**Challenge: ret2libc\_chall**

- ret2libc (return-to-libc) là kỹ thuật lợi dụng lỗ hổng tràn bộ đệm để điều khiển luồng thực thi chương trình bằng cách gọi trực tiếp các hàm trong thư viện libc (thường là system(), exit(), và truyền tham số "/bin/sh").

- Các chế độ bảo vệ ảnh hưởng đến ret2libc:

A screenshot of a black screen

AI-generated content may be incorrect.

- Với file chall:

A screen shot of a computer

AI-generated content may be incorrect.

- No canary -> có thể buffer overflow

- NX ko bật -> ko thể thực thi shellcode trực tiếp trên stack

- PIE: No PIE -> các địa chỉ của file là cố định

- Các chế độ bảo vệ khác không ảnh hưởng

A screenshot of a computer screen

AI-generated content may be incorrect.

- Các địa chỉ của file bin là tĩnh và các địa chỉ của hàm trong thư viện libc là động -> cần leak

**\* Xem pseudo code ở IDA xem, chương trình hoạt động ntn:**

- Có chỉ có 2 hàm chính init() và main():

Mô tả chương trình: Đầu tiên khởi tạo buffer buf[80] sau đó gọi hàm init để khởi tạo các bộ đệm, nhập buf bằng gets và kết thúc chương trình

A screenshot of a computer program

AI-generated content may be incorrect.

**\* Nhận diện ret2libc:**

- Stack có thể bị overflow do khai báo 80 mà nhập tận 120 nhưng không thực thi được shellcode trên stack (NX enabled)

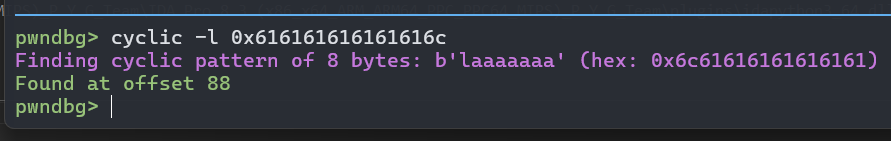
- Có thư viện libc, nhưng không có cơ chế bảo vệ mạnh như PIE hoặc Canary

- Chương trình gọi gets(), scanf(), read() vào biến local

-> Mục tiêu: Mình phải leak đc địa chỉ của libc từ đó tạo shell và chiếm shell.

**\* Phân tích:**

**-** Đầu tiên ta tìm offset của hàm read đến ret



-> 88 byte

\* Khái niệm và GOT và PLT:

- GOT (Global Offset Table): là nơi chứa địa chỉ các hàm trong thư viện libc

- PTL (Procedure Linkege Table): là nơi thực thi những hàm bên trong GOT

\* Mục tiêu của ret2libc: ta sẽ thực thi hàm system(“/bin/sh”) ở trong thư viện libc . Đầu tiên ta phải leak đc địa chỉ của libc sau đó lấy shell

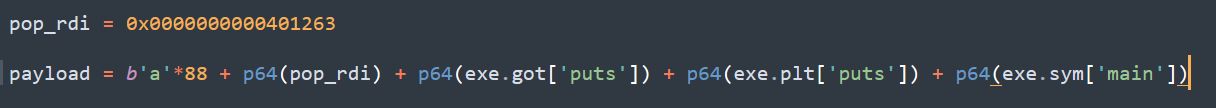
**\*Stage 1: Leak địa chỉ libc:**

- Đầu tiên, ta tìm 1 gadget để điều khiển thanh ghi rdi:



- Ta có thể thiết lập thanh ghi rdi thành put@got, sau khi ta thiết đc argument 1 của hàm puts thì ta chỉ cần thực thi nó thôi

- Sau đó cho chạy lại hàm main và đã có quyền điều khiển chương trình



- Sau đó ta leak địa chỉ của libc bằng cách lấy đủ 6 byte nhận đc + 2 byte null cho đủ 8 byte của hàm u64() rồi in ra địa chỉ dưới dạng hex để kiểm tra

A close up of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

-> có đc libc\_leak 3 byte cuối luôn giống nhau ( lưu đây là địa chỉ libc của máy, mỗi máy khác nhau sẽ chạy 1 hệ điều hành libc khác nhau)

- Nhưng khi kết nối với sever, ta thấy libc\_leak sẽ khác với ở local ( run sever ở folder docker)

A screenshot of a computer

AI-generated content may be incorrect.

- Giờ ta sẽ cần tìm 1 cái libc tương tự với cái libc địa chỉ của sever bằng 1 trang web có sẵn đó là <https://libc.rip/>

- Ta đang địa chỉ của hàm puts nên sẽ điền puts vào symbol, còn address sẽ là địa chỉ libc\_leak ta vừa leak trên sever

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

- Sẽ cho ra kết quả của các libc đều có 3 byte cuối là 3a0 nên ta buộc phải thử từng file libc để tìm ra đúng cái libc mà sever đang chạy thì ta gửi dữ liệu mới đúng

- Để kiểm tra ta sẽ dùng tool pwninit để patch file bin của mình với file libc vừa download về.

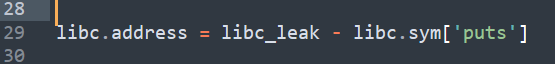
- Ta sẽ thử lần lượt khai thác với các file libc tải trên web về, cái nào mà khai thác được cả trên local với sever đều được thì là file đúng.

- Ta thử với file đầu tiên libc6-amd64\_2.31-0ubuntu9.3\_i386.so.

A screenshot of a computer program

AI-generated content may be incorrect.

- Tiếp theo ta sẽ tính địa chỉ libc base ( là địa chỉ nhỏ nhất của file libc mà ta có được), ta có leak\_base = địa chỉ libc\_leak – địa chỉ của puts trong libc



- Rồi in ra để kiểm tra

A screenshot of a computer

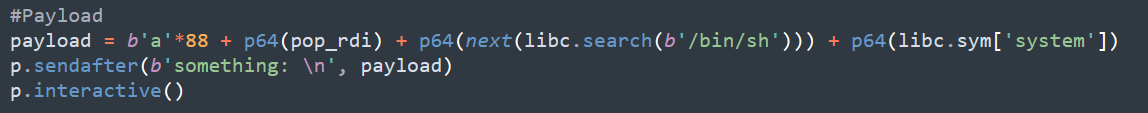
AI-generated content may be incorrect. -> như này là ok

**\*Stage 2: Lấy shell qua hàm system trong libc:**

- Sau khi gửi payload trước chương trình sẽ thực hiện lại hàm main, nên ta có payload mới vẫn sẽ truyền vào 88 byte rác trước sau đó ta truyền pop\_rdi tìm đc trước đó để thiết lập argument1 của system().

- Do libc đã có sẵn địa chỉ /bin/sh rồi ta đã sẽ p64 vào payload luôn bằng lệnh next(libc.search(b'/bin/sh')) ( lệnh này sẽ trả về địa chỉ chuỗi chuỗi /bin/sh trong libc)

- Sau khi thiết lập được argument1 thì ta chỉ cần thực thi hàm system trong libc thôi

- Ta có script:  


- Giờ ta chạy thử solve.py xem chiếm đc shell chưa:

A screen shot of a computer

AI-generated content may be incorrect.

- chưa được nên ta sẽ thử file libc khác trên web

- Tương tự với các file libc khác sẽ có 1 file đúng đó là libc6-amd64\_2.31-0ubuntu9.1\_i386

- Ta sẽ patch lại với file binary rồi nộp thử ở local

A screenshot of a computer

AI-generated content may be incorrect. -> đã chiếm đc shell

- Giờ ta sẽ nộp lên sever với ip: 127.0.0.1 và port: 9993 để lấy cờ

A screen shot of a computer

AI-generated content may be incorrect.