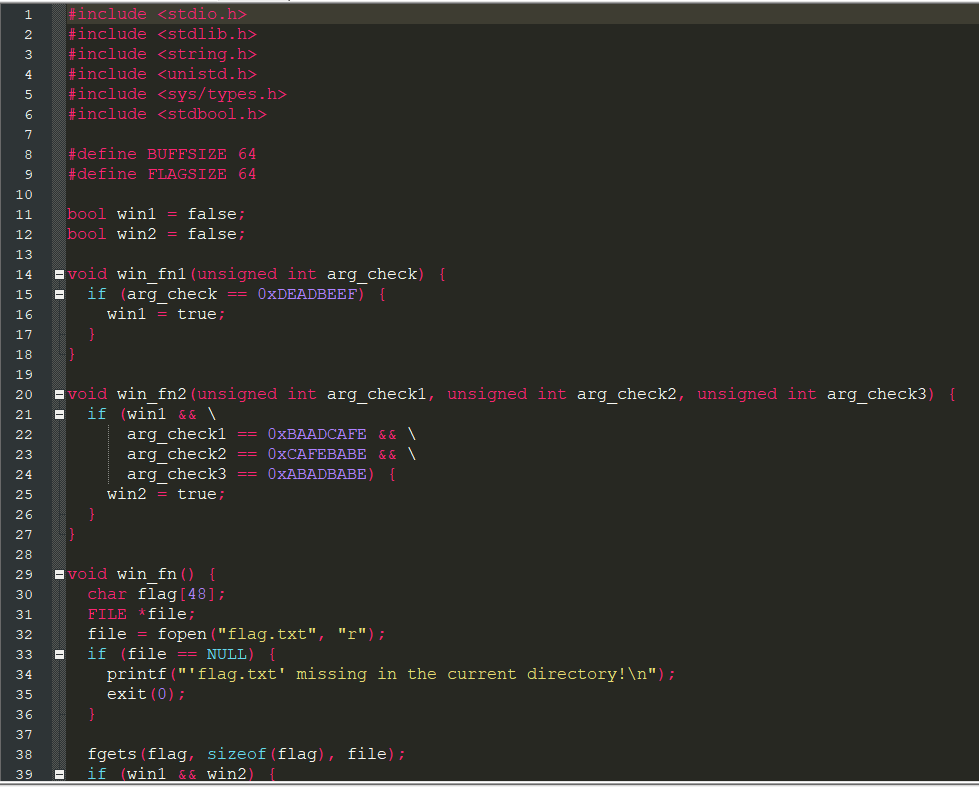
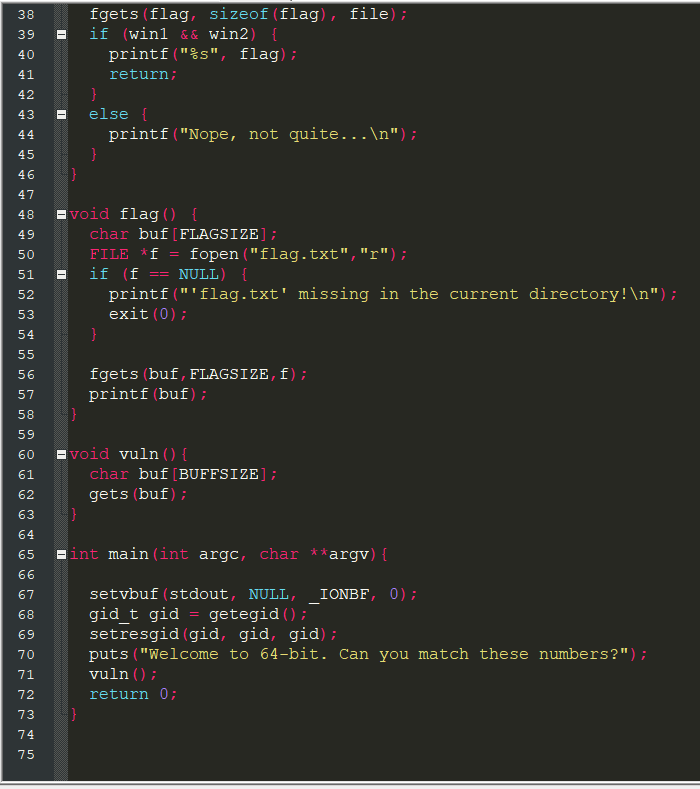
NEWOVERFLOW 2 SOLUTION

1. Đọc hiểu source code:





Trong hàm main, ta thấy chương trình gọi hàm vuln( ). Hàm vuln() nhập cho nhập vào buf. Rồi, thế là hết.

Mục đích của ta là lấy flag, ta thấy có 2 hàm có thể lấy được flag: flag( ) và win\_fn( )

Hàm flag( ) : chỉ in ra thẳng flag

Hàm win\_fn( ) : phải kiểm tra win1 và win2 bằng true thì mới in ra flag. Mà để win1 true thì phải gọi hàm win\_fn1, để win2 true thì phải gọi hàm win\_fn2 sau khi win1 true và nhận 3 tham số:

arg\_check1 == 0xBAADCAFE

arg\_check2 == 0xCAFEBABE

arg\_check3 == 0xABADBABE)

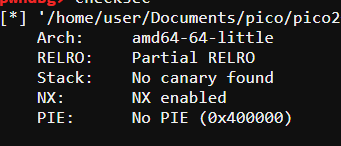
rất phức tạp. Nên chúng ta lựa chọn gọi thẳng hàm flag.

1. Tìm lỗi và hướng khai thác:

Ta thấy trong vuln( ) , hàm gets(buf) nhập vào buf mà không kiểm tra độ dài kí tự mà chúng ta nhập vào. Điều đó gây ra lỗi buffer overflow.

Ta sẽ lợi dụng lỗi trên để làm tràn địa chỉ return của vuln thành địa chỉ hàm flag( ) . Như vậy khi nhập xong buf, ra khỏi vuln( ) thì thay vì quay về main, chương trình sẽ đi đến hàm flag( ) và in ra flag( ) cho chúng ta.

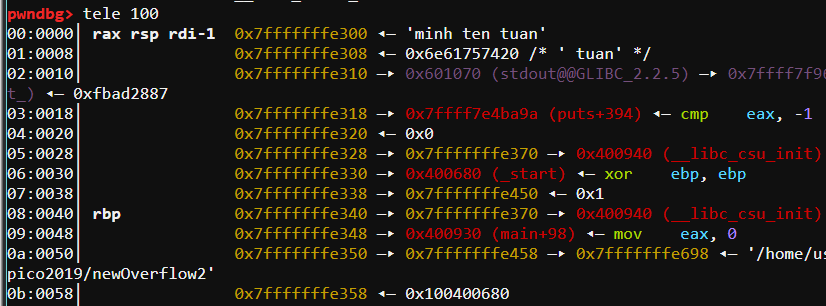
Kiểm tra file binary xem có gì ngăn cản hướng exploit của chúng ta không ?



Ở đây chúng ta không bị chặn canary và cũng no pie. Cho nên rất thoải mái để khai thác.

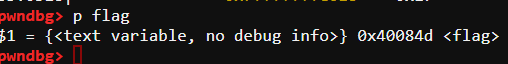
1. Viết code khai thác:

Để làm tràn sang địa chỉ return của vuln thì ta cần biết được offset từ buf cho đến địa chỉ return của vuln.



Mình nhập vào dòng chữ “minh ten tuan” . sau đó show stack ra, thì có thể thấy offset từ buf ( chuỗi mình nhập) đến rbp+8 là: 0x48 bytes.

Tiếp theo, ta cần địa chỉ hàm flag( ) .



Dựa vào ý tưởng trên thì payload của ta có dạng:

Payload = [64 bytes] + [địa chỉ flag-ghi đè lên địa chỉ rbp+8]

Code:

from pwn import \*

s = process('./vuln')

raw\_input('debug')

flag = 0x40084d

ret = 0x000000000040028d

payload = 'a'\*0x48

payload += p64(ret)

payload += p64(flag)

s.sendline(payload)

s.interactive()