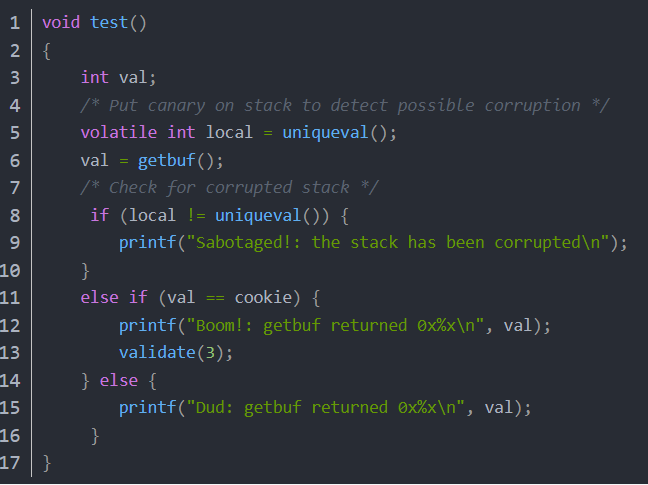
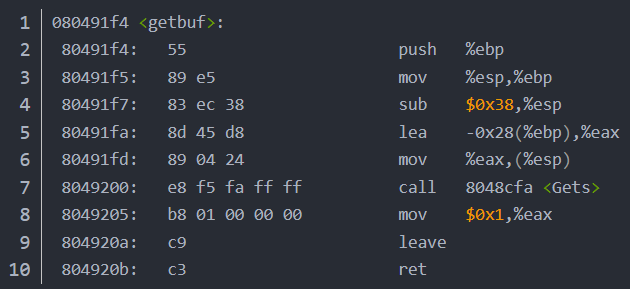
## 实验名称：buflab

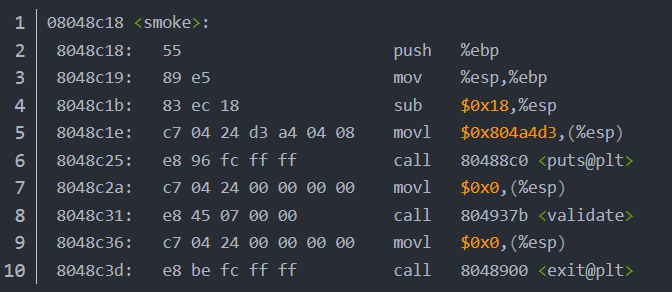
## Level0



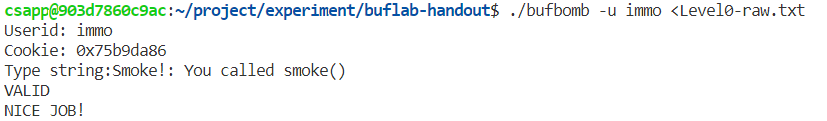
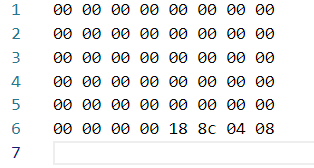
主函数如上，当test调用完getbuf函数后会正常向下执行，如果我们想要让其跳转到smoke函数，那么我们就要利用缓冲区溢出的漏洞来修改getbuf函数的返回地址。



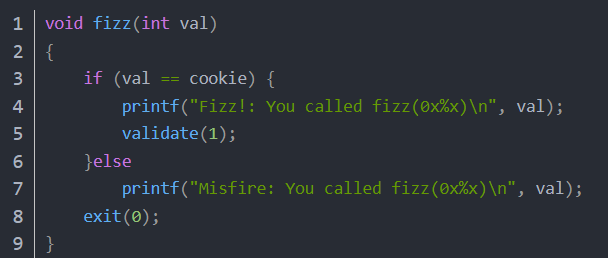
再来看getbuf函数的汇编代码，从第五行可以看出开辟了40个字节用以存储输入的字符串。而返回地址是在%ebp+0x4处，push %ebp本身又占了四个字节，所以结构为：0x28+4+4=48个字节。并且其最后4个字节应是smoke函数的地址，正好覆盖ebp上方的正常返回地址。这样再从getbuf返回时，取出的根据攻击字符串设置的地址，就可实现控制转移。



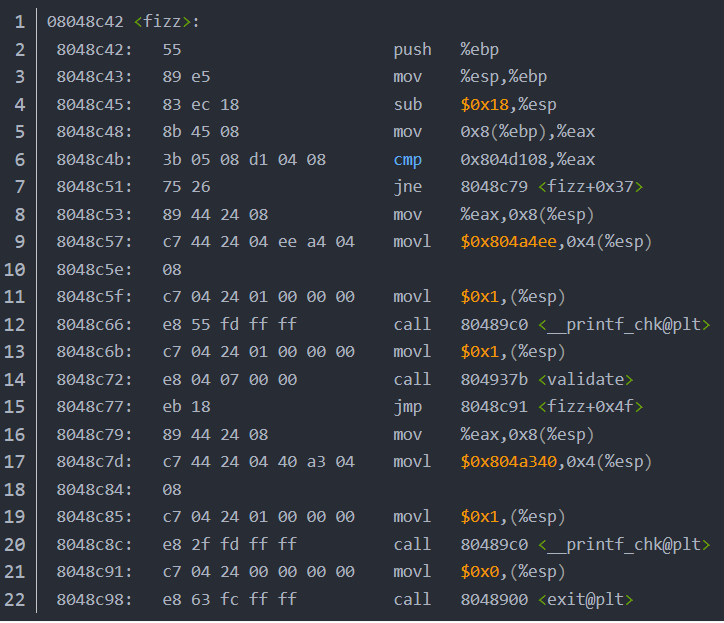
由反汇编可得smoke函数的入口地址为0x08048c18。因此，我们需要做的就是把上面的44个字节随意填满（不要填换行），然后把原来的返回地址改为smoke函数的入口地址。故答案如下，结果通过。



## Level1



Level1 和Level0差不多，唯一的区别是 fizz(int) 函数有一个整型的参数，并且在 fizz函数中还要校验cookie， test函数调用getbuf函数，调用完getbuf以后不返回getbuf的调用者test而是去执行fizz函数。



由fizz的反汇编可知：fizz函数的入口地址为0x08048c42。由栈帧图示可知，ebp存放了调用者的旧ebp（saved %ebp），其上一位置ebp+4存放了调用者的返回地址，所以参数的地址应该为ebp+8的位置，我们只需要将自己的cookie放置在该位置即可。

但在这个过程中getbuf函数和fizz函数的栈结构不同，getbuf函数调用时是正常调用，ebp+4为返回地址，ebp+8为主函数栈末尾。而调用fizz函数是通过缓冲区漏洞强行进入，系统不会存储返回地址，此时ebp指向的位置是getbuf中的ebp+4的位置，此时第一个参数ebp+8指向的位置是getbuf中的ebp+12的位置.

故答案如下，结果通过。

