BÁO CÁO THỰC HÀNH

**Bài thực hành số 03: Nhập môn Pwnable**

**Môn học:** Lập trình an toàn và khai thác lỗ hổng phần mềm

**Lớp:** NT521.N11.ATCL

**THÀNH VIÊN THỰC HIỆN (Nhóm xx):**

|  |  |  |
| --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** |
| 1 | Vũ Hoàng Thạch Thiết | 20521957 |
| 2 | Lê Viết Tài Mẫn | 20521593 |

MỤC LỤC

[**A.** BÁO CÁO CHI TIẾT 2](#_Toc118401900)

[1. Khai thác lỗ hổng buffer overflow cơ bản 2](#_Toc118401901)

[a. Khai thác lỗ hổng buffer overflow khi không sử dụng canary 2](#_Toc118401902)

[b. Cơ chế ngăn lỗ hổng buffer overflow với canary 3](#_Toc118401903)

[2. Khai thác buffer overflow để truyền shellcode 6](#_Toc118401904)

[a. Ví dụ khai thác buffer overflow để truyền code thực thi đơn giản 6](#_Toc118401905)

[b. Viết shellcode 8](#_Toc118401906)

[c. Bài tập khai thác buffer overflow để truyền và thực thi shellcode 10](#_Toc118401907)

# BÁO CÁO CHI TIẾT

## Khai thác lỗ hổng buffer overflow cơ bản

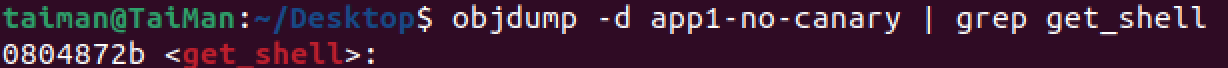
### Khai thác lỗ hổng buffer overflow khi không sử dụng canary

Ảnh có chứa văn bản

Mô tả được tạo tự động

Từ hàm check thấy rằng esp – 0x18 là cấp phát bộ nhớ stack 24 bytes. Old ebp chứa thêm 4 bytes nữa

Nên để ghi đè được địa chỉ trả về thì phải chuyền vào 28 bytes.



Địa chỉ của get\_shell là 080472b vì thế để chạy được hàm get\_shell thì ta phải ghi đè địa chỉ return address với địa chỉ của get\_shell

Ảnh có chứa văn bản

Mô tả được tạo tự động

Truyền 28 bytes bất kì vào stack sau đó truyền địa chỉ của get\_shell để ghi đè return address dưới dạng little endian.

Kết quả là khai thác thành công

Ảnh có chứa văn bản

Mô tả được tạo tự động

### Cơ chế ngăn lỗ hổng buffer overflow với canary

Khi disas hàm main

Ảnh có chứa văn bản

Mô tả được tạo tự động

Từ địa chỉ 0x08048588 có truyền một giá trị từ gs:0x14 vào stack

Dự đoán giá trị canary được ở dưới return address sau stack.

Ebp – 0x8 chứa giá trị canary

Trước khi kết thúc hàm. Giá trị ô nhớ này sẽ được kiểm tra với giá trị ở gs:0x14. Nếu 2 giá trị khác nhau thì đã bị đã xảy ra lỗi buffer overflow.

Địa chỉ kiểm tra giá trị canary trước khi kết thúc hàm

Ảnh có chứa văn bản

Mô tả được tạo tự động

Từ địa chỉ 0x0804860a Trở đi.

Xác định giá trị canary:

Cách 1:

Giá trị canary tại $ebp - x

Text

Description automatically generated

Cách 2:

Ảnh có chứa văn bản

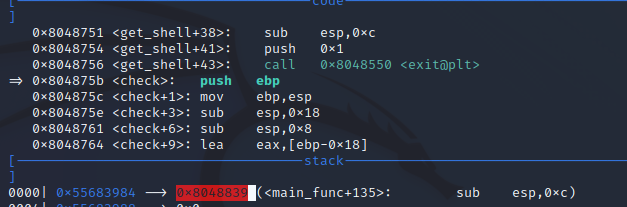
Mô tả được tạo tự động

Giá trị của canary là 0xa9846f00

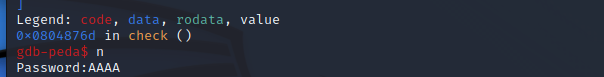
Sau mỗi lần debug thì giá trị của canary là khác nhau.

## Khai thác buffer overflow để truyền shellcode

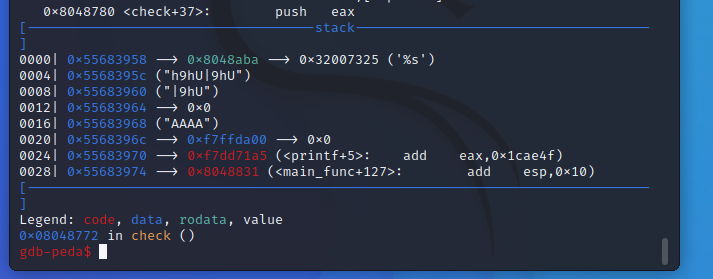
### Ví dụ khai thác buffer overflow để truyền code thực thi đơn giản



+ Trong hàm check() ta đã xác định được địa chỉ của câu lệnh tiếp theo sau khi hàm check() được gọi

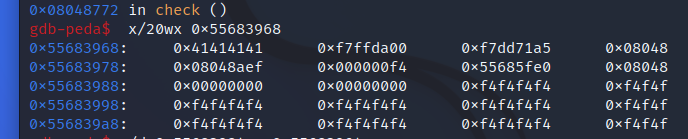


+ Dùng lệnh n cho đến khi nào tới hàm để nhập



+ Stack sau khi nhập

+ Ta thấy chuỗi “AAAA” được lưu vào địa chỉ 0x55683968 sẽ là địa chỉ bắt đầu để lưu trữ biến buf



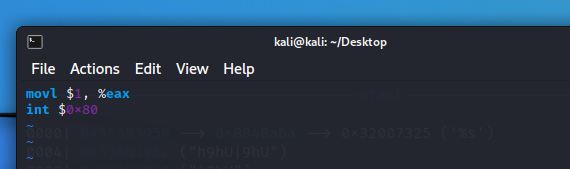
+ Tìm địa chỉ trả về

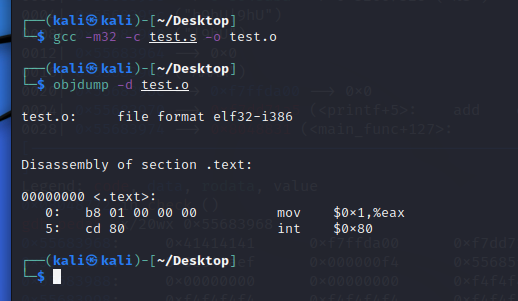
Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Số ký tự nhiều nhất có thể nhập

* Tạo mã thực thi





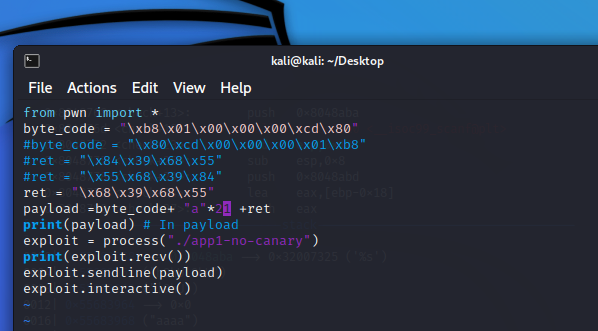
+ Tạo file .o để tạo các byte code thực thị

+ Do file cần khai thác là 32 bits nên có option -32 để tạo byte code ở dạng 32 bits

+ Để thực thi được các byte code chuẩn bị đưa vào stack, ta cần biết được địa chỉ cụ thể của nó ở đâu để có thể điều hướng chương trình đến vị trí đó

+ Tuy nhiên, địa chỉ này chỉ xác định khi chương trình chạy

+ Như đã tìm ở trên thì địa chỉ bắt đầu để lưu chuỗi sẽ là 0x55683968



+ file code thực thi

+ Trong đó:

* Byte\_code: Những byte code thực thi sau khi đã biên dịch
* Ret: Địa chỉ lưu chuỗi đã nhập
* Payload: Cần đảm bảo ở đầu là nững byte\_code thực thi và ở cuối là địa chỉ lưu chuỗi input

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Kết quả: khi chạy file exploit thì chương trình thoát ra ngay chứ không in “End of program” hay “Segmentation Fault”.

### Viết shellcode

section .text

global \_start

\_start:

Push rax # Đẩy tham số vào stack

Xor rdx, rdx # rdx = null là tham số thứ 2 cùa exceve

Xor rsi, rsi # rsi = null là tham số thứ 3 của exceve

Mov rbx, ‘/bin//sh’ # cho rbx = ‘/bin/sh’

Push rbx # push ‘/bin/sh’ vào stack. Rsp sẽ trỏ đến‘/bin/sh’

Push rsp #push giá trị rsp, push địa chỉ ‘/bin/sh’

Pop rdi #rdx sẽ chứa tham số đầu exceve -> ‘/bin/sh’

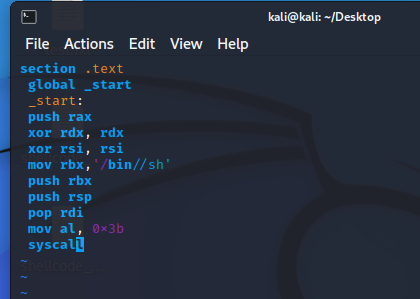
Mov al, 0x3b #syscall numver exceve

Syscall

+ Để biên dịch ta lưu code này vào 1 file .asm .Sau đó dủng lệnh sau để biên dịch

Ảnh có chứa văn bản

Mô tả được tạo tự động



Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Tạo shellcode: là các byte code thực thi vừa được biên dịch

**Ảnh có chứa văn bản

Mô tả được tạo tự động**

+ Dùng công cụ objdump để xem giá trị cũa byte code

+ Ta được chuỗi byte code: \x50\x48\x31\xd2\x48\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x53\x54\

x5f\xb0\x3b\x0f\x05

+ Kiểm tra shellcode

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Viết chương trình c để kiểm tra shellcode

Ảnh có chứa văn bản, đen, màn hình, ảnh chụp màn hình

Mô tả được tạo tự động

+ Biên dịch và chạy file

### Bài tập khai thác buffer overflow để truyền và thực thi shellcode

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Ta thấy mỗi lần chạy chượng trình thì nó sẽ xuất ra một địa chỉ khác nhau và cho ta nhập sau đó thoát chương trình

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Sử dụng gdb để debug chương trình

Ảnh có chứa văn bản, thiết bị, thiết bị đo

Mô tả được tạo tự động

+ Đặt break point tại main+44 là hàm gets()

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, màu bạc

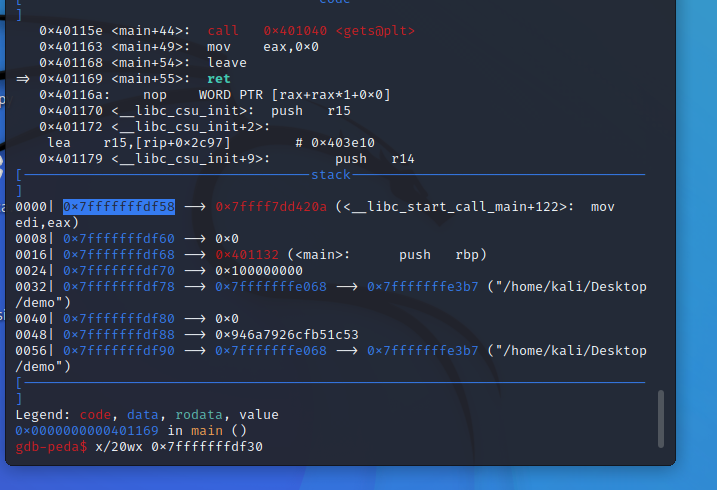
Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Bấm n tới khi nào chương trình cho mình nhập

+ Ta nhập thử 123456 và thấy nó lưu ở địa chỉ 0x7fffffffdf30 đây sẽ là địa chỉ bắt đầu lưu chuỗi



+ Địa chỉ trả về sẽ được lưu trong 0x7fffffffdf58

+ Đỗ dài chuỗi khai thác sẽ là 0x7fffffffdf58 - 0x7fffffffdf30 = 40 byte

+ Shellcode ta sẽ dùng lại của phần trên để gọi syscall exceve("/bin/sh", NULL, NULL)

\x50\x48\x31\xd2\x48\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x53\x54\

x5f\xb0\x3b\x0f\x05

Ảnh có chứa văn bản

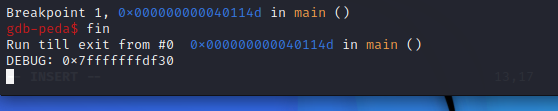
Mô tả được tạo tự động

+ Ta xem mã nguồn của file demo thì thấy hàm printf() sẽ in địa chỉ của buffer sau đó sẽ đưa địa chỉ của buffer vào hàm gets() để lưu chuỗi ta nhập

+ Nên ý tưởng ta sẽ viết code khai thác truyền shellcode sau đó với những byte để trần bộ đệm và cuối cùng là lấy địa chỉ buffer để có thể nhảy đến địa chỉ của buffer để truyền shellcode

Ảnh có chứa văn bản

Mô tả được tạo tự động



+ Kết quả của hàm printf() là in ra địa chỉ của buffer như ta đã biết

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Code khai thác

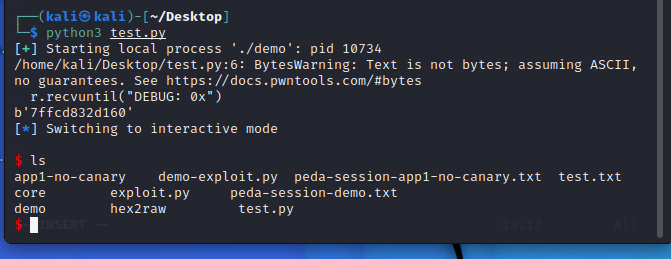
+ Giải thích

* Ta truyền payload là những byte shellcode như bài trên và 16 byte ‘A’ để buffer overflow
* Nhưng chưa biết được địa chỉ lưu chuỗi buffer nên ta cần dùng r.recvuntil để lấy địa chỉ sẽ được in ra theo như hàm printf()
* Sau đó sẽ cộng địa chỉ đó với payload vì đây là chương trình 64 bits và hệ điều hành theo little edian nên có hàm p64(int(address,16))

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Sau đó gửi payload và thực hiện khai thác



+ Kết quả khai thác thành công

TÀI LIỆU THAM KHẢO