**BABY STACK**

**Kiểm tra file**

**Ảnh có chứa văn bản

Mô tả được tạo tự động**

* file 32 bit, dynamic linked

**Checksec**

**Ảnh có chứa văn bản

Mô tả được tạo tự động**

* NX được bật, như vậy ta sẽ không thể chèn shellcode được. Canary và PIE đều tắt.

**Phân tích**

**Ảnh có chứa văn bản

Mô tả được tạo tự động**

Tại hàm main, chương trình gọi hàm alarm và hàm scanInput, trong đó hàm scanInput chứa một lỗ hổng buffer overflow. Biến buf được khai báo với 40 bytes, tuy nhiên hàm read lại đọc 0x40 (64) bytes.

Ta lưu ý các điểm sau:

* Chương trình chỉ sử dụng 2 hàm: read và alarm, cho nên việc gom các gadgets để khai thác ROP là không khả thi
* Return to libc truyền thống cũng bất khả thi vì: không có bất kì hàm nào để leak địa chỉ của libc, thứ hai là cũng không được cung cấp bất kì libc nào.
* Khai thác **return to dl-resolve()**

**Return to dl-resolve()**

Mục tiêu là sẽ tạo fake 3 cấu trúc JMPREL, STRTAB, SYMTAB, và sau đó gọi dl\_runtime\_resolve() để phân giải các lệnh mà mình muốn, ví dụ như là system.

Ta sẽ sử dụng vùng bss để lưu trữ các cấu trúc fake này.

Luồng hoạt động của ta như sau: vì hàm read đọc vào 64 bytes (40 bytes của biến buf) và chỉ đè được 24 bytes (không đủ cho việc chúng ta). Cho nên chúng ta sẽ chia làm 3 lần khai thác:

* Lần đầu tiên sẽ tiến hành ghi đè địa chỉ trả về thành hàm **read**, và truyền các tham số thích hợp.
* Lần thứ 2 sẽ tiến hành ghi để tạo các fake cấu trúc để đánh lừa trình phân giải địa chỉ.
* Sau khi tạo xong thì ta sẽ gọi resolve, truyền offset để nhảy tới fake mà ta đã tạo, cuối cùng là truyền tham số cho hàm system.

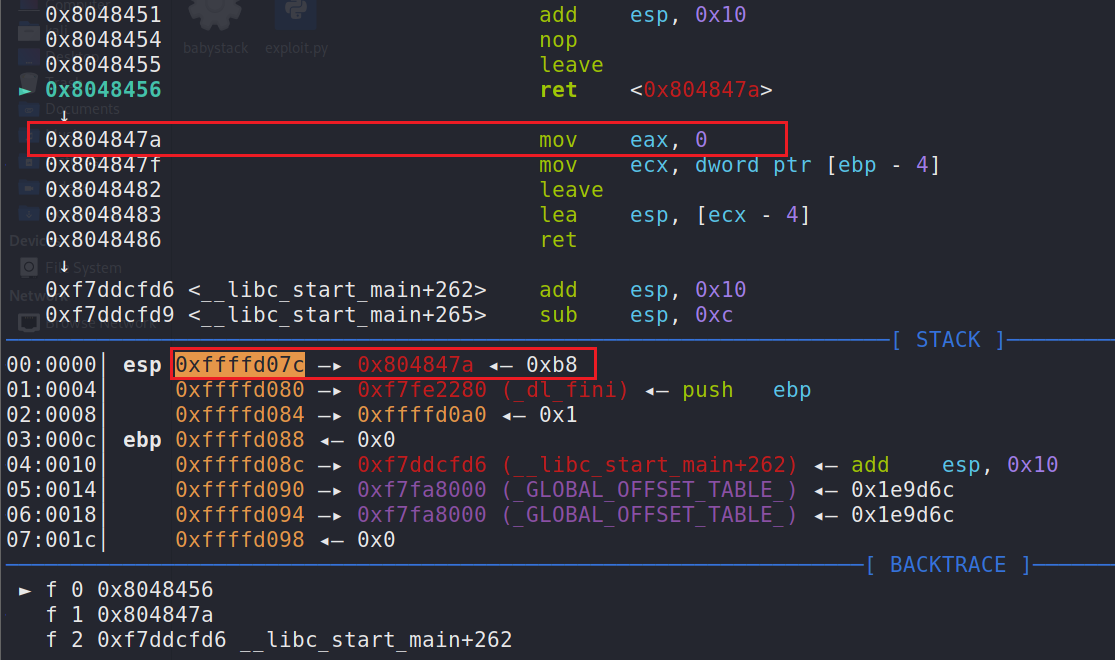
Tính offset để ghi đè lên địa chỉ trả về:

Biến buf lưu tại địa chỉ 0xffffd050

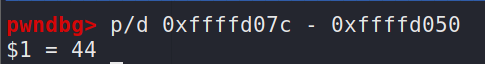
Ảnh có chứa văn bản

Mô tả được tạo tự động

Địa chỉ trả về của hàm scanInput lưu tại địa chỉ: 0xffffd07c



Như vậy offset để buffer overflow là: 44 bytes



Địa chỉ symtab của ta phải bắt đầu tại bss + 0xc (để khi tính R\_SYM) cho tròn

**Đầu tiên là JMPREL:**

Mình có thể chọn bất kì hàm nào có trong JMPREL gốc của chương để giả mạo.

Ở đây mình sẽ lấy hàm alarm => r\_offset = elf.sym[‘alarm’]

Tính r\_info của hàm alarm: R\_SYM= (bss+0xc – symtab) / 16 => r\_info = (R\_SYM << 8) + 0x7 (bởi vì offset tận cùng bằng 7)

**SYMTAB**

Ở symtab ta cần tính **st\_name:** ở đây ta sẽ lấy địa chỉ tại fake strtab trừ đi cho strtab thật của chương trình để tính offset: (bss+28) – strtab

Các biến còn lại ta cho là 0x00 (bytes tuỳ ý)

**STRTAB**

Ta sẽ thêm chuỗi: “system\x00” và tham số của hàm này “/bin/sh\x00”