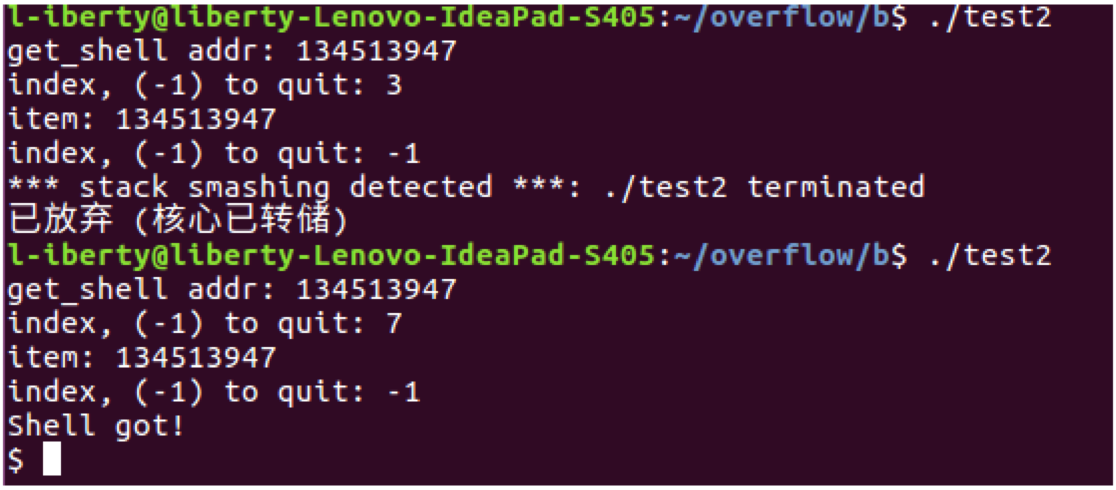
Stack Overflow (3)

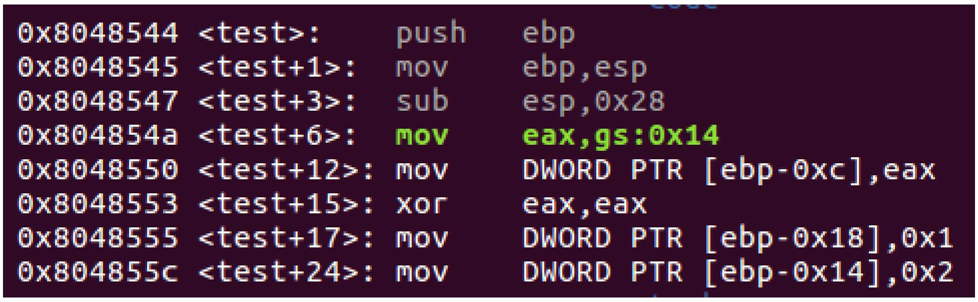
仍以b部分的代码为例，开启堆栈保护 `gcc test.c -o test2 -m32`

尝试向buf[3]和buf[7]赋值，结果如下:



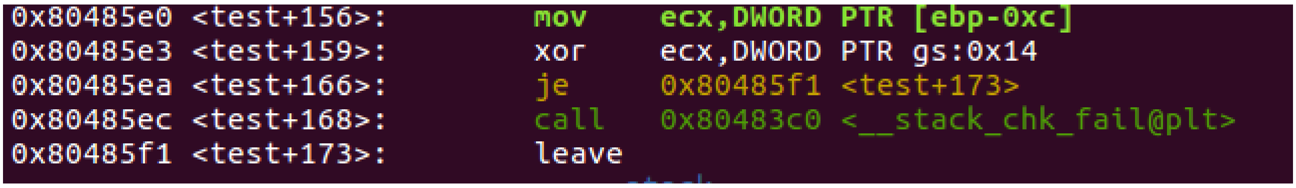
```
l-iberty@liberty-Lenovo-IdeaPad-S405:~/overflow/b$ ./test2
get_shell addr: 134513947
index, (-1) to quit: 3
item: 134513947
index, (-1) to quit: -1
*** stack smashing detected ***: ./test2 terminated
已放弃 (核心已转储)
l-iberty@liberty-Lenovo-IdeaPad-S405:~/overflow/b$ ./test2
get_shell addr: 134513947
index, (-1) to quit: 7
item: 134513947
index, (-1) to quit: -1
Shell got!
$ 
```

向buf[3]赋值导致溢出并被检测到，向buf[7]赋值则成功拿到shell，下面通过gdb反汇编分析:



```
0x8048544 <test>:    push   ebp
0x8048545 <test+1>:  mov    ebp,esp
0x8048547 <test+3>:  sub    esp,0x28
0x804854a <test+6>:  mov    eax,gs:0x14
0x8048550 <test+12>: mov    DWORD PTR [ebp-0xc],eax
0x8048553 <test+15>: xor    eax,eax
0x8048555 <test+17>: mov    DWORD PTR [ebp-0x18],0x1
0x804855c <test+24>: mov    DWORD PTR [ebp-0x14],0x2
```

ebp-0xc处的内存紧邻buf[2]，称为溢出检测单元，该处存放一个来自gs:0x14的随机数，若向数组赋值时发生溢出就可能覆盖检测单元，程序最后比较溢出检测单元和gs:0x14处的数值，如果不相等则认为发生溢出(如下图).



```
0x80485e0 <test+156>:        mov    ecx,DWORD PTR [ebp-0xc]
0x80485e3 <test+159>:        xor    ecx,DWORD PTR gs:0x14
0x80485ea <test+166>:        je     0x80485f1 <test+173>
0x80485ec <test+168>:        call   0x80483c0 <__stack_chk_fail@plt>
0x80485f1 <test+173>:        leave
```

但是如果回避溢出检测单元，直接覆盖返回地址，溢出攻击仍可成功.