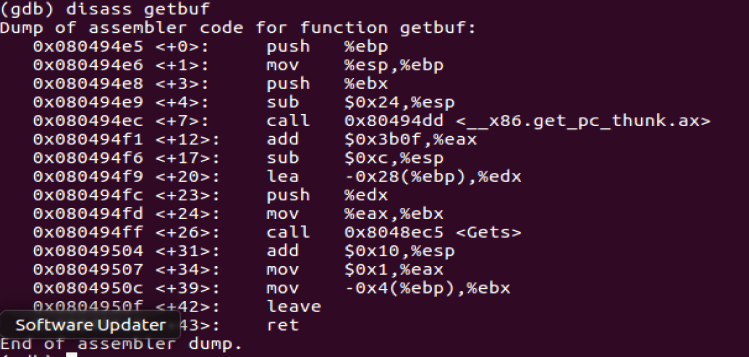
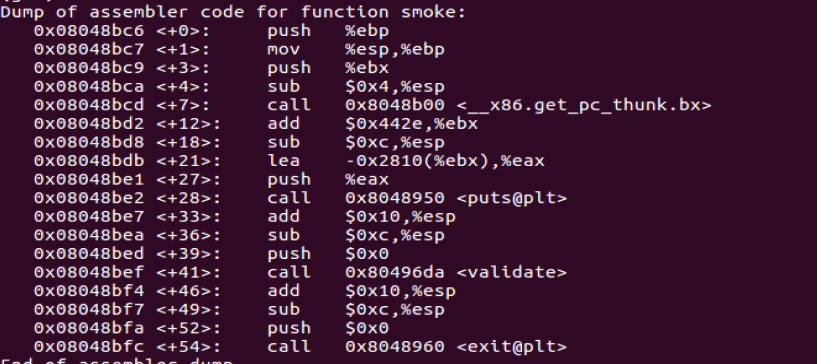
**Level0：**

第一题当test函数调用getbuf函数时，按照惯例会返回test函数，而要求救赎要当getbuf函数执行结束后，返回到smoke函数中；大致目的就是输入一个过长的字符串，把getbuf函数的返回地址覆盖掉，改成smoke函数的地址。

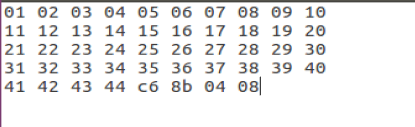
首先反汇编getbuf函数：



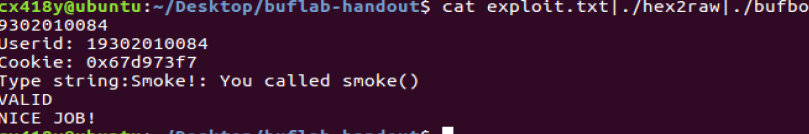
可以发现调用get函数写数据的位置距返回地址有44字节，加上返回地址就是48字节，所以只需要在前44个字节任意输入，在后四个字节输入smoke的入口地址就可以了，接下来反汇编smoke函数：

****

可以发现入口地址是0x08048bc6，小端法保存就是c6 8b 04 08，所以可以写如下的exploit.txt文件；

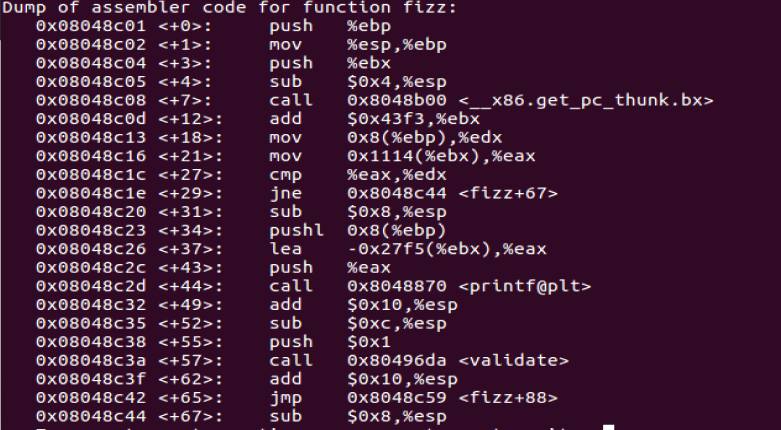


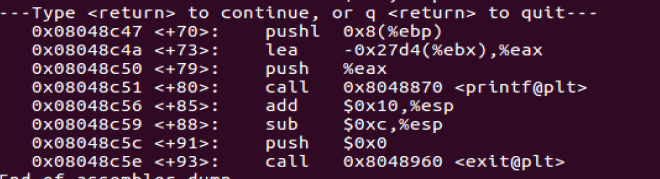
然后检测，成功调用了smoke函数：



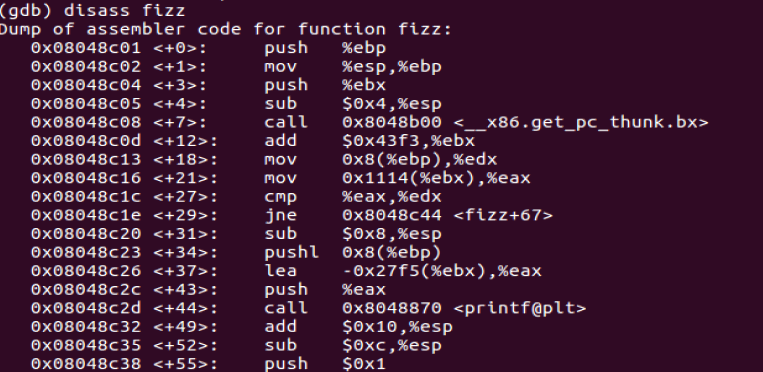
**Level1：**

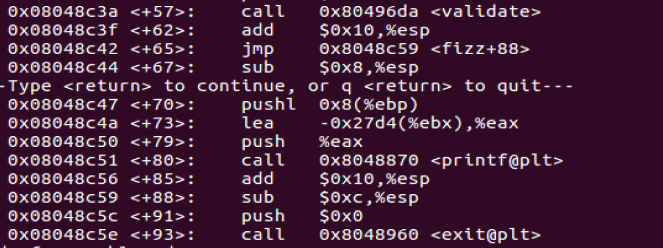
与level0类似，我们需要让test函数返回到fizz，而不是test，但是额外的要求是要传入自己的cookie作为参数，首先还是反汇编fizz函数：



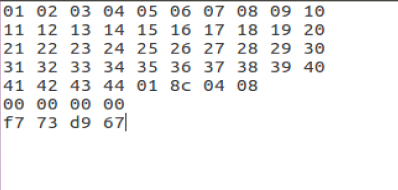


可以看到依旧是前44位是可以随便填入，第45-48是返回地址，只需要改成fizz函数的入口地址就可以了，我们先反汇编fizz函数：

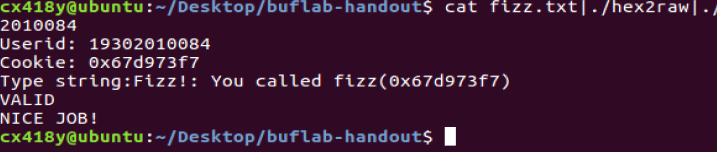




可以看到fizz的入口地址是0x08048c01，小端表示为01 c8 04 08，可以看到fizz函数中先将%ebp压入栈，在将%ebx压入栈，之后将%ebp+8处的值给%ebx，所以在输入fizz的入口地址之后，应该再任意输入四个字节，之后再输入cookie的值，于是可以有以下输入：



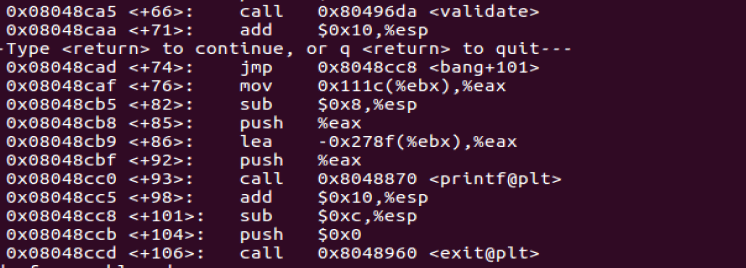
之后测试，成功得到结果：



**Level2：**

与Level0和Level1类似，需要让bufbomb在执行过程中执行bang而不是回到test；首先我们需要找到全局变量global\_value的内存地址，所以先反汇编bang函数：

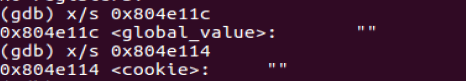




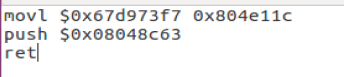
可以看到在<+18>~<+32>是将地址为%ebx+0x111c处的值和地址为%ebx+0x111c处的值进行比较，于是我们在getbuf处设置断点，进入查看%ebx的值如下：



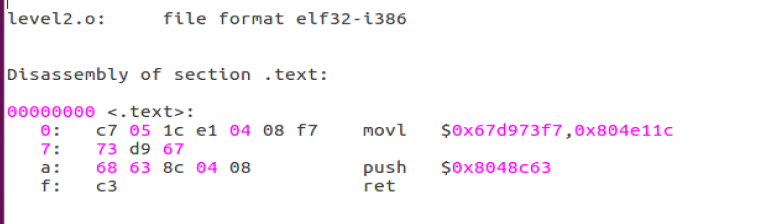
然后得到地址%ebx+0x111c和%ebx+0x111c并查看，发现分别储存的是全局变量global\_value和cookie：



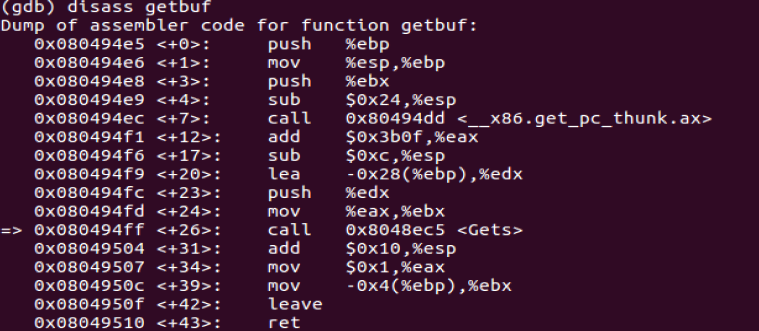
于是可以编写如下的汇编代码将cookie的值传给global\_value，然后将bang的入口地址压入栈，之后返回。并将其转化为机器码：



机器码：

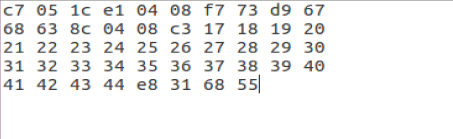


接下来考虑执行getbuf函数的时候，将其返回地址修改为这个函数的地址，然后getbuf执行完后就执行这个函数，然后自动执行bang函数，所以要找到我们写的这个函数的地址，在getbuf中设置断点，反汇编getbuf：

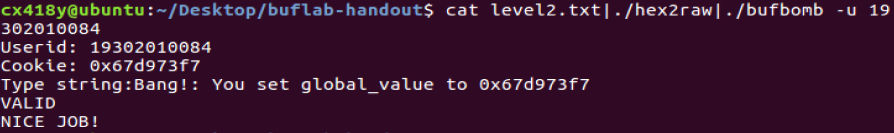
我们的函数的地址应该是在%edx中，于是查看%edx的值：



然后就可以根据机器码和函数地址构造输入的内容了：

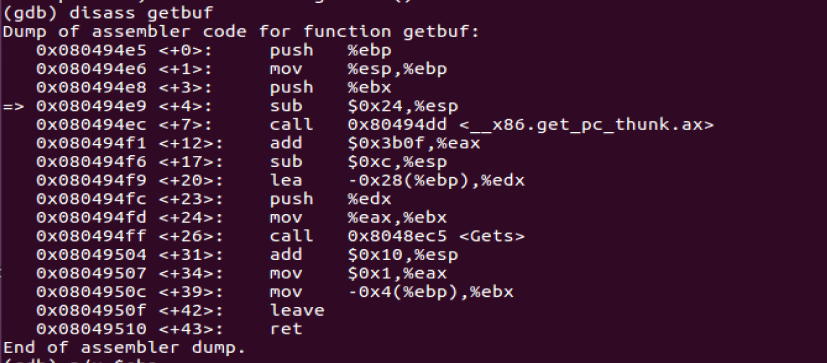


然后测试一下：



**Level3：**

level3要修饰修改getbuf的返回值为cookie，而不是1,；需要解决的问题有两个，一个是修复破坏的栈帧，另一个是修改返回值，与前几个level不同，这个level需要正确地返回到test函数，所以我们急需要保证旧的%ebp不被改写，又要保证返回地址不出错。首先在getbuf处设置断点，运行，反汇编getbuf函数：



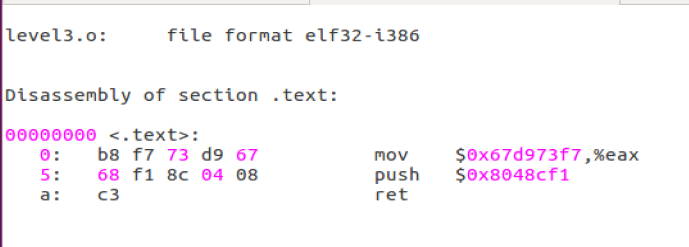
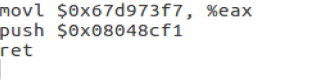
然后查看此时%ebp的值：



对于返回地址，可以把getbuf的下一句的地址再压栈，所以下反汇编test：



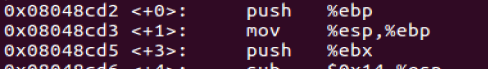
可以看到getbuf执行完后的下一句地址为0x08048cf1，因此我们可以编写汇编代码中先将cookie的值赋给%eax，并将下一条指令的地址压栈并生成机器码：



之后是还原ebp，在输入的字符串41-44字节还原ebp的值，在进入getbuf函数前一步查看ebp：

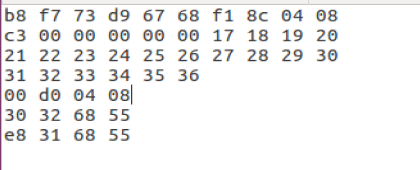
用小端表示应该是30 32 68 55；

之后是返回地址，和level2的返回地址一样，注意到test函数一开始将%ebx压栈，

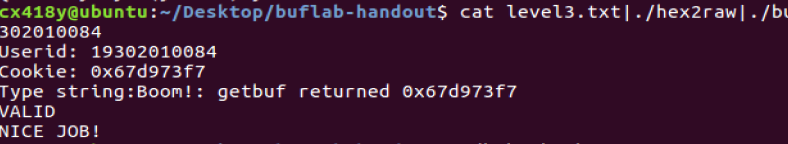


并且通过后面的代码可以发现%eax的值是通过%ebx得到的，所以%ebx也应该要复原，所以先查看%ebx一开始的值，并且写在36-40字节的位置，

于是就可以构造字符串了：

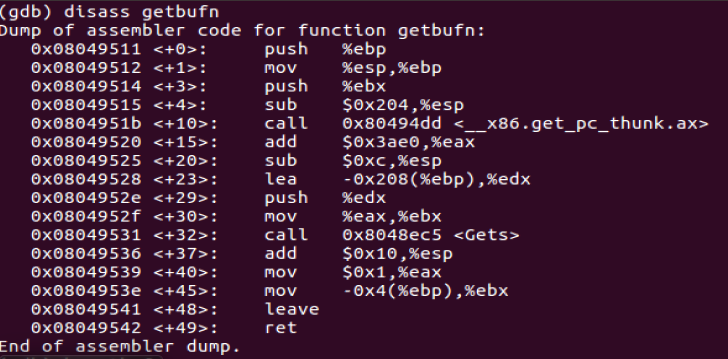


然后测试：



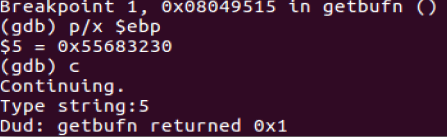
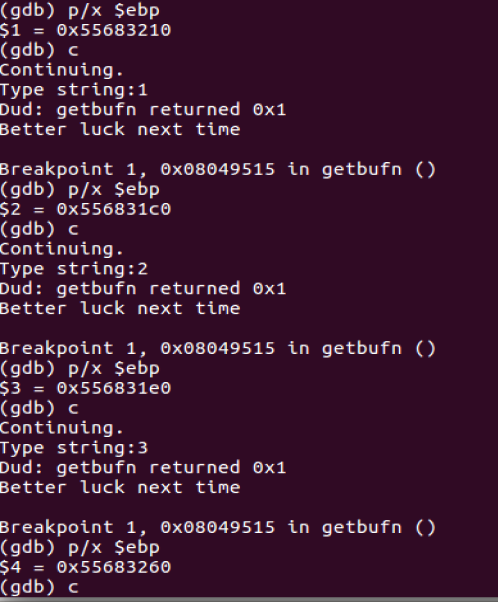
**Level4：**

Level4算是level3的进阶版，要使getbufn函数连续运行5次，每次需要设置返回值为我们自己的cookie，并正确的返回到testn函数中。首先还是先查看getbufn的汇编代码：

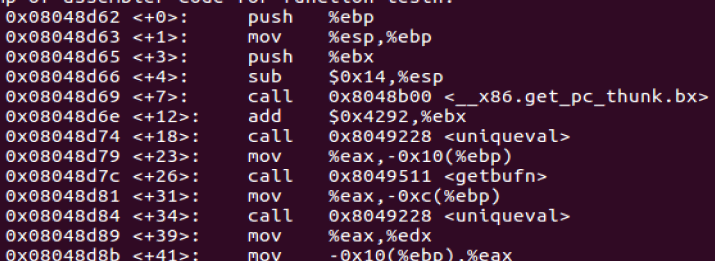


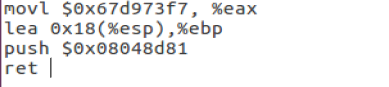
可以看到buf的首地址为%ebp-208，也就是有520字节的大小。这样每次运行testn的ebp都是不同的，因此getbufn保存的testn的ebp也是随机的，但是不变的是栈顶的esp是始终固定的，因此可以用esp表示ebp。

首先在getbufn处设置断点，并使用-n模式运行成程序，并查看ebp的值，虽然ebp的值是变化的，但是运行五次可以看到他的范围，只需要找到他的最大值，如图所示最大值应该是0x55683260，用他减去0x208，就是最高的buf地址0x55683058。

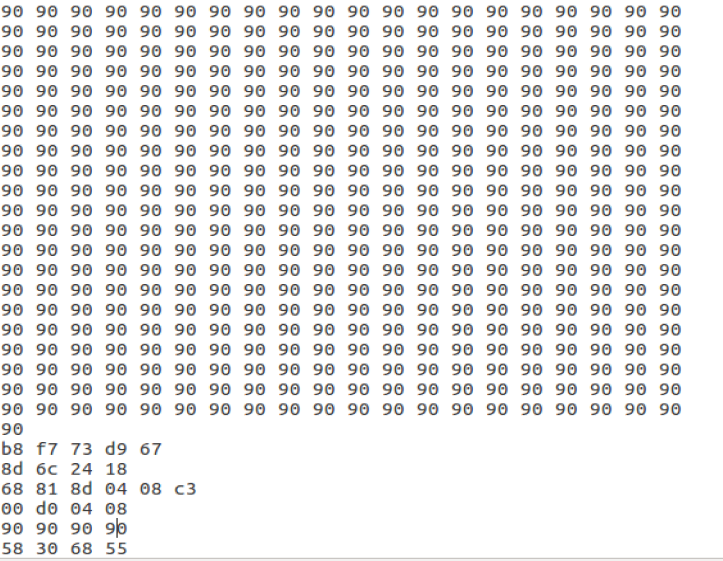


接下来在testn的汇编代码中可以看到testn中原%ebp的值为%esp-0x18，getbufn返回的地址是0x08048d81，于是可以编写汇编代码：





由于缓冲区有520个字节加上返回地址和原ebp,总共528个字节。注意同level3一样还需要复原ebx的值，需要在ebp前四个字节写上原ebx的值，除去上述机器码15个字节，返回地址4个字节，ebx4个字节，剩余505个字节用nop填充，对应ascii码为90。如下：（只截取了片段）



然后测试：

