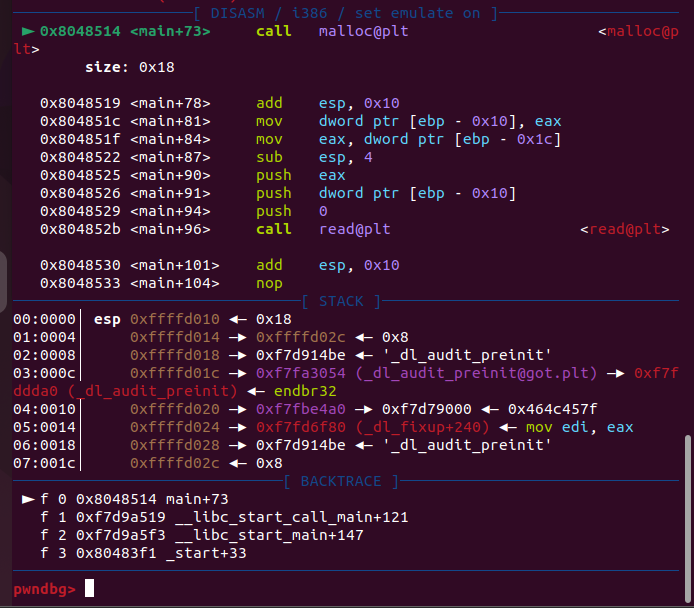


* Dùng gdb để mở file malloc-overflow

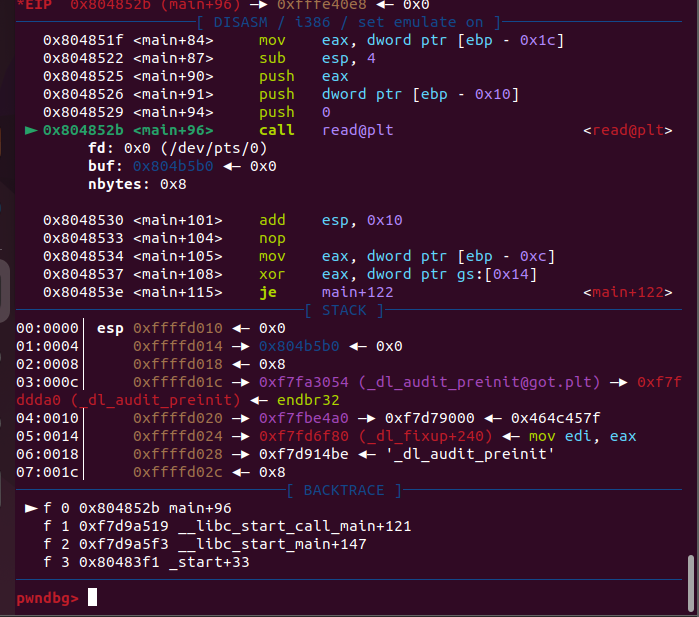


* Sau đó disass main





* R để chạy chương trình và nhập và nhập vào 1 số nguyên dương ví dụ như là 8
* Hình trên là trạng thái stack trước khi gọi hàm malloc, ta thấy hàm này cần 1 tham số có giá trị là data\_len + 0x10 = 8 + 0x10 = 0x18 bytes

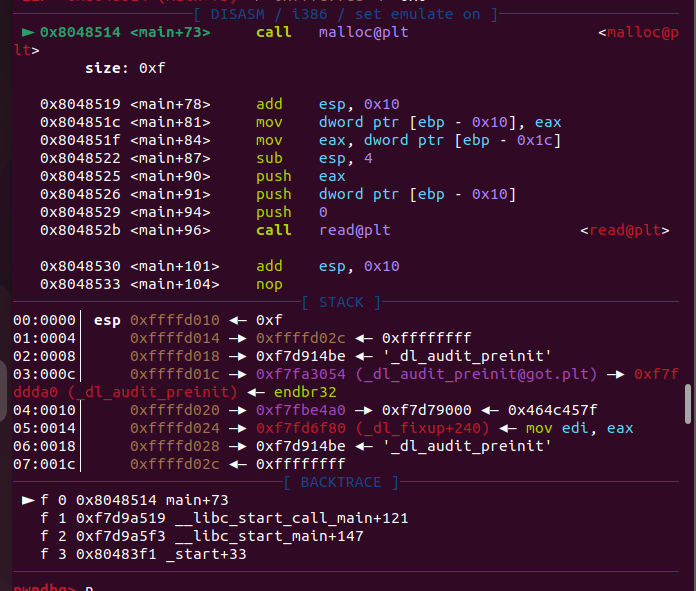


* Ta debug đến hàm read sẽ thấy tham số thứ 3 tương ứng là độ dài cần đọc , cũng được gán giá trị là data\_len=8(0x8) như đã nhập

Yêu cầu 3. Với data\_len nhập vào là -1, hàm malloc() sẽ nhận giá trị tham số bao nhiêu? Read sẽ đọc chuỗi có giới hạn là bao nhiêu byte? Giải thích các giá trị? Lưu ý: Báo cáo các giá trị sau khi đã chuyển sang hệ 10. Ví dụ: 0xB là 11



* Ta thử nhập 1 giá trị data\_len bất thường là -1 để xem hoạt động của chương trình như thế nào



* Giá trị -1 sẽ được lưu thành 0xffffffff
* Hàm malloc nhận tham số có giá trị 0xf do khi 0xffffffff + 0x10 sẽ xảy ra tràn số

Ảnh có chứa văn bản

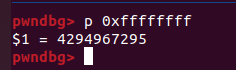
Mô tả được tạo tự động

* Ta thấy kết quả đã vượt quá phạm vi biểu diễn 4 bytes do đó sẽ bị cắt bớt bytes đầu là 1. Nên kết quả sẽ lưu thành 0000 000f

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Hàm read nhận tham số thứ 3 là 0xffffffff tuy nhiên hàm này nhận số nguyên không dấu do đó read sẽ hiểu thành

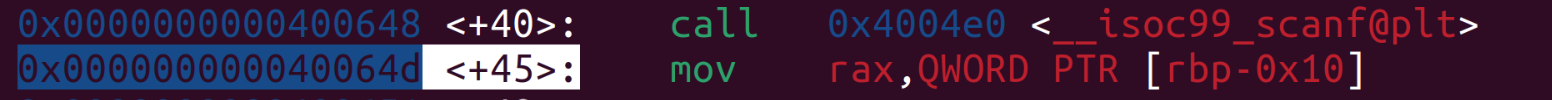
 là số ký tự tối đa read đọc

Yêu cầu 4. Sinh viên thử tìm 1 giá trị của a để chương trình có thể in ra thông báo “OK! Cast overflow done”? Giải thích?

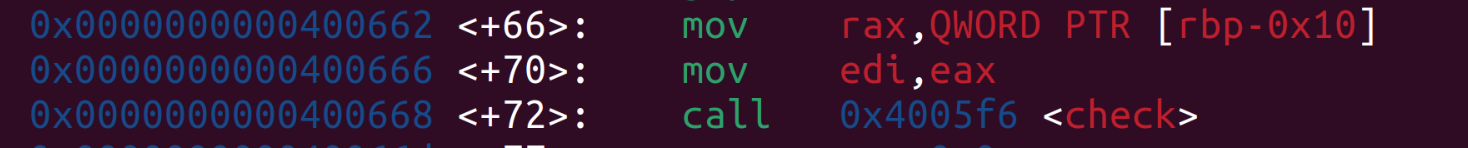
Graphical user interface, text, application

Description automatically generated

Từ source code ta có thể thắng rằng nếu n = 0 sẽ là kết quả



Disas hàm main ta thấy dòng +45 như thế này: biến a kiểu long int 16 bit sẽ được lưu trong thanh ghi rax

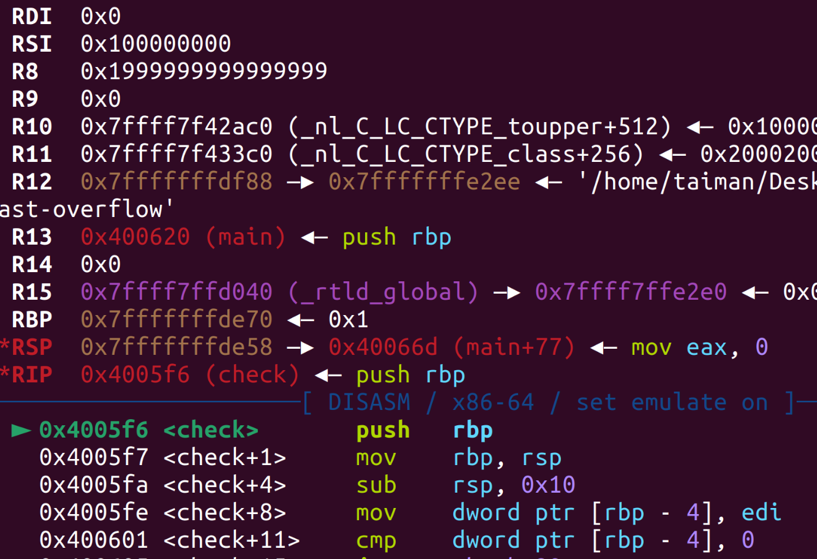


Trước khi vào hàm check, edi sẽ là tham số đầu vào cho hàm check.

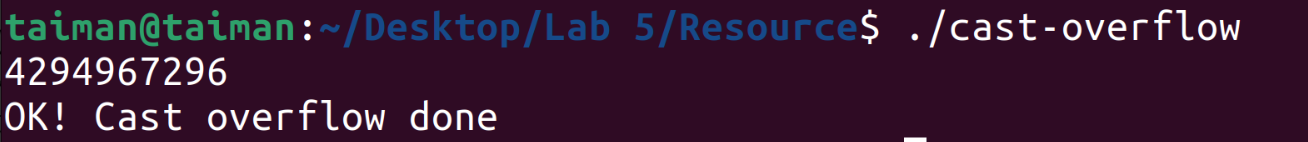
Ta nhập số 4294967296 là 0x100000000

Thì số hex này sẽ được lưu vào thanh ghi rax.

RAX có trọng số lớn hơn EAX 4 bytes nên từ dòng main+70, eax sẽ chỉ lấy 8 bytes( 0x00000000) của rax ( 0x100000000) để làm tham số cho hàm check.



Lúc này edi = 0 được so sánh với 0 ở dòng check +11 sẽ cho ra kết quả OK!



B.2. Tràn ngăn xếp (Stack Overflow)

Yêu cầu 5. Sinh viên khai thác lỗ hổng stack overflow của file thực thi vulnerable, điều hướng chương trình thực thi hàm success. Báo cáo chi tiết các bước thực hiện

Ảnh có chứa bàn

Mô tả được tạo tự động

Disas hàm success ta có địa chỉ của success là 0x0804846b

Graphical user interface, text

Description automatically generated with medium confidence

Ở hàm vulnerable, ta thấy chương trình cấp cho buffer 20 bytes (0x14)

Ở đây ta cần truyền 20 bytes bất kỳ và cộng thêm 4 bytes để ghi đè saved frame pointer và 4 bytes cuối cùng là địa chỉ của hàm success

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Yêu cầu 6. Sinh viên tự tìm hiểu và giải thích ngắn gọn về: procedure linkage table và Global Offset Table trong ELF Linux

Text, letter

Description automatically generated

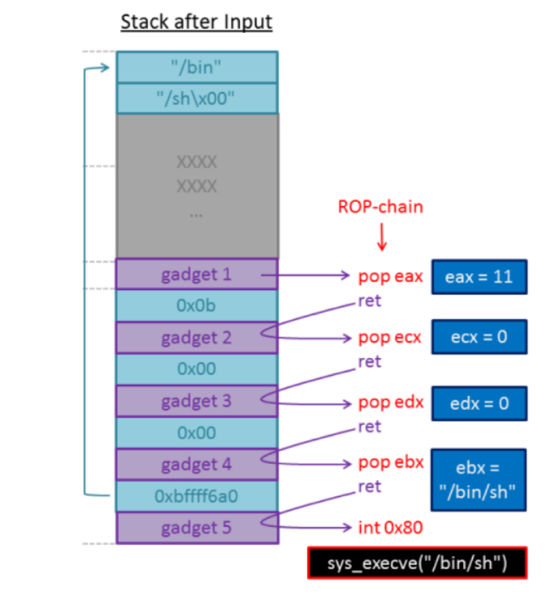
Ref:

https://quantrimang.com/lang-cong-nghe/lien-ket-dong-trong-linux-va-windows-31369

Yêu cầu 7. Sinh viên khai thác lỗ hổng stack overflow trong file rop để mở shell tương tác.

Để có thể khai thác ROP

Ta cần truyền các chuỗi gadget có chứa ret như hình bên dưới



Sử dụng ROPgadget để tìm địa chỉ pop eax ret, pop ecx ret, pop edx ret, int 0x80

Ảnh có chứa văn bản

Mô tả được tạo tự động

địa chỉ của pop eax, ret 0x080bb196

Ảnh có chứa văn bản

Mô tả được tạo tự động

Vì có địa chỉ gộp cả ba cái edx, ecx và ebx nên ta có thể sử dụng địa chỉ này luôn

Ta thấy rằng pop edx, pop ecx, pop ebx, ret nằm trong một địa chỉ 0x0806eb90

Ảnh có chứa văn bản

Mô tả được tạo tự động

Địa chỉ của /bin/sh là 0x080be408

Ảnh có chứa văn bản

Mô tả được tạo tự động

Địa chỉ của hàm exit: 0x08049421

để có thể lấy được system call ta cần truyền cho các thanh ghi những giá trị như sau

Text

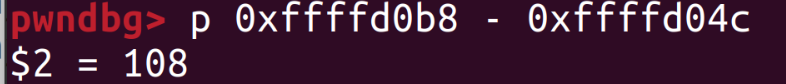
Description automatically generated

Trước khi thực hiện Return-oriented-Programming, ta cần xác định được địa chỉ trả về để ghi đè vị trí này

Text

Description automatically generated

Ta sẽ đặt hàm break ở gets và xác định được ebp của hàm get là 0xffffd030 và buffer được bắt đầu địa chỉ 0xffffd04c



Buffer chứa 108 bytes và ta phải chèn thêm 4 bytes bất kỳ để có thể ghi đè địa chỉ vậy cần padding là 112

Payload:

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated