6

Lab

Buffer Overflow Attack (Buffer Bomb) Part 2

Thực hành Lập trình Hệ thống

Lưu hành nội bộ

A. TỔNG QUAN

A.1 Mục tiêu

Trong bài lab này, sinh viên sẽ vận dụng những kiến thức về cơ chế của stack trong bộ xử lý IA32, nhận biết code có lỗ hổng buffer overflow trong một file thực thi 32-bit để khai thác lỗ hổng này, từ đó làm thay đổi cách hoạt động của chương trình theo một số mục đích nhất định. Bài thực hành được thực hiện trong 2 buổi:

- Buổi 5: Level cơ bản 0 – 1.
- Buổi 6: Level nâng cao 2 – 3.

A.2 Môi trường

- Môi trường debug file thực thi Linux 32-bit:

  + Cách 1: Remote debug từ máy Windows cài IDA Pro sang máy Linux chạy file thực thi (xem hướng dẫn ở Lab 4 hoặc URL đính kèm).

  + Cách 2: Sử dụng GDB trên máy Linux.

- Các file source của bài lab:

  1. **bufbomb**: file thực thi Linux 32-bit chứa lỗ hổng buffer overflow cần khai thác.
  2. **makecookie**, **hex2raw**: một số file hỗ trợ.

A.3 Liên quan

- Có kiến thức về cách mà hệ thống phân vùng bộ nhớ.
- Có kỹ năng sử dụng một số công cụ để debug như **IDA, gdb**.

B. NHẮC LẠI VỀ BUFFER BOMB LAB

B.1 Chương trình bufbomb

bufbomb là file thực thi dạng command line có lỗ hổng buffer overflow. Chương trình này chạy dưới dạng command line và sẽ nhận tham số đầu vào là một chuỗi. Khi chạy, **bufbomb** đi kèm nhiều option như sau:

-u userid Thực thi **bufbomb** của một user nhất định. Khi thực hiện bài lab ***luôn phải cung cấp tham số*** này.

-h In danh sách các option có thể dùng với bufbomb.

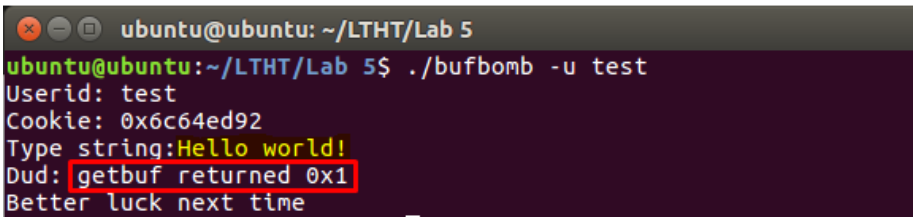
Trong hoạt động của **bufbomb** nhận một chuỗi đầu vào với hàm **getbuf** như sau:

```
1  /* Buffer size for getbuf */
2  #define NORMAL_BUFFER_SIZE 32
3  int getbuf()
4  {
5        char buf[NORMAL_BUFFER_SIZE];
6        Gets(buf);
7        return 1;
8  }
```

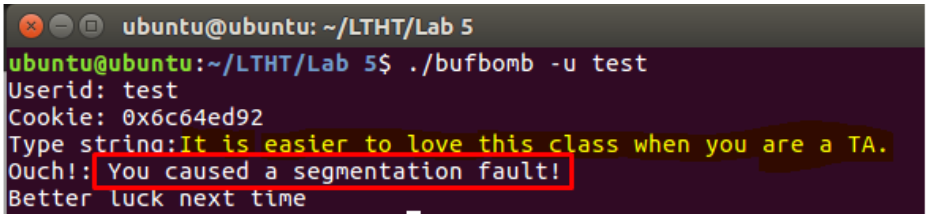
Hàm **Gets** giống với thư viện hàm chuẩn **gets** – đọc một chuỗi đầu vào và lưu nó ở một vị trí đích xác định. Trong đoạn code phía trên, có thể thấy vị trí lưu này là một mảng buf có kích thước 32 ký tự.

Vấn đề ở đây là, khi lưu chuỗi, hàm **Gets** không có cơ chế xác định xem **buf** có đủ lớn để lưu cả chuỗi đầu vào hay không. Nó chỉ đơn giản sao chép cả chuỗi đầu vào vào vị trí đích đó, do đó dữ liệu nhập vào có trường hợp sẽ vượt khỏi vùng nhớ được cấp trước đó.

Với **bufbomb**, trong trường hợp nhập vào một chuỗi có độ dài không vượt quá 31 ký tự, **getbuf** hoạt động bình thường và sẽ trả về 1, như ví dụ thực thi ở dưới:



Tuy nhiên, thử nhập một chuỗi dài hơn 31 ký tự, có thể xảy ra lỗi:



Khi tràn bộ nhớ thường khiến chương trình bị gián đoạn, dẫn đến lỗi truy xuất bộ nhớ.

Trong bài thực hành này, đối tượng cần khai thác chủ yếu là **getbuf** và stack của nó. **Nhiệm vụ của sinh viên là truyền vào cho chương trình bufbomb (hay cho getbuf) các chuỗi có *độ dài và nội dung* phù hợp để nó làm một số công việc thú vị. Ta gọi đó là những chuỗi "exploit" – khai thác.**

Lưu ý: mỗi nhóm sinh viên sẽ có riêng 1 phiên bản file bufbomb

B.2 Một số file hỗ trợ

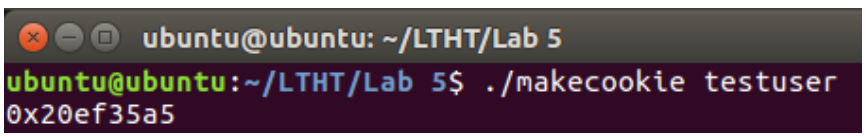
Bên cạnh file chính là **bufbomb,** thư mục source của Buffer Bomb lab gồm một số file hỗ trợ quá trình thực hiện bài thực hành:

- **makecookie**

File này tạo một cookie dựa trên userid được cung cấp. Cookie được tạo ra là một chuỗi ***8 số hexan*** duy nhất với userid. Cookie này cần được dùng trong một số level của bài lab.

Cookie có thể được tạo như sau:

```
$ ./makecookie <userid>
```



- **hex2raw**

hex2raw sẽ giúp chuyển những byte giá trị không tuân theo bảng mã ASCII (các byte không gõ được từ bàn phím) sang chuỗi có thể truyền làm input cho file **bufbomb**. **hex2raw** nhận đầu vào là chuỗi dạng hexan, mỗi byte được biểu diễn bởi ***2 số hexan*** và các byte cách nhau bởi khoảng trắng (khoảng trống hoặc xuống dòng). Ví dụ chuỗi các byte: **00 0C 12 3B 4C**

Cách dùng: soạn sẵn giá trị của các byte trong một file text với đúng định dạng yêu cầu sau đó truyền vào cho **hex2raw** bằng lệnh sau:

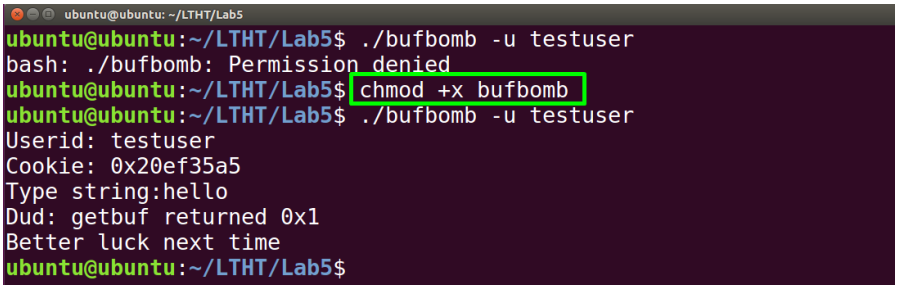
```
$ ./hex2raw < <file>
```

Hoặc

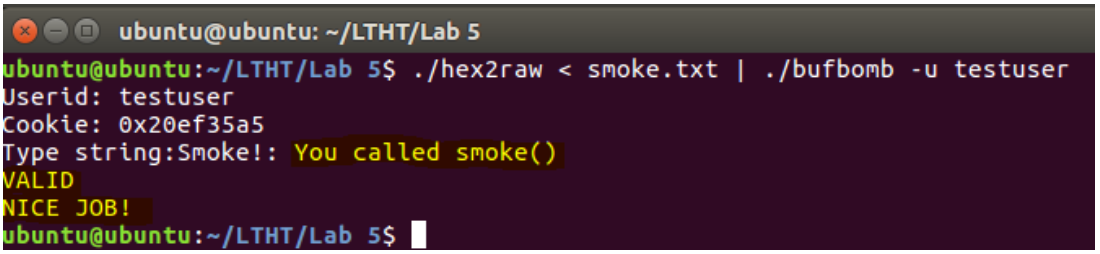
```
$ cat <file> | ./hex2raw
```

B.3 Một số lưu ý

- Các lưu ý khi tạo các byte của chuỗi exploit – chuỗi input cho bufbomb:
  1. Chuỗi exploit ***không được*** chứa byte hexan **0A** ở bất kỳ vị trí trung gian nào, vì đây là mã ASCII dành cho ký tự xuống dòng ('\n'). Khi **Gets** gặp byte này, nó sẽ giả định là người dùng muốn kết thúc chuỗi.
  2. **hex2raw** nhận các giá trị hexan 2 chữ số được phân cách bởi khoảng trắng. Do đó nếu sinh viên muốn tạo một byte có giá trị là 0, cần ghi rõ là 00.
  3. Cần để ý đến byte ordering trong Linux là Little Endian khi cần truyền cho hex2raw các giá trị lớn hơn 1 byte. Ví dụ để truyền 1 word 4 bytes **0xDEADBEEF**, cần truyền **EF BE AD DE** (đổi vị trí các byte) cho **hex2raw**.
- Khi thực thi các file **bufbomb**, **hex2raw** hay **makecookie**, nếu gặp lỗi về **Permission denied**, sinh viên cần cấp quyền thực thi các file.



- Khi sinh viên đã giải quyết đúng một trong các level, ví dụ level 0 sẽ có thông báo:



C. Các bước khai thác file bufbomb

Bài lab gồm 4 cấp độ từ 0 – 3 với mức độ từ dễ đến khó tăng dần, tập trung khai thác lỗ hổng buffer overflow có trong **bufbomb** ở hàm **Gets**. Sinh viên thực hiện các bước sau:

Bước 1. Chọn userid và tạo cookie tương ứng

Sinh viên sử dụng một userid trong bài thực hành và khi chạy file **bufbomb** luôn luôn phải truyền vào tham số **-u <userid>**.

Bắt buộc: *userid được tạo từ **4 số cuối** MSSV của 2 sinh viên trong 1 nhóm.*

Ví dụ: Với 2 MSSV 19520260 và 19520143, ta có userid 02600143.

Xem cookie tương ứng với userid bằng cách chạy chương trình **makecookie**, ghi nhớ giá trị này để sử dụng về sau. Giá trị cookie này là cơ sở để đánh giá của một số level.

Bước 2. Phân tích file bufbomb và Xác định chuỗi exploit cho từng level

Có 2 bước cần thực hiện để xác định chuỗi exploit:

Bước 2.1. Xác định độ dài chuỗi exploit

Phụ thuộc vào:

- Độ dài buffer được cấp phát: chuỗi exploit ít nhất phải có độ dài lớn hơn không gian dành cho buffer để làm tràn được buffer.
- Khoảng cách giữa ô nhớ cần ghi đè so với buffer trong stack: ví dụ ghi đè để thay đổi địa chỉ trả về, giá trị biến,…

Bước 2.2. Xác định nội dung chuỗi input

Chuỗi exploit đã xác định được độ dài ở bước trên sẽ được điền nội dung với những giá trị byte phù hợp để thực hiện đúng ý định, có 2 mức độ:

- Thay đổi giá trị vùng nhớ lân cận (level 0, 1).
- Truyền vào và thực hiện một số câu lệnh nhất định (level 2, 3).

Bước 3. Thực hiện truyền chuỗi exploit vào bufbomb

Sinh viên viết các chuỗi exploit dưới dạng các byte và sử dụng **hex2raw** để truyền cho **bufbomb**. Giả sử chuỗi exploit dưới dạng các cặp số hexan cách nhau bằng khoảng trắng trong file **exploit.txt** như bên dưới.

```
00 01 02 03
00 00 00 00
00 01 02 03
00 00 00 00
00 01 02 03
00 00 00 00
```

Sinh viên có thể truyền chuỗi raw cho **bufbomb** bằng cách lệnh sau:

```
$ ./hex2raw < exploit.txt | ./bufbomb -u testuser
```

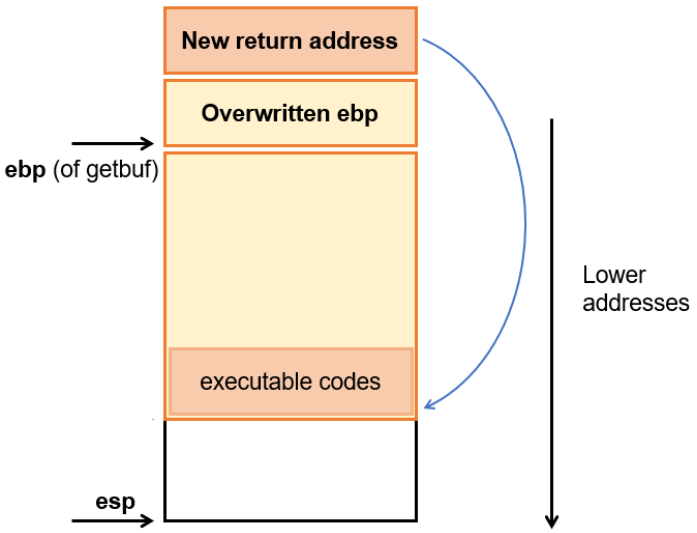
D. CÁC LEVEL NÂNG CAO CỦA BUFFER BOMB LAB

D.1 Sử dụng buffer overflow để chèn và thực thi mã độc

Một dạng phức tạp hơn của tấn công buffer overflow là chuỗi exploit là một chuỗi chứa các byte code có thể thực thi được, được chèn vào stack để thực thi. Khi đó, ***chuỗi exploit sẽ thay đổi địa chỉ trả về để trỏ về vị trí của những câu lệnh này trên stack***. Khi hàm được gọi (trong trường hợp này là **getbuf**) thực thi câu lệnh **ret**, chương trình sẽ nhảy đến vị trí lưu mã thực thi đã chèn vào để thực thi thay vì quay về hàm trước.

Như ở hình bên, sinh viên sẽ:

- Tạo và chèn thêm những byte code thực thi của một số lệnh (phần ***executable codes*** màu cam đậm trong hình), có độ dài tuỳ thuộc vào các lệnh mà chúng đại diện.
- Tìm địa chỉ trả về mới phù hợp để ghi đè lên **địa chỉ trả về** (phần màu cam đậm phía trên) để thực hiện ý đồ dấu mũi tên.
- Các byte còn lại (màu cam nhạt) có thể mang giá trị tuỳ ý (khác 0x0A) hoặc tuỳ yêu cầu.



- **Tạo mã thực thi**

Để tạo mã thực thi trong chuỗi exploit, sinh viên thực hiện các bước sau:

- Viết code dưới dạng mã assembly trong các file **.s**. Ví dụ:

test.s

```
movl $1, %eax
int $0x80
```

- Chạy các lệnh để tạo các byte code tương ứng (khoanh đỏ) đưa vào chuỗi exploit.

```
$ gcc -m32 -c <file .s đầu vào> -o <file .o đầu ra>
$ objdump -d <file .o>
```

```
ubuntu@ubuntu: ~/LTHT/Lab6
ubuntu@ubuntu:~/LTHT/Lab6$ gcc -m32 -c input.s -o input.o
ubuntu@ubuntu:~/LTHT/Lab6$ objdump -d input.o

input.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   b8 01 00 00 00          mov    $0x1,%eax
   5:   cd 80                   int    $0x80
ubuntu@ubuntu:~/LTHT/Lab6$
```

- **Vị trí mã thực thi trong chuỗi exploit và địa chỉ trả về mới**

Để chương trình thực thi được các byte code thực thi trong chuỗi exploit, nên đảm bảo:

(1) Mã thực thi (executable codes) nằm ***ở đầu*** chuỗi exploit.

(2) **Địa chỉ trả về mới** là ***vị trí lưu*** của chuỗi exploit trong stack. Vị trí lưu này *chỉ xác định khi chương trình chạy*.

Các vị trí này có thể tuỳ chỉnh, tuy nhiên **luôn đảm bảo** rằng: **địa chỉ trả về mới *trỏ đúng*** vào **vị trí bắt đầu của những byte code đầu tiên** trong chuỗi exploit. Nếu không, khi chương trình nhảy đến vị trí những byte không phải mã thực thi, cố gắng thực thi chúng sẽ gây ra lỗi.

Nhiệm vụ của sinh viên là truyền vào các chuỗi exploit có *độ dài và nội dung chứa mã thực thi* phù hợp cho chương trình bufbomb (hay cho getbuf) để nó làm một số công việc thú vị.

D.2 Level 2

Trong file **bufbomb** có một hàm **bang** (cũng không được gọi trong **bufbomb**) như sau:

```
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 }
```

Level này yêu cầu sẽ gọi thực thi hàm **bang.** Tuy nhiên, hàm này có so sánh giá trị một biến toàn cục **global_value** (được gán ban đầu là 0) với giá trị **cookie**. Khi nào 2 giá trị này bằng nhau thì level này mới coi như thành công thành công. Vì giá trị **global_value** này không nằm trên stack (không nằm gần **buf**), nên không thể áp dụng phương pháp ghi đè những vùng nhớ lân cận ở lab trước, thay vào đó cần thực thi một số lệnh gán giá trị. Do đó, ta cần định nghĩa và truyền vào chương trình những code thực thi để đảm bảo trước khi gọi **bang**, ta thay đổi được giá trị của **global_value**.

Yêu cầu: Khai thác lỗ hổng buffer overflow để truyền vào **bufbomb** một chuỗi exploit có chứa mã thực thi sao cho:

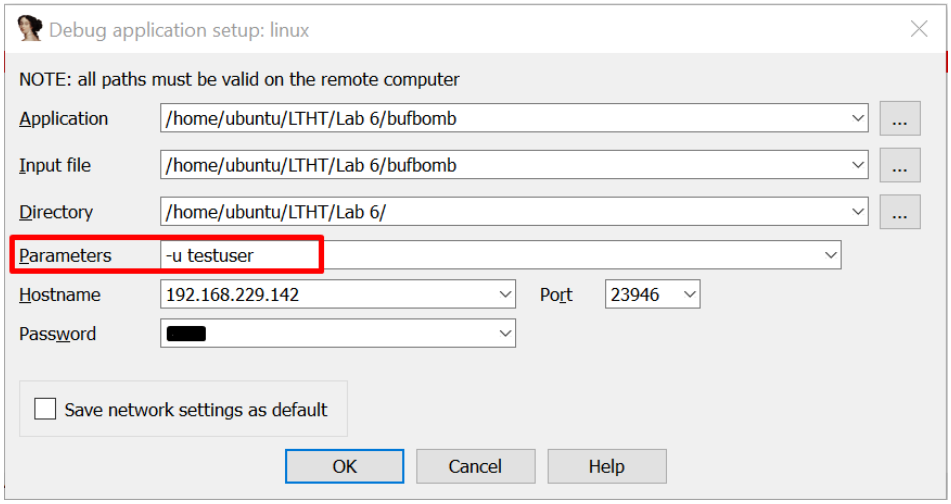
- Thay đổi được giá trị của **global_value**.
- Gọi được hàm **bang** thay vì trở về hàm **test**.

Gợi ý:

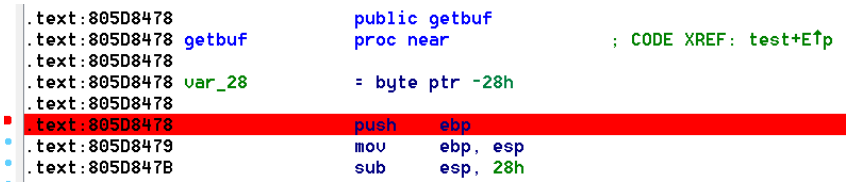
- ***Địa chỉ trả về mới*** nên là địa chỉ bắt đầu lưu chuỗi exploit trong stack. Địa chỉ này chỉ xác định trong lúc chương trình chạy, nên cần debug chương trình với breakpoint ở **getbuf** để xem vị trí chính xác của **buf**.

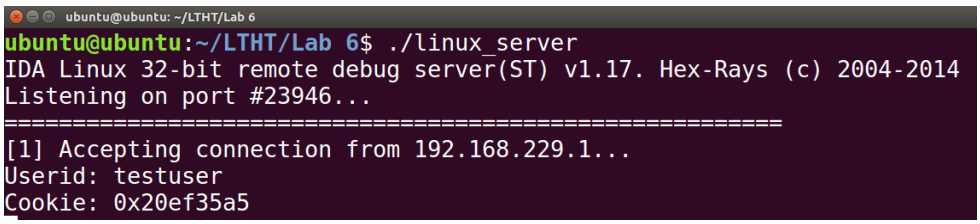
+ **Cách 1**: Thực hiện remote debug với IDA Pro (bản 32 bit)

Thiết lập cấu hình remote debugger (xem lại Lab 4) phù hợp, lưu ý cần truyền tham số `-u <userid>` khi debug.

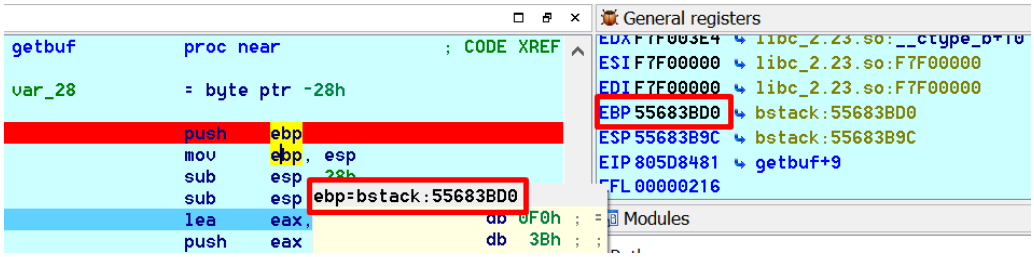


Ta cần tìm vị trí lưu của chuỗi **buf** trong stack của **getbuf**, đặt breakpoint tại hàm **getbuf** và tiến hành debug.





Ta cần di chuyển đến dòng lệnh phù hợp và xem giá trị các thanh ghi/ô nhớ khi ở dòng lệnh đó. Ví dụ bên dưới, khi debug đến dòng lệnh màu xanh dương đậm, trỏ trên code assembly hay quan sát ở cửa sở **General registers** cho ta giá trị của **ebp** = **0x55683BD0**.



+ **Cách 2**: Dùng GDB trên Linux

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

```
$ gdb bufbomb
(gdb) break getbuf
(gdb) run -u <userid>
```

```
ubuntu@ubuntu: ~/LTHT/Lab 6
ubuntu@ubuntu:~/LTHT/Lab 6$ gdb bufbomb
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 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:
<http://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 "apropos word" to search for commands related to "word"...
Reading symbols from bufbomb...(no debugging symbols found)...done.
(gdb) break getbuf
Breakpoint 1 at 0x805d847e
(gdb) run -u testuser
Starting program: /home/ubuntu/LTHT/Lab 6/bufbomb -u testuser
Userid: testuser
Cookie: 0x20ef35a5

Breakpoint 1, 0x805d847e in getbuf ()
```

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

```
(gdb) disassemble getbuf
(gdb) info registers ebp
```

```
(gdb) disassemble getbuf
Dump of assembler code for function getbuf:
   0x805d8478 <+0>:     push   %ebp
   0x805d8479 <+1>:     mov    %esp,%ebp
   0x805d847b <+3>:     sub    $0x28,%esp
=> 0x805d847e <+6>:     sub    $0xc,%esp
   0x805d8481 <+9>:     lea    -0x28(%ebp),%eax
   0x805d8484 <+12>:    push   %eax
   0x805d8485 <+13>:    call   0x805d7f28 <Gets>
   0x805d848a <+18>:    add    $0x10,%esp
   0x805d848d <+21>:    mov    $0x1,%eax
   0x805d8492 <+26>:    leave
   0x805d8493 <+27>:    ret
End of assembler dump.
(gdb) info registers ebp
ebp            0x55683bd0       0x55683bd0 <_reserved+1039312>
(gdb)
```

Cả 2 cách debug cho giá trị **ebp = 0x55683BD0**. Sử dụng giá trị ebp này để tính tiếp vị trí cụ thể của chuỗi **buf**?

- Mã thực thi (executable code) trong chuỗi exploit cần thực hiện 2 công việc:
  1. Gán giá trị cookie của userid cho biến toàn cục **global_value** (có thể tìm vị trí của **global_value** khi xem code assembly của hàm **bang** có sử dụng nó)
  2. Nhảy đến hàm **bang** để thực thi tiếp.
- Gợi ý: Một cách để đến thực thi 1 hàm:

```
push $<địa chỉ của hàm muốn thực thi>   // hằng số
ret
```

D.3 Level 3

Ở các level trước, trong quá trình bị tấn công buffer overflow, có 1 số vùng nhớ trên stack bị ghi đè bằng những byte tuỳ ý, trong đó có những vùng nhớ chứa thông tin liên quan đến các trạng thái thanh ghi/bộ nhớ của hàm mẹ (hàm **test**). Việc ghi đè này có thể khiến chương trình không thể quay về hàm ban đầu sau khi bị buffer overflow. Các tấn công ở những level trước chỉ khiến cho chương trình nhảy đến đoạn code của những hàm khác, sau đó thoát chương trình, do đó ảnh hưởng này không rõ rệt. Mục đích của level này là làm cho chương trình sau khi bị khai thác, thực thi một số hoạt động nhất định vẫn có thể quay về hàm mẹ ban đầu (hàm **test**). Để làm được điều đó, sinh viên cần biết được các trạng thái thanh ghi/bộ nhớ của hàm mẹ đã bị thay đổi và khôi phục lại giá trị đúng.

Kiểu tấn công này cần thực hiện các bước:

1) Đưa được mã thực thi lên stack thông qua input

2) Thay đổi địa chỉ trả về thành địa chỉ bắt đầu của chuỗi exploit chứa mã thực thi

3) Trong đoạn mã thực thi, bên cạnh việc thực hiện một công việc nào đó, cần khôi phục bất kỳ thay đổi nào đã gây ra với stack và trở về đúng hàm mẹ ban đầu.

Yêu cầu: Khai thác lỗ hổng buffer overflow để truyền vào một chuỗi exploit chứa mã thực thi sao cho **getbuf** khi thực thi xong, giá trị trả về sẽ là cookie tương ứng với userid cho hàm **test**, thay vì trả về 1.

Mã thực thi cần có các lệnh thực hiện các công việc:

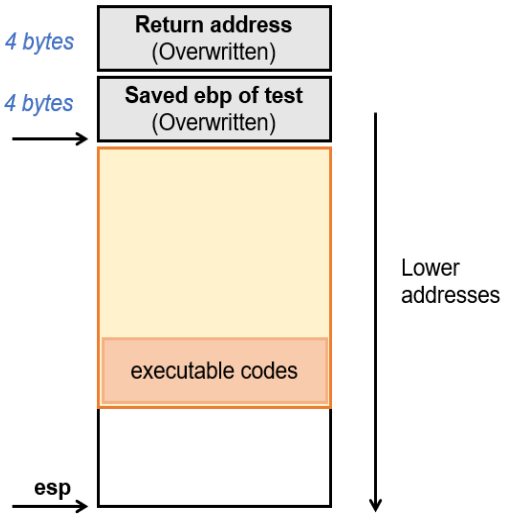
- Gán cookie vào giá trị trả về.
- Khôi phục các trạng thái thanh ghi/bộ nhớ bị thay đổi của hàm mẹ (**test**)
- Đẩy địa chỉ trả về đúng vào stack (là 1 câu lệnh cần thực thi tiếp theo của **test**).
- Thực thi câu lệnh **ret** để trở về **test**.

Gợi ý:

- Giá trị trả về của hàm thường lưu trong %eax.

- Quan sát stack trước và sau khi lưu input, không tính địa chỉ trả về bị thay đổi, có ô nhớ lưu **ebp của test** bị ghi đè. Làm cách nào tìm được giá trị ban đầu của nó?
- Có thể khôi phục giá trị thanh ghi bằng 1 trong 2 cách:
  (1) Ghi đè trực tiếp giá trị lên ô nhớ ở vị trí tương ứng trong chuỗi exploit.
  (2) Khôi phục trong mã thực thi bằng lệnh:

```
movl <giá trị>, <thanh ghi/địa chỉ>
```



Yêu cầu thêm (+1 điểm): Có 1 cách để khôi phục giá trị ebp cũ của hàm test bằng code thực thi nhưng không cần debug để tìm giá trị chính xác. Sinh viên thử đề xuất phương pháp và thực hiện tấn công thử?

E. YÊU CẦU & ĐÁNH GIÁ

Sinh viên thực hành và nộp bài **theo nhóm tối đa 2 sinh viên** theo thời gian quy định. Sinh viên nộp cả file báo cáo trình bày:

- Các bước thực hiện để xác định được các chuỗi exploit cho từng phần **D2, D3**.
- Hình ảnh chụp màn hình kết quả thực thi file với chuỗi exploit.

Lưu ý: báo cáo cần ghi rõ nhóm sinh viên thực hiện.

File báo cáo .pdf được đặt tên theo quy tắc sau:

Lab6-NhomX_MSSV1-MSSV2.pdf

Ví dụ: *Lab6-Nhom2_19520yyy-19520zzz.pdf*

F. THAM KHẢO

[1] Randal E. Bryant, David R. O'Hallaron (2011). *Computer System: A Programmer's Perspective*

[2] Hướng dẫn sử dụng công cụ dịch ngược IDA Debugger – phần 1 [Online] https://securitydaily.net/huong-dan-su-dung-cong-cu-dich-nguoc-ma-may-ida-debugger-phan-1/

HẾT