BÁO CÁO THỰC HÀNH

**Môn học: Lập trình an toàn và khai thác lỗ hổng phần mềm**

**Kỳ báo cáo: Buổi 03 (Session 03)**

**Tên chủ đề: Bài tập An toàn thông tin**

*GVHD: Nghi Hoàng Khoa*

*Ngày báo cáo: 08/11/2021*

**Nhóm: SSNI-502**

1. **THÔNG TIN CHUNG:**

*(Liệt kê tất cả các thành viên trong nhóm)*

Lớp: NT521.M11.ATCL

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** | **Email** |
| 1 | Nguyễn Chí Toàn | 19522361 | 19522361@gm.uit.edu.vn |
| 2 | Phùng Ngọc Huyến | 19521646 | 19521646@gm.uit.edu.vn |
| 3 | Lê Công Trình | 19522404 | 19522404@gm.uit.edu.vn |
| 4 | Lê Hà Quang Thịnh | 19522277 | 19522277@gm.uit.edu.vn |

1. **NỘI DUNG THỰC HIỆN:**

|  |  |  |
| --- | --- | --- |
| **STT** | **Công việc** | **Kết quả tự đánh giá** |
| 1 | Kịch bản 01 | 100% |
| 2 | Kịch bản 02 | 100% |
| 3 | Kịch bản 03 | 100% |
| 4 | Kịch bản 04 | 90% |
| 5 | ROPchain | 100% |
| 6 | Start - pwnable | 100% |

**Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.**

BÁO CÁO CHI TIẾT

## Kịch bản 01

## Kịch bản 02

## Kịch bản 03

## Kịch bản 04

## ROPchain

A screenshot of a computer

Description automatically generated with medium confidence

- Tại hàm main của ROPchain ta thấy được mục tiêu ở đây sẽ khai thác vào hàm vunl

Graphical user interface, text, application, email

Description automatically generated

- Khi dùng IDA ta thấy buffer ở hàm vunl có độ dài buffer là 0x88

Graphical user interface, text

Description automatically generated

Như đã biết thì nếu ta muốn ghi đè luôn của return address thì ta cần thêm 0x4 nữa nên ta sẽ cần 0x88 + 0x4 =0x8c đổi ra Dec sẽ là 140

- Bước tiếp theo ta sử dụng ROPgadget để tìm các gadget để exploit.

Text

Description automatically generated

- eax

Text

Description automatically generated

Ta sẽ chọn 0x080a89e6 # pop eax; ret

- edx, ecx, ebx

Text

Description automatically generated

- Tiếp theo sẽ là đến string /bin/sh

Graphical user interface

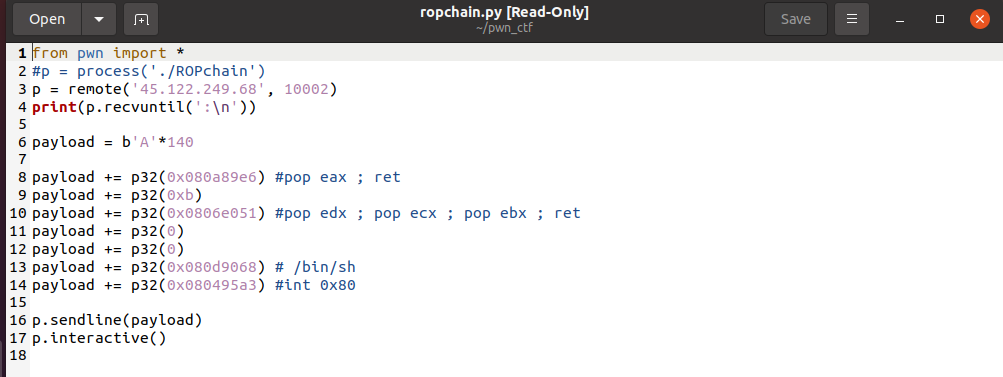
Description automatically generated with low confidence

- Và cuối cùng sẽ là đến int

A picture containing diagram

Description automatically generated

- Payload



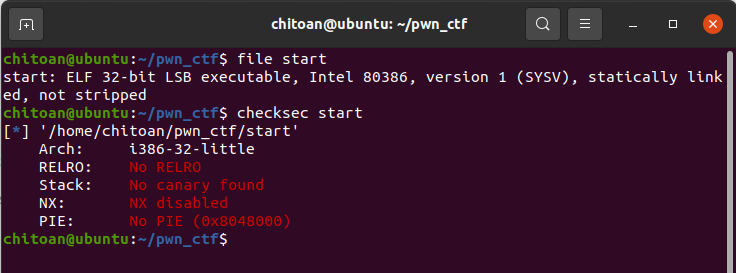
- Kết quả:

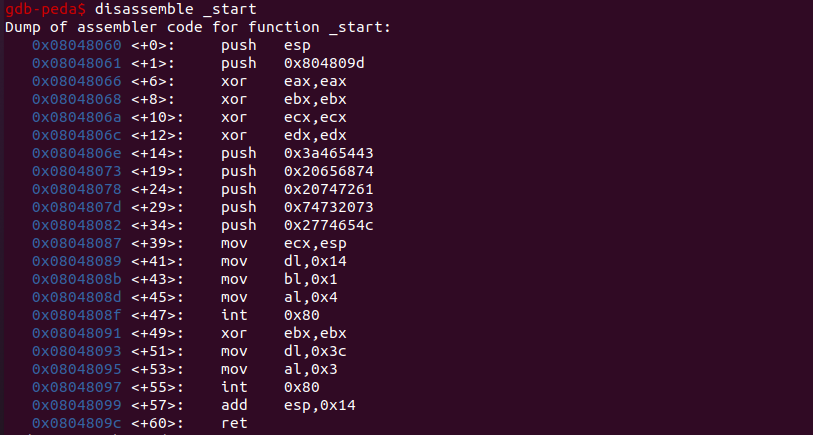
Text

Description automatically generated

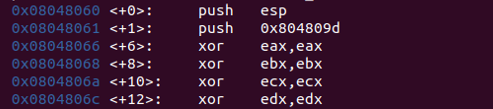
## Start – pwnable

- file & checksec start

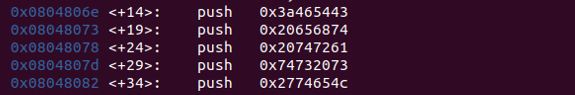




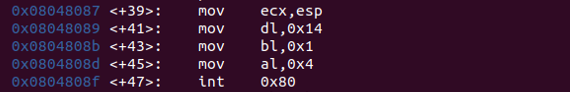
- Dùng gdb để disassemble \_start để phân tích luồng chạy của file.



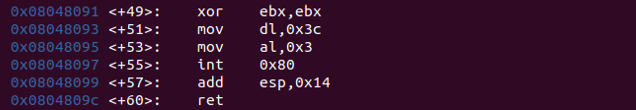
- Đoạn này push esp và địa chỉ hàm thoát vào stack, khởi tạo giá trị cho các thanh ghi.



- 5 lệnh push này để push đoạn “Let’s start the CTF:” vào stack.



- Ngắt 0x80 đầu tiên, eax = 4, chương trình gọi lệnh sys\_write() để print vào stdout (ebx = 1) 20 kí tự (edx = 0x14) tại địa chỉ esp (ecx = esp), mục đích là in dòng ‘Let’s start the CTF:’ ra màn hình.

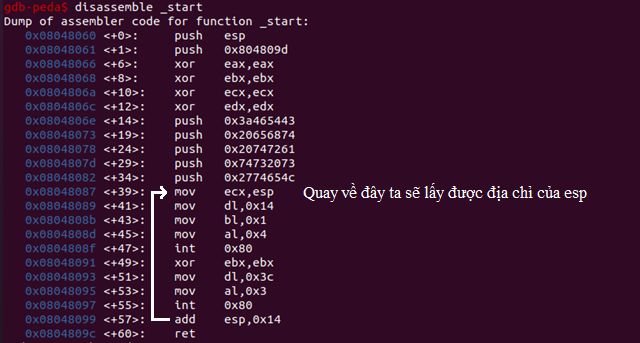


- Ngắt 0x80 thứ hai, eax = 3, chương trình gọi lệnh sys\_read() đọc tối đa 60 kí tự (edx = 0x3c) từ stdin (ebx = 0), lưu vào stack tại vị trí esp (ecx = esp).

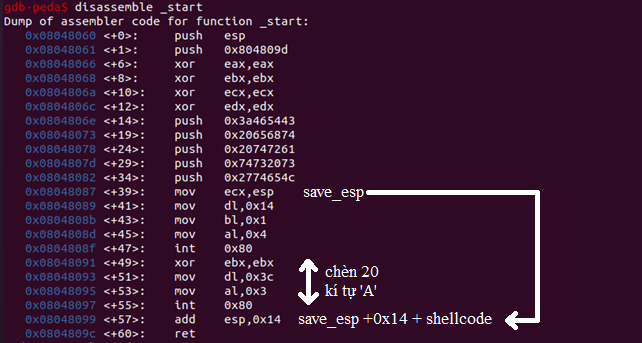
- Tăng giá trị esp lên 20 và return.

Ở đoạn này chương trình cho nhập vào 0x3c tức 60 kí tự vào stack nhưng sau đó chỉ tăng esp lên 0x14 tức 20 và return như vậy buffer overflow sẽ xảy ra. Để khai thác lỗi này cần 2 lần chèn payload:

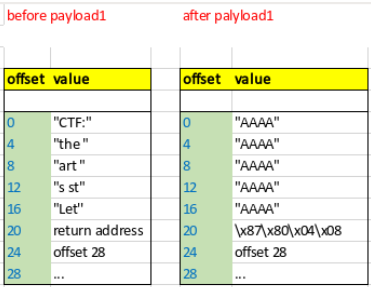
+ Lần 1: chèn payload1 làm tràn 20 bytes và điều khiển chương trình về 0x08048087 để save địa chỉ của esp, lưu thành 1 biến save\_esp để dùng ở payload2.



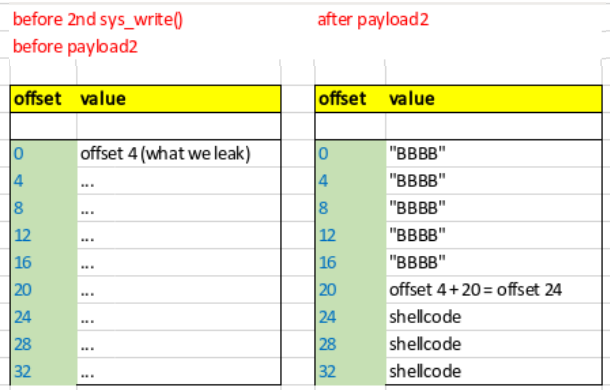
+ Lần 2: chèn payload 2 làm tràn 20 bytes và lấy save\_esp + 0x14 + shellcode, như vậy shellcode sẽ được thực thi mà không phải return về hàm thoát. (không cần quan tâm phần print 20 bytes ở bước này). Lưu ý shellcode cần đủ ngắn vì chỉ được tối đa 60 kí tự



- Stack trước và sau khi chèn payload1



- Stack trước và sau khi chèn payload2



- Code payload trong file start.py

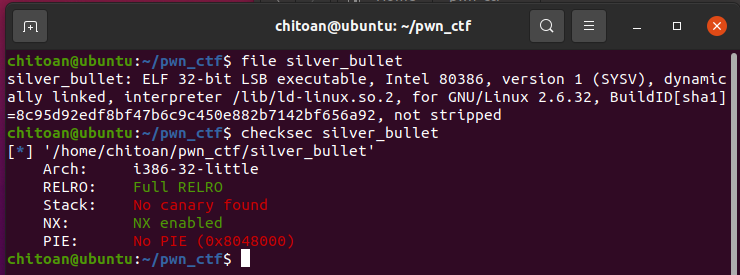


- Kết quả: file flag nằm trong home/start



## Silver\_bullet

- file & checksec silver\_bullet

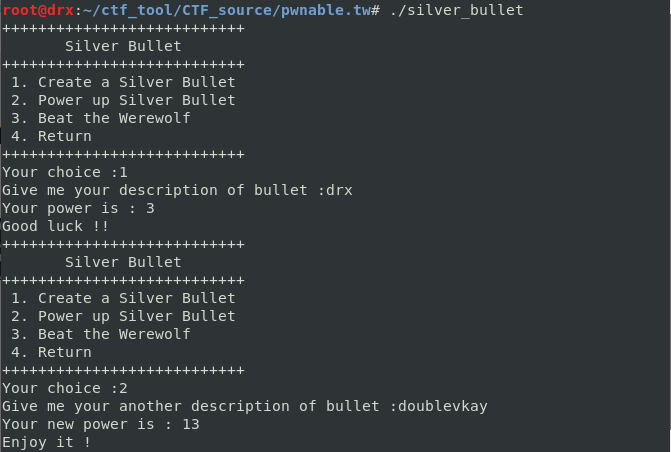


- run silver\_bullet, ta thấy có 3 chức năng tương ứng 3 hàm:

+ Tạo viên đạn

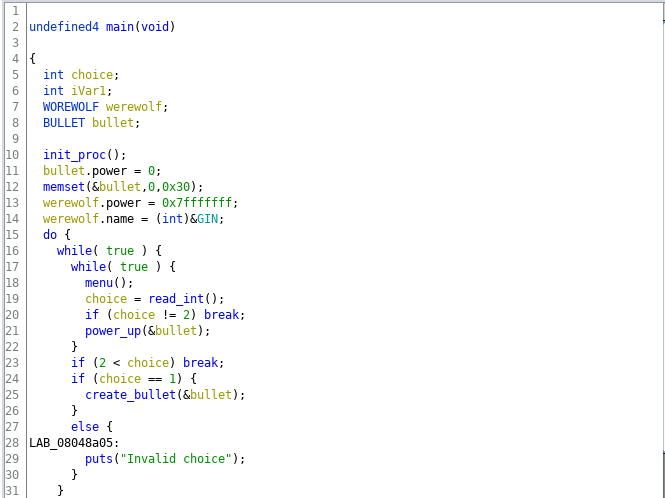
+ Tăng sức mạnh cho đạn

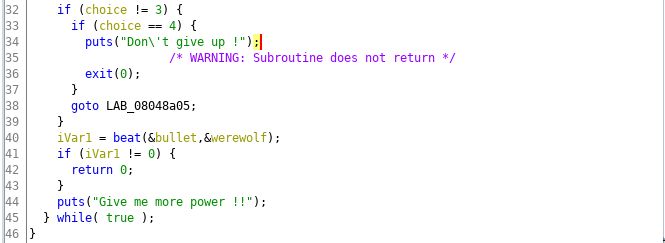
+ Bắn sói



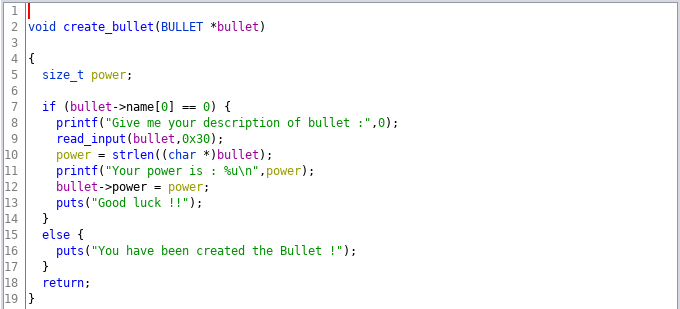
- Xem file silver\_bullet dưới dạng mã giả

+ Hàm main

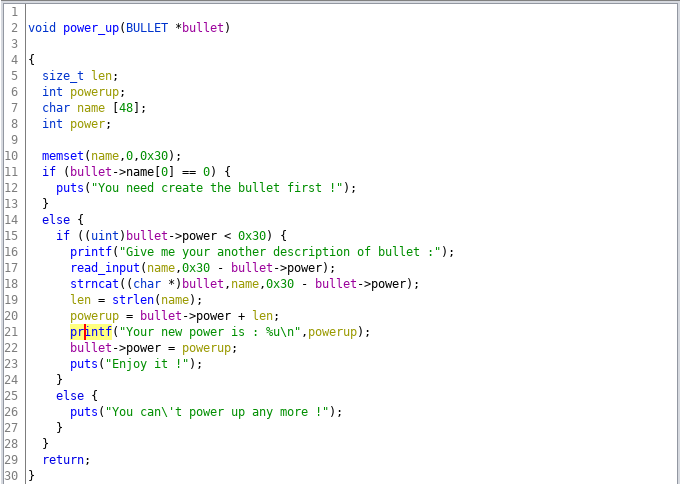




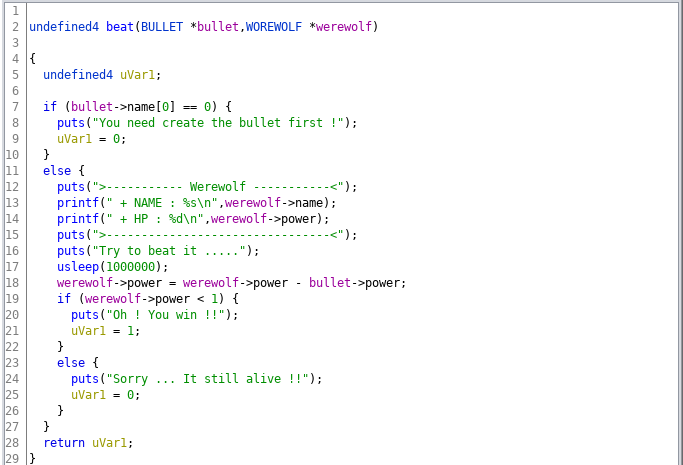
+ Hàm create\_bullet()



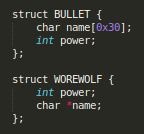
+ Hàm power\_up()



**+** Hàm beat()



+ struct BULLET & WOREWOLF



Phân tích luồng thực thi của file

- Ban đầu khởi tạo một con sói WOREWOLF \*werewolf có tên là werewolf.name = "GIN" và sức mạnh là werewolf.power = 0x7fffffff. Mục đích của trò chơi là ta sẽ tạo 1 viên đạn BULLET \*bullet và đem đi bắn con sói. Bắn đến khi nào sói hết máu thì chiến thắng.

- Chương trình có 3 chức năng chính tương ứng với 3 hàm:  
+ **create\_bullet()**: tạo một viên đạn BULLET \*bullet có tên là bullet.name dài tối đa 0x30 kí tự do người dùng nhập vào, set power của bullet là bullet.power = len(bullet.name).  
**+ power\_up()**: ta có thể tăng sức mạnh cho bullet bằng cách nối dài thêm name, bullet.power sẽ được cập nhật lại là chiều dài của tên mới. Chiều dài tối đa của bullet.name vẫn là 0x30  
+ **beat()**: tiến hành bắn sói, sức mạnh của sói sẽ giảm đúng bằng bullet.power. Nếu sói hết máu thì thông báo chiến thắng.

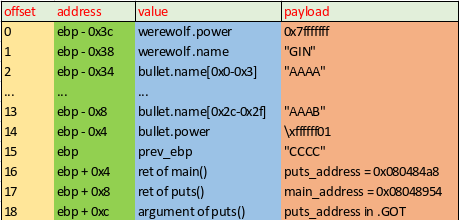
- Sau khi tìm hiểu phát hiện ra có lỗi ở hàm power\_up() cụ thể là lệnh strncat():  
 **strncat(bullet.name, name, 0x30-bullet.power)**

**-** Hàm power\_up() có tác dụng nối chuỗi name mà người dùng nhập với chuỗi bullet.name, với số kí tự được copy tối đa là 0x30-bullet.power. Hàm strncat() sẽ copy chuỗi name vào chuỗi bullet.name, sau đó sẽ thêm 1 ký tự NULL vào cuối chuỗi. Do đó kí tự NULL này sẽ overwrite vào biến nằm ngay sau bullet.name trong stack và đây cũng là vị trí của bullet.power

⭢  Chuỗi name mà ta nhập vào sẽ overwrite từ vị trí của bullet.power, do đó ta có thể overwrite return address.

- File được set NX enabled nên không thể thực hiện ret2shellcode, đề bài cho file libc\_32.so.6  
⭢ Dùng kĩ thuật ret2libc để gọi hàm system() với tham số là “bin/sh”

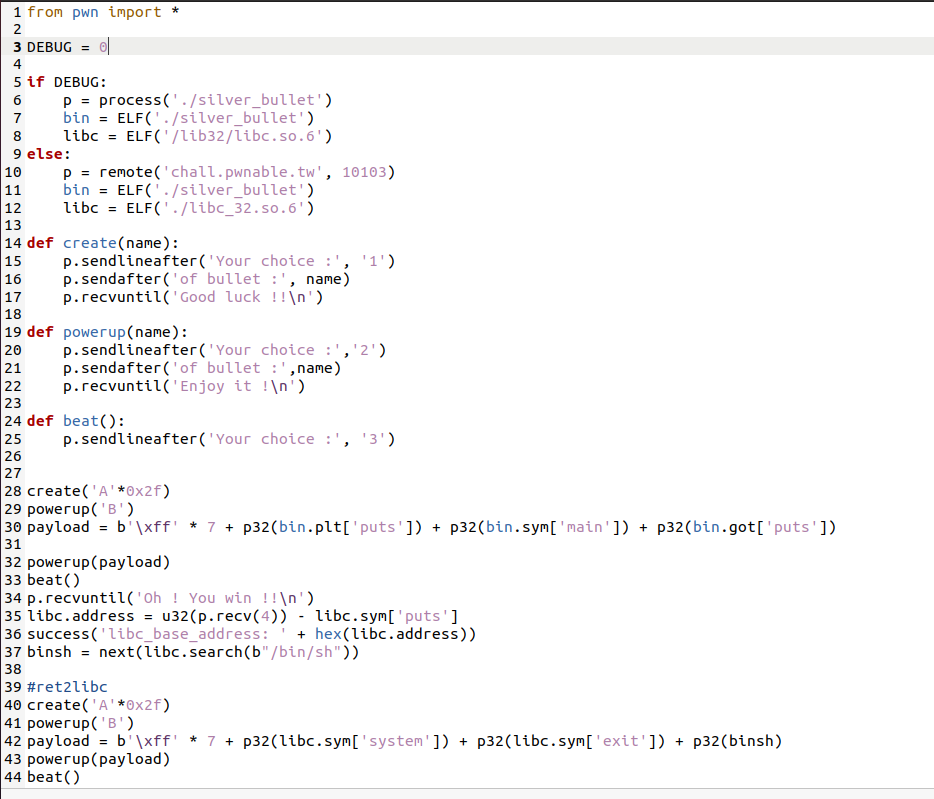
- Do có sử dụng thư viện liên kết động trước hết ta cần tìm được địa chỉ của thư viện libc sau khi load ⭢ Overwrite return address thành hàm puts() với tham số là 1 hàm nào đó của libc (chẳng hạn như hàm puts), mục đích là để chương trình in ra địa chỉ của hàm puts sau khi được load vào vùng .GOT. Hình ảnh minh họa:

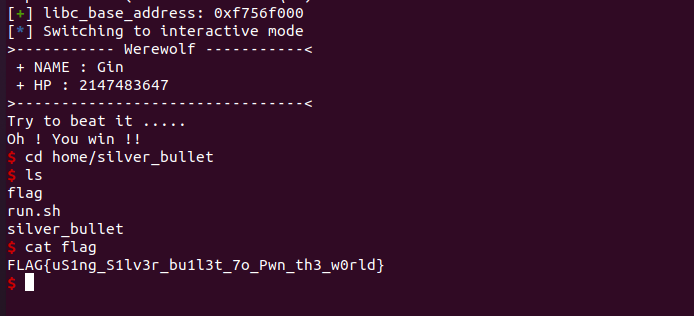


- Tính địa chỉ của libc bằng cách:

**(libc\_base\_address) = (puts\_address\_leaked) - (offset\_of\_puts)**

**⭢** Từ đây có thể tính được địa chỉ của system() và chuỗi “bin/sh”

- Payload:  


- Kết quả:  


**HẾT**