Lab3实验报告

前情提要：

Lab3是关于栈溢出攻击的

做法有奇奇怪怪的，反正什么都有

然后致敬一下周四下午

传奇的一个下午

T3随便写了一下然后过了

心想这不是插代码很简单吗？

然后到T4的时候爆了一下午栈

莫名其妙的被吃掉字符等等问题

最后莫名奇妙好了

不过最后也算是写完了

还有

关于最后一题

没人说运行hex2raw要加-n的

我自己没加-n后发现只输入了一次字符串

然后就发明了复制5次字符串

然后两个字符串用换行隔开

也就是0a

（小声bb：下次ppt能不能说明一下啊）

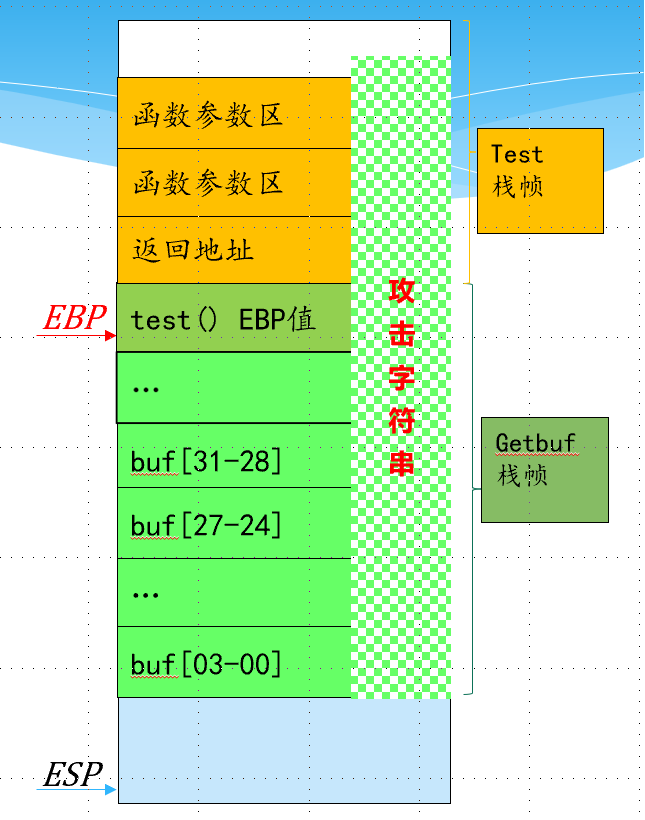
本实验经我探索

发明了一些奇奇怪怪的做法

但是我会先说正常人的做法，然后再讲逃课流做法

现在是正常做法:

回顾一下栈的结构



以上是栈的结构

而gets这种不安全的函数会一直读，读到结束或换行为止

所以就有可能把旧的%ebp的值，返回地址都洗掉

而我们就是要把返回地址改掉，然后跳到其他地方执行

（小声bb：这getbuf函数怎么没有金丝雀值保护一下啊？）

但是我们洗到返回地址的同时也会把旧的%ebp值洗掉

到T4的时候就会出问题

没事

我们一步一步分析

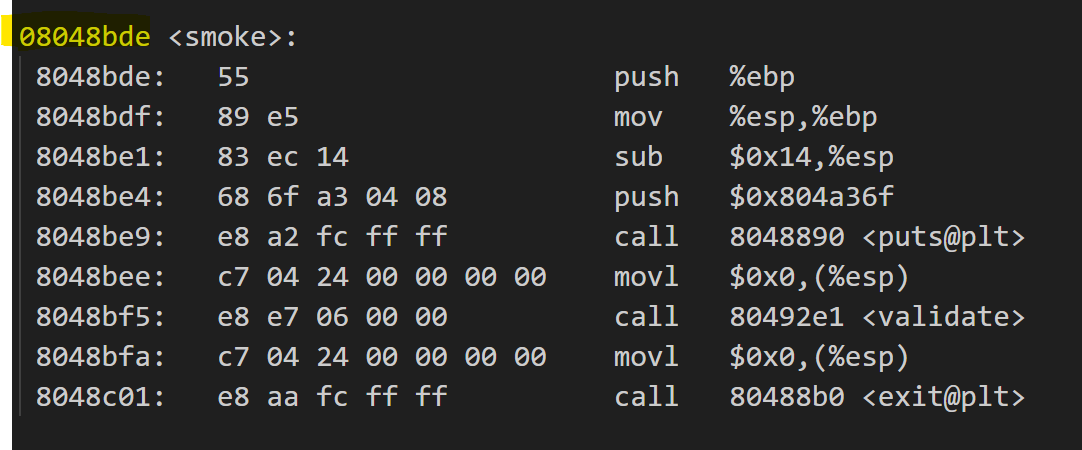
等下就知道问题所在了

Smoke

要求我们调用smoke函数

那我们只需要把返回地址写成smoke函数的开头就行了

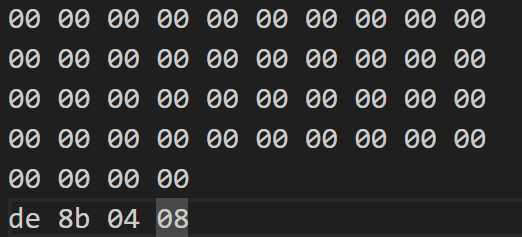
也就是下图黄色部分



因为地址是小端法

所以地址要反着写

也就是下图



所以然后用下hex2raw就行了

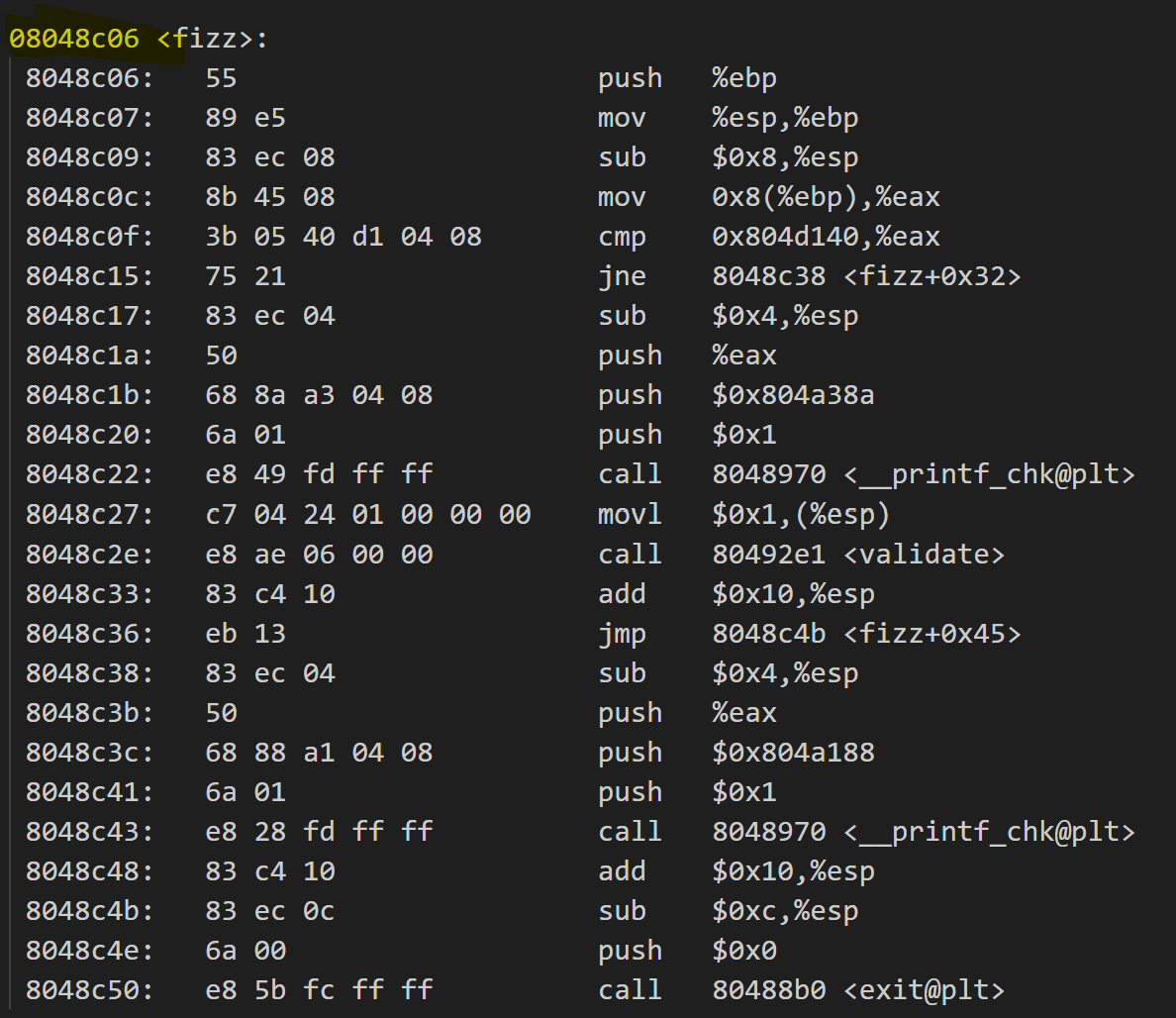
Fizz

本题要求

调用fizz，同时以自己的cookie作为参数

返回地址很好求

就是下图黄色部分



但是怎么改调用函数的参数呢？

（关于一些指令的说明（个人理解）：

leave 等价于 mov %ebp,%esp 和 pop %ebp

ret 等价于 pop %eip

）

执行完getbuf的ret

%ebp已经不知道指到哪了（因为%ebp的旧值已经不见了）

但是%esp的值指的是返回地址加4的地方

然后到fizz函数时，它push %ebp

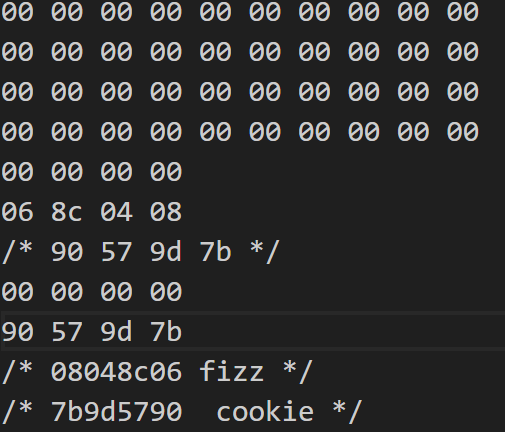
又把%esp赋给了%ebp，所以现在%ebp的指的地方就是返回地址的地方（但是现在放的不是返回地址了）

好，那我们又知道传的第一个参数的位置是 %ebp+8的地方

所以我们只要在返回地址+8的地方开始写cookie就行了

也就是如下图

然后我们就可以顺利地通过此关了



Bang

这题要求我们把调用bang函数，同时把全局变量的值改成cookie值

当然这题如果直接把返回地址写到bang函数，我们是改不了全局变量的值的

还记得我们输入的字符串吗？

诶

这不是可以运行一段程序吗

只要我们把pc指到这里

就是把返回地址写到这里

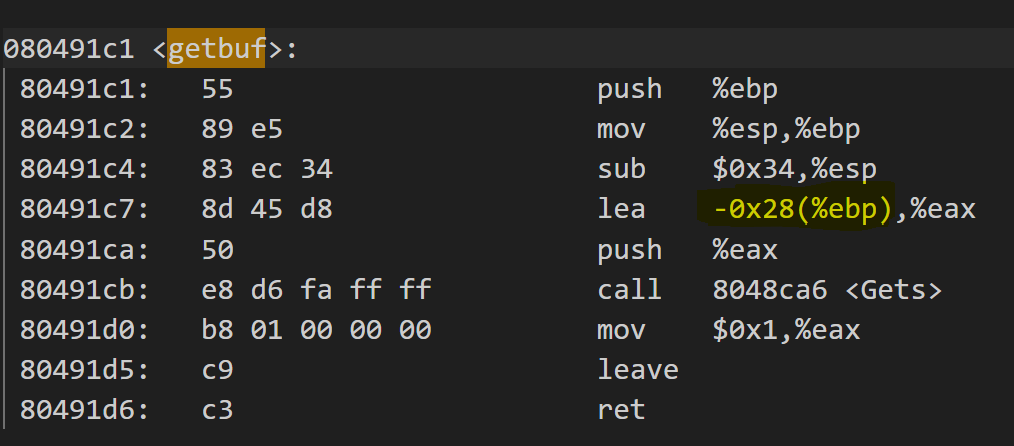
哦

恍然大悟

所以我们就可以把pc指到这里

然后运行恶意代码了

那这里是哪里呢？



不就是%ebp-0x28吗？

然后我们通过gdb

合理的读出%ebp的值然后-0x28就是我们要的返回地址了

好那我们要执行什么恶意代码呢？

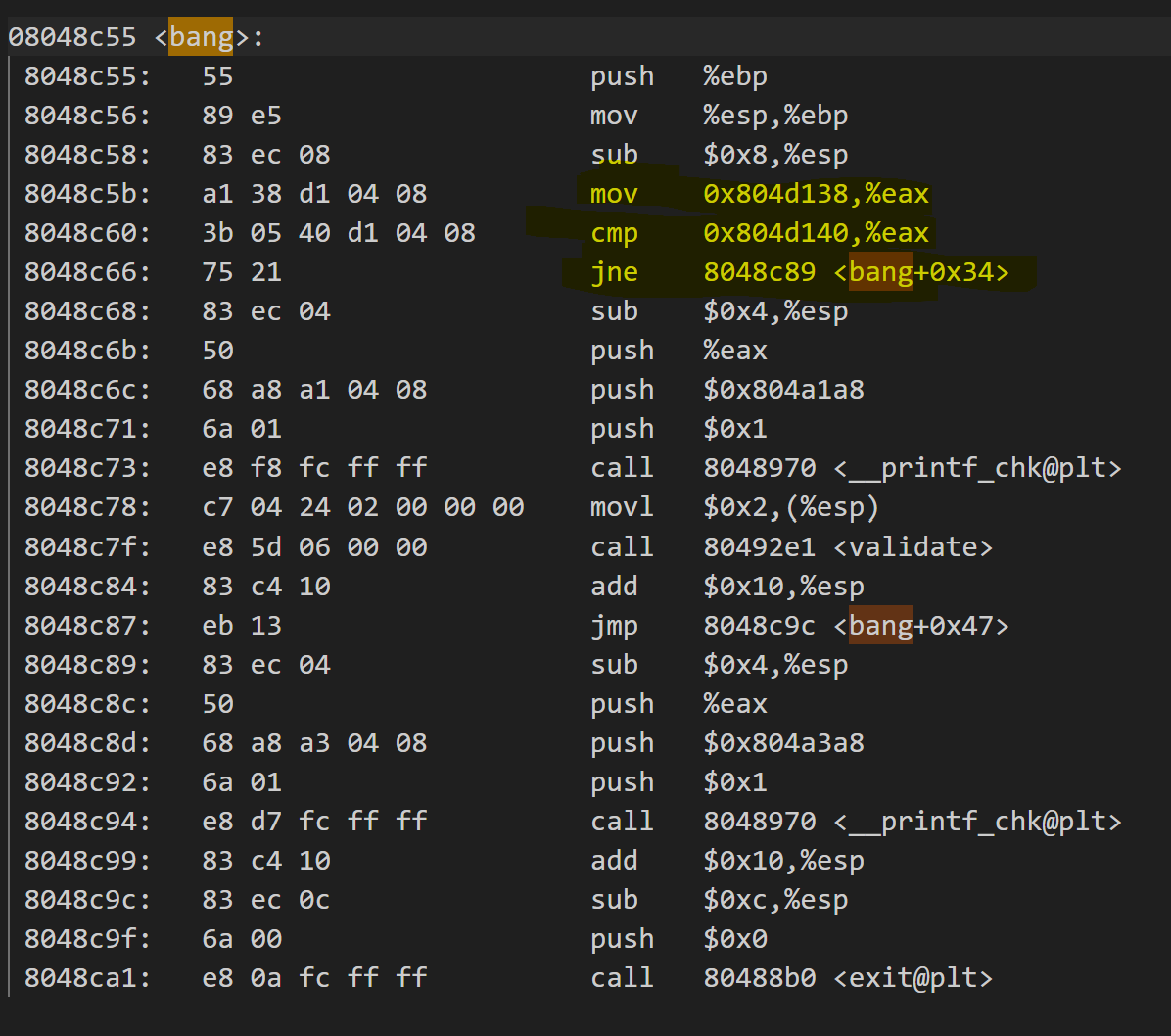
1是要把全局变量的值给改了

2是要跳到bang执行

要解决第一个问题

问全局变量在哪呢？

Bang中有如下的三行



我们可以查一下值，发现0x804d140好像存的是cookie的值

0x804d138好像是全局变量的地址

所以我们可以直接mov cookie值到%eax，然后再mov %eax,0x804d138

当然一步完成好像也可以？（不懂没试过？反正我是两步完成的）

因为知道了cookie的值存放位置，我就直接把从cookie值存放的位置移到寄存器，然后再移到全局变量中。

最后要调用函数

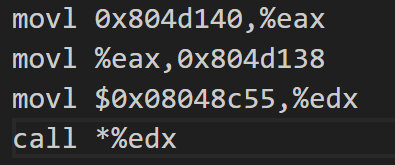
调用是用call就是了

然后关于call的二进制编码是用的pc相对的寻址方式

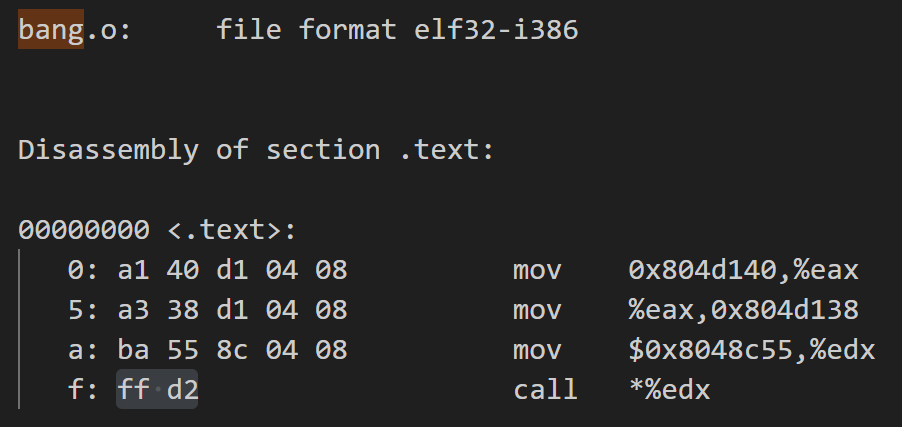
（算的话容易错）

我是先把地址放到寄存器中，然后call \*寄存器

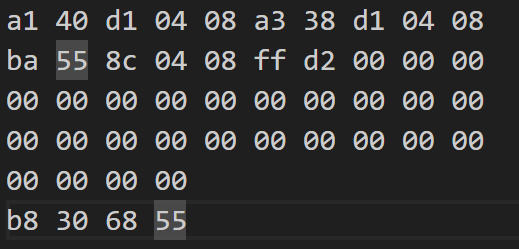
然后就有如下图的汇编代码



然后链接再反汇编就能生成二进制代码了



然后把他们抄到字符串里就完事了



Boom

爆栈仙人爆了一下午的传奇故事，不过多赘述

本题是要求返回值为cookie然后正常返回回test

所以正常写也是要插代码了

然后你正常去写

前面是正常的赋值给%eax

同时返回

返回就有好多种写法了

有 push 然后 再ret的

我是把值存在寄存器中，然后再jmp \*寄存器

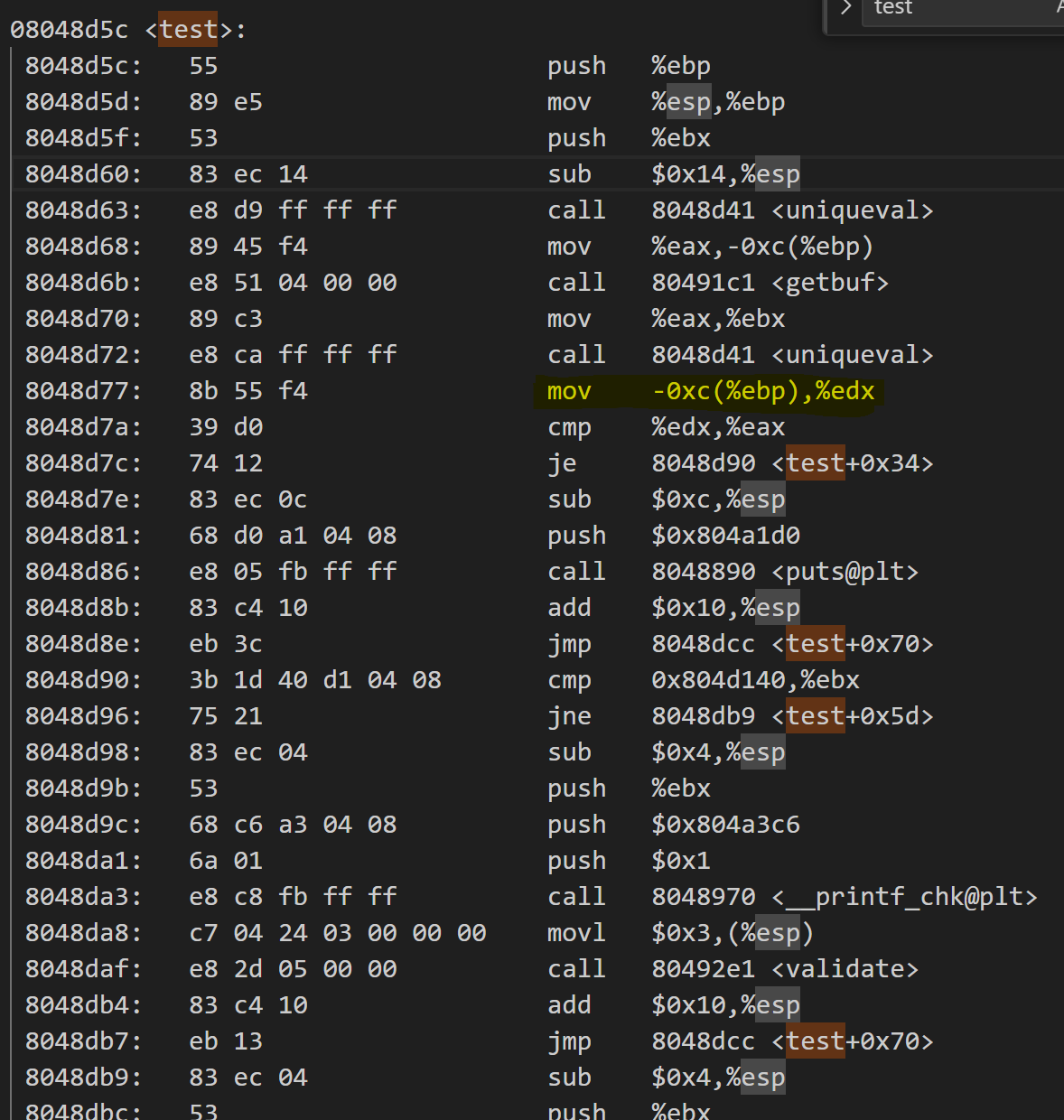
Ok，然后你这样写完

发现被检测爆栈了

然后你一步步调试

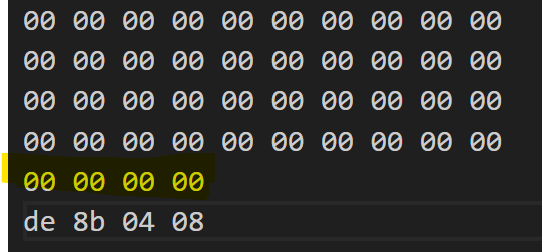
发现某个地方用到了%ebp，但是%ebp，又被我们洗掉了

我们只能把它搞回来了



怎么搞呢？

1. 可以直接gdb读出原来%ebp的值，然后我们输字符串的时候不要把%ebp的值洗成0了，把它改成原来的值就好,就是如下图黄色部分



1. 是可以通过%esp计算%ebp，因为每次test开的栈帧是固定大小的

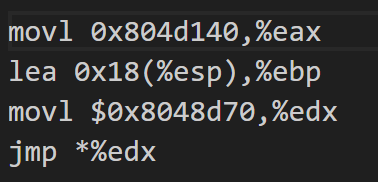
所以估计有相对差值

然后我们根据gdb读出两个的值，%ebp可以在调用getbuf前读，或者你在getbuf里读也可以但是要在gets前读。%esp要在ret后读

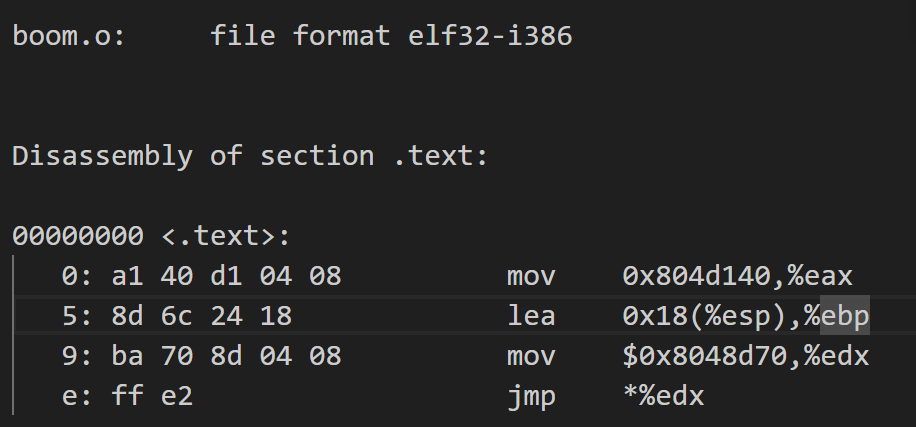
最后在汇编代码就可以写出如下代码



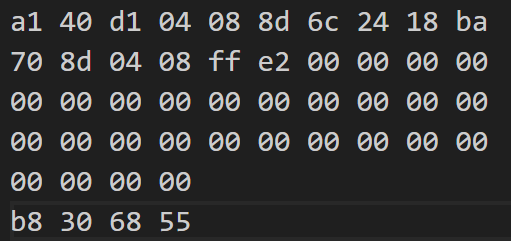
最后汇编代码就是



然后汇编反汇编，就有以下部分



最后把它们插到字符串里就好了



Nitro

反正做这题我是把字符串复制了5遍

然后用换行隔开，也就是0a也能过

（谁叫-n不早点说呢）

所以怎么写呢？

现在%esp会变了

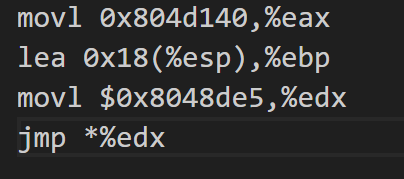
但是给的字符串足够长

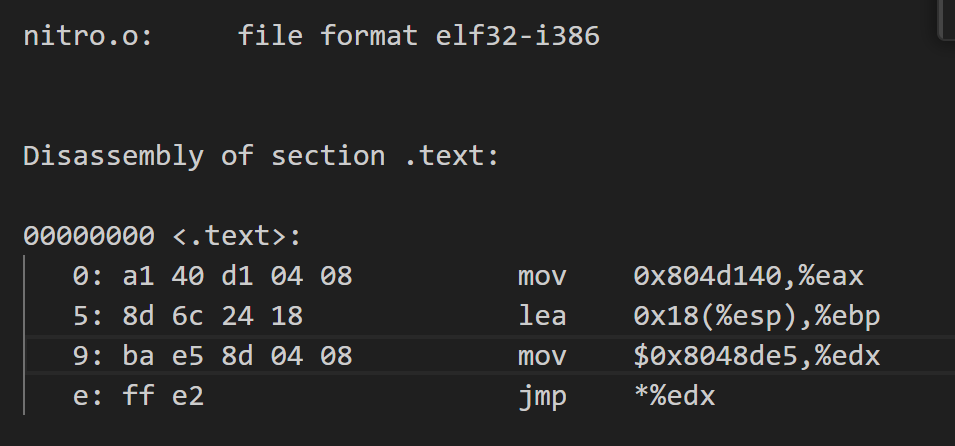
我猜测

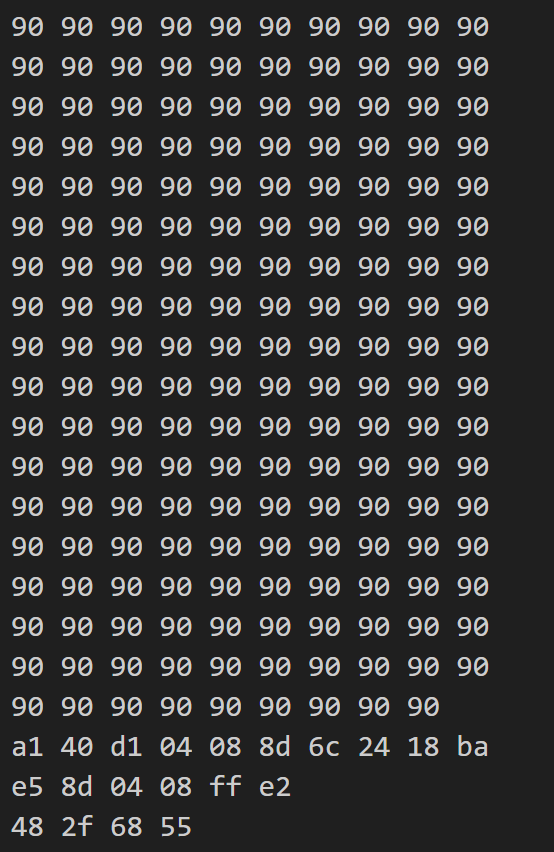
%esp的变化幅度应该不会很大

所以我直接把第一个%ebp-0x104作为返回地址，然后前面加nop

最后放我们要执行的代码，也就是和boom一样就可以了







剩下就不过多解释吧

反正这些文件在lab3中

有兴趣可以自己看看吧（因为主要还是讲怎么逃课吧）

**逃课流**

计基实验太难怎么办？

拼尽全力却无法战胜！

那我们就不要题目的要求

只要让它过了就好

怎么过呢？

关于逃课流

前三题可以直接改返回地址

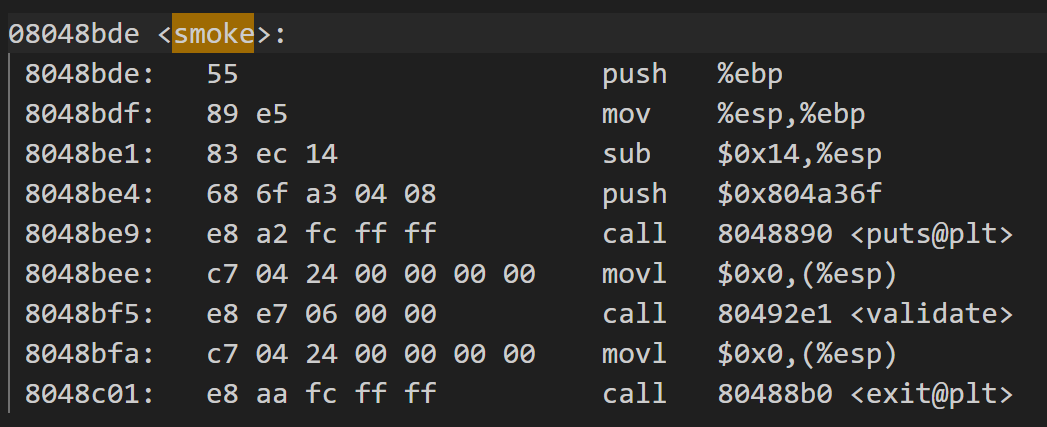
然后第四题只要会用gdb就可以了

最后一题好像必须要插恶意代码了（暂时没想到其它好的逃课方式）

好

然后让我们分析一下这些函数

举个smoke的例子吧



发现它调用了puts

猜测应该是输出调用成功的信息

然后调用了validate，目测应该是传数据的吧？或者是判断正确性的

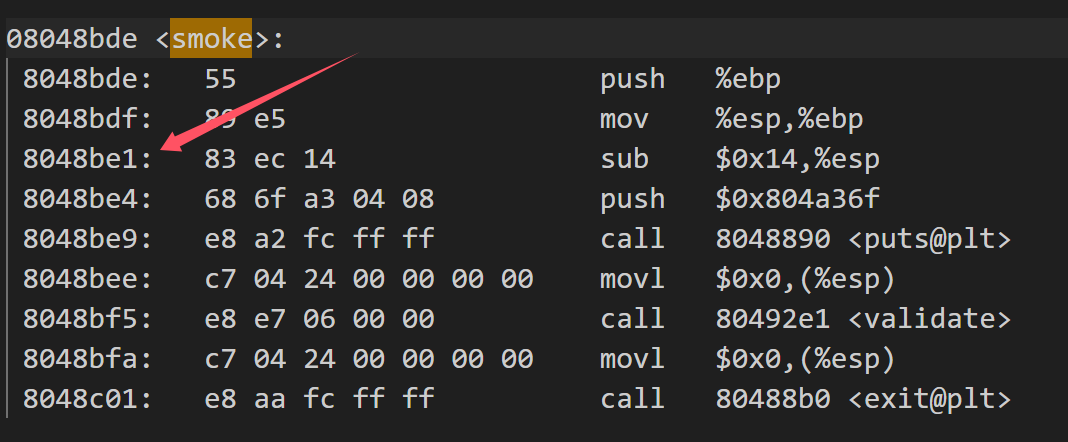
最后调用了exit，直接结束了

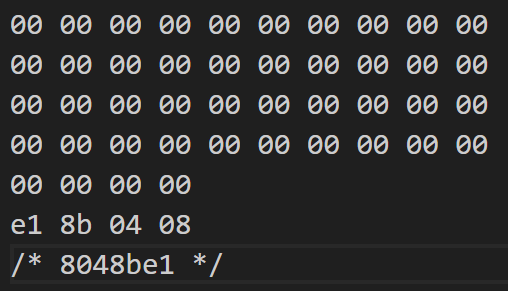
然后你找找，发现其他函数也有大概这一段的代码

然后你直接把返回地址写到sub那一句

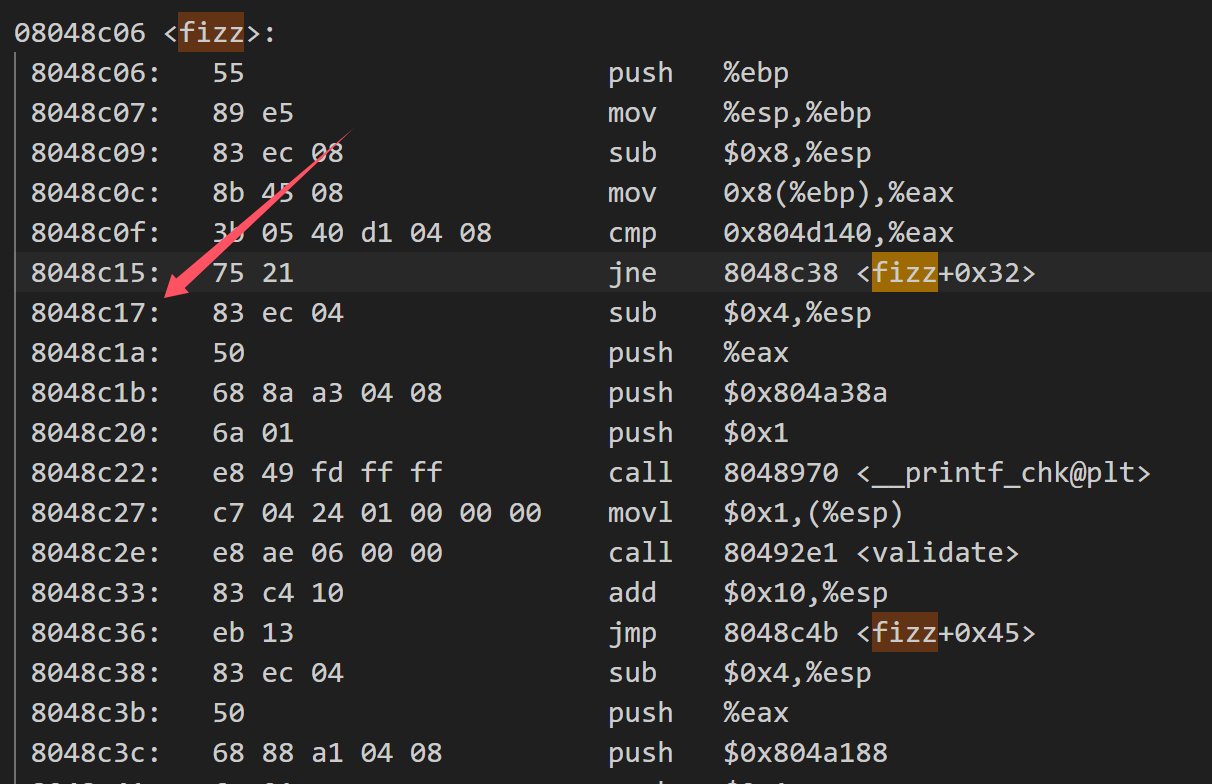
跳过前面的判断就行了

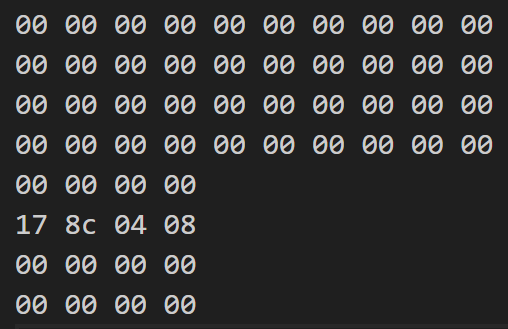
Smoke



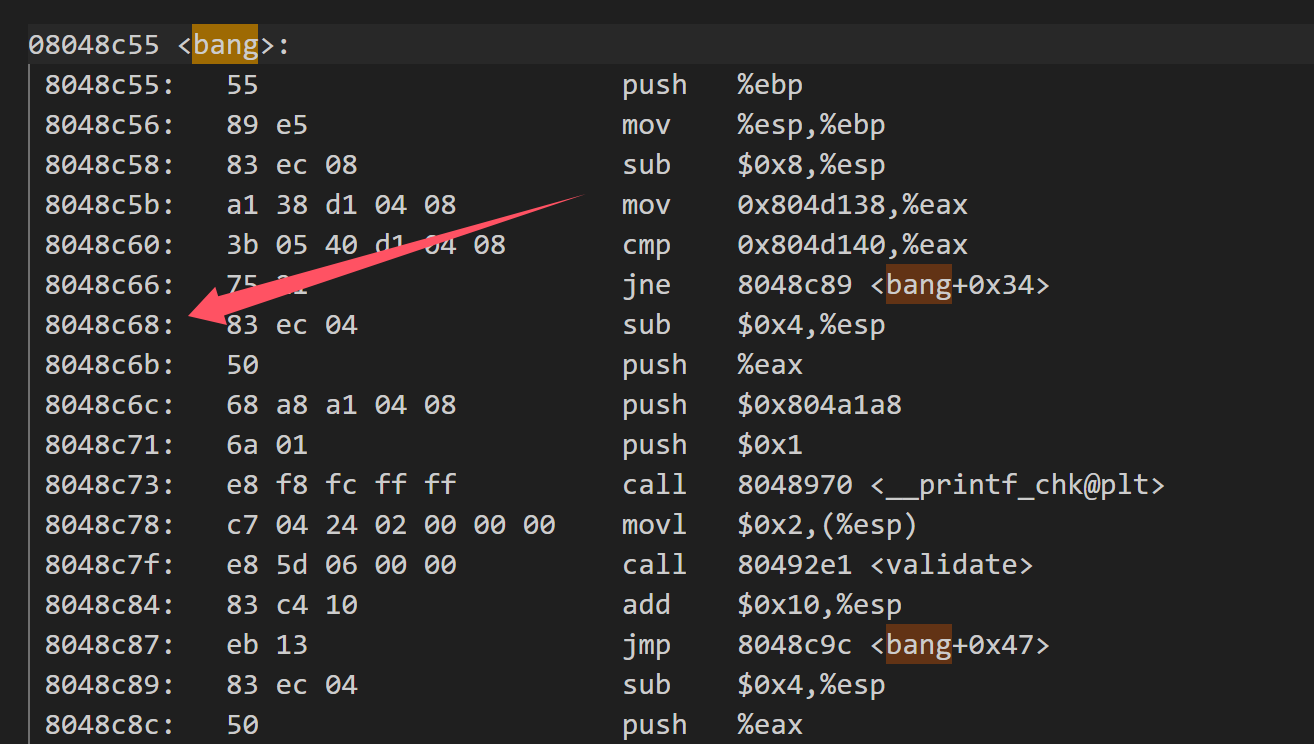
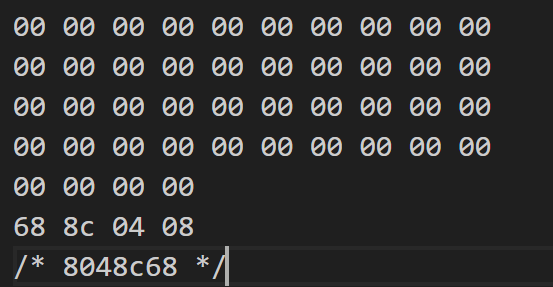


Fizz





Bang

. 

Boom

然后你就酷酷过了三题了

当到了这题

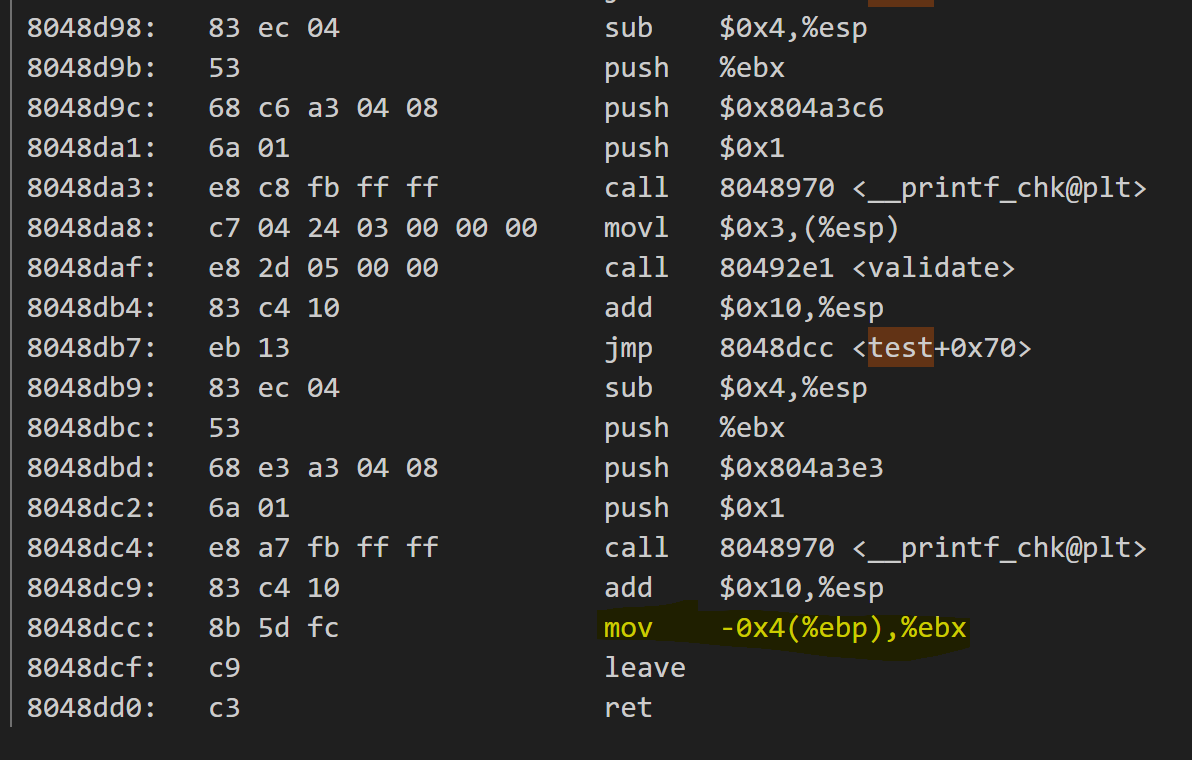
你一试

虽然发现输出了成功信息

但是发现引发了段错误

然后你发现提交上去是没成功的、

去查了一下



发现没有使用exit了

而且发现标黄的地方用了ebp的值

那么就会引发段错误

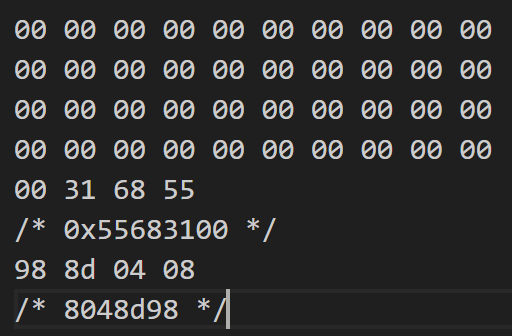
当然

这不代表逃课流失败了

还记得上面讲boom的时候吗

我们有两种方式可以复原ebp值

这里我们采用第一种方式直接从gdb里读出ebp的值，放在我们输的字符串里



然后这样又逃过了第四题了

Nitro

这第五题试了一种逃课（不用插代码的逃课）但没成功

具体方案如下

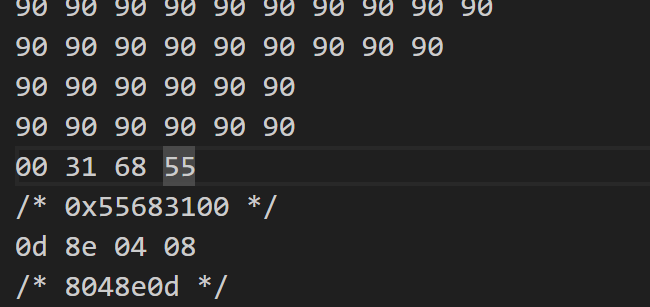
因为老师没有说第五题hex2raw要-n

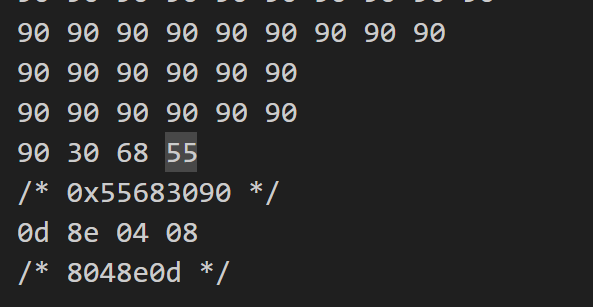
然后就有了我自己发现可以输5个字符串然后用换行隔开也就是0a

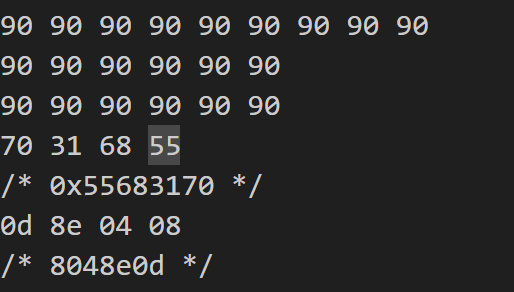
然后根据boom逃课的构造

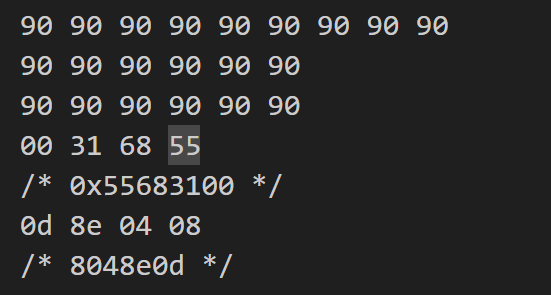
我们就可以输入5个不同的字符串

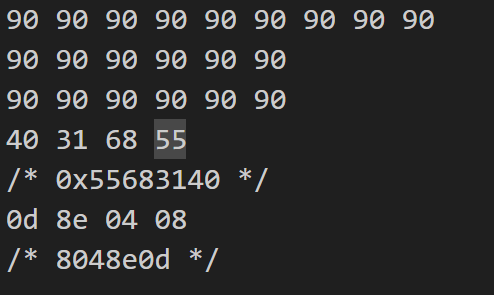
然后每次读出ebp的值放到新的字符串中











然后在本地跑是能成功

交上去就是invalid

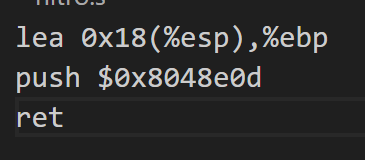
那估计是上面的栈随机化和本地的不一样吧

那现在只能插代码了

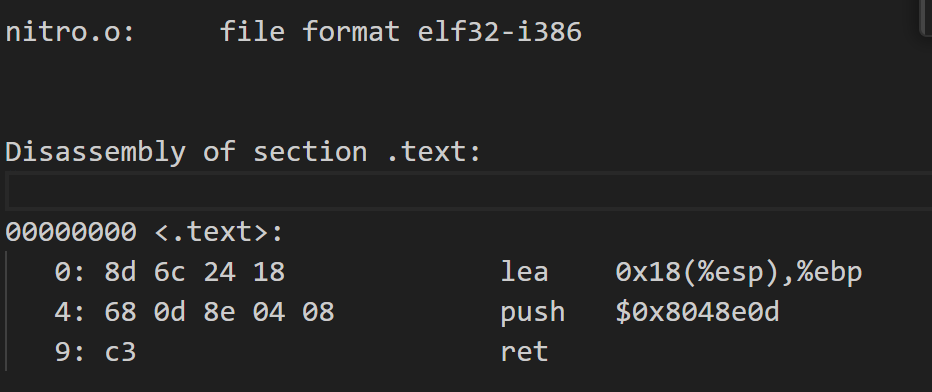
但是现在还可以半逃课

还是跳到sub那边

然后就可以插入以下代码

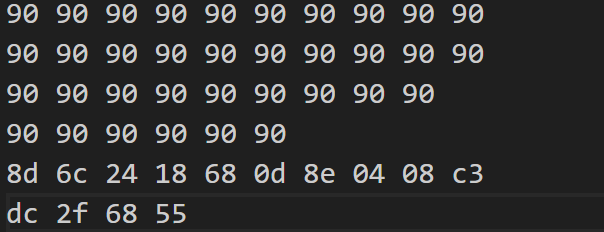


汇编反汇编就有



然后跟上面的nitro一样了

插入字符串就完事了



好

逃课流到这里就结束了

但是

还有其他的想法之类的（例如把恶意代码复刻为传数据的那一串

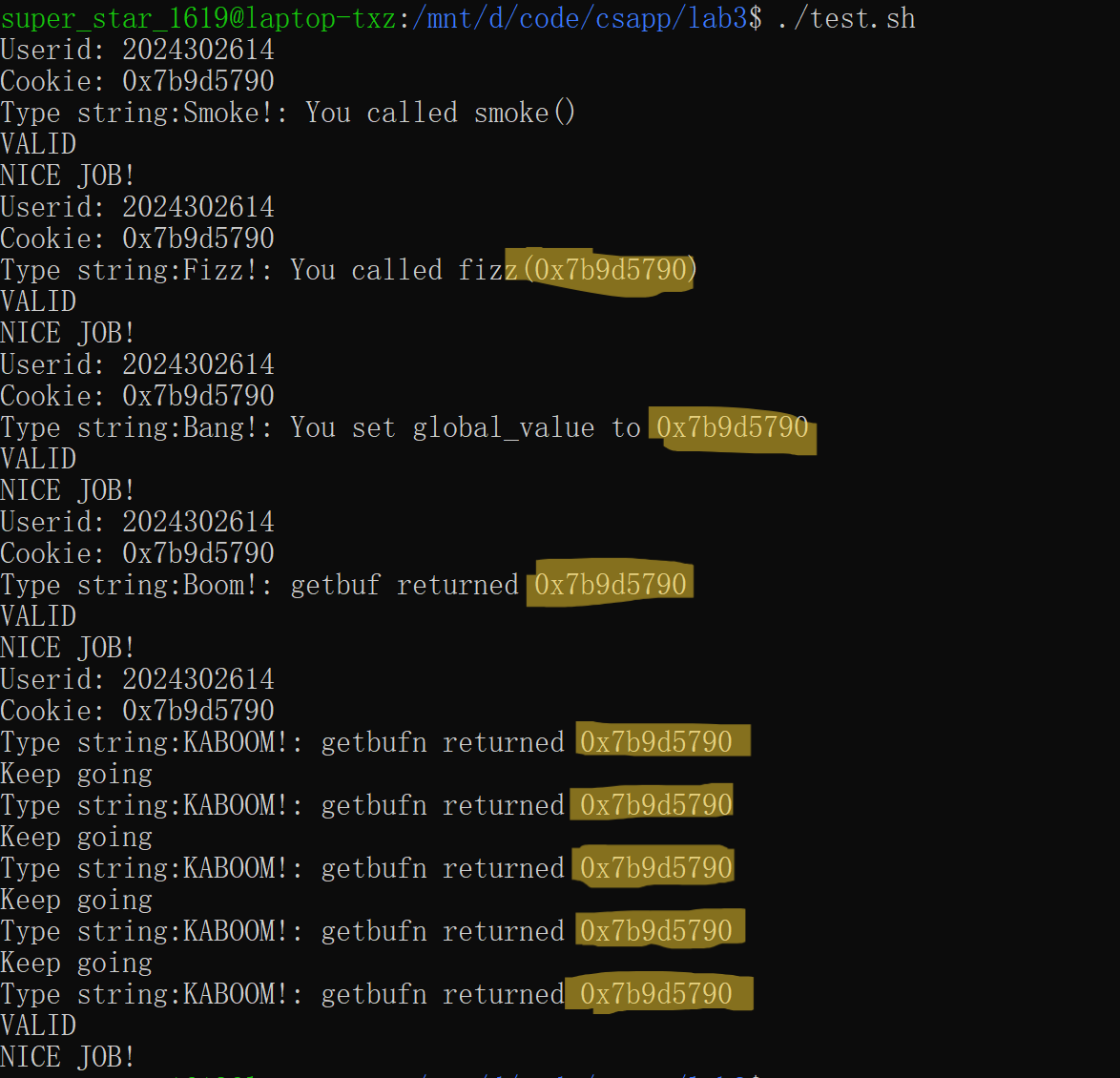
但是因为老是被吞字符

而且调试难度登天就没有尝试

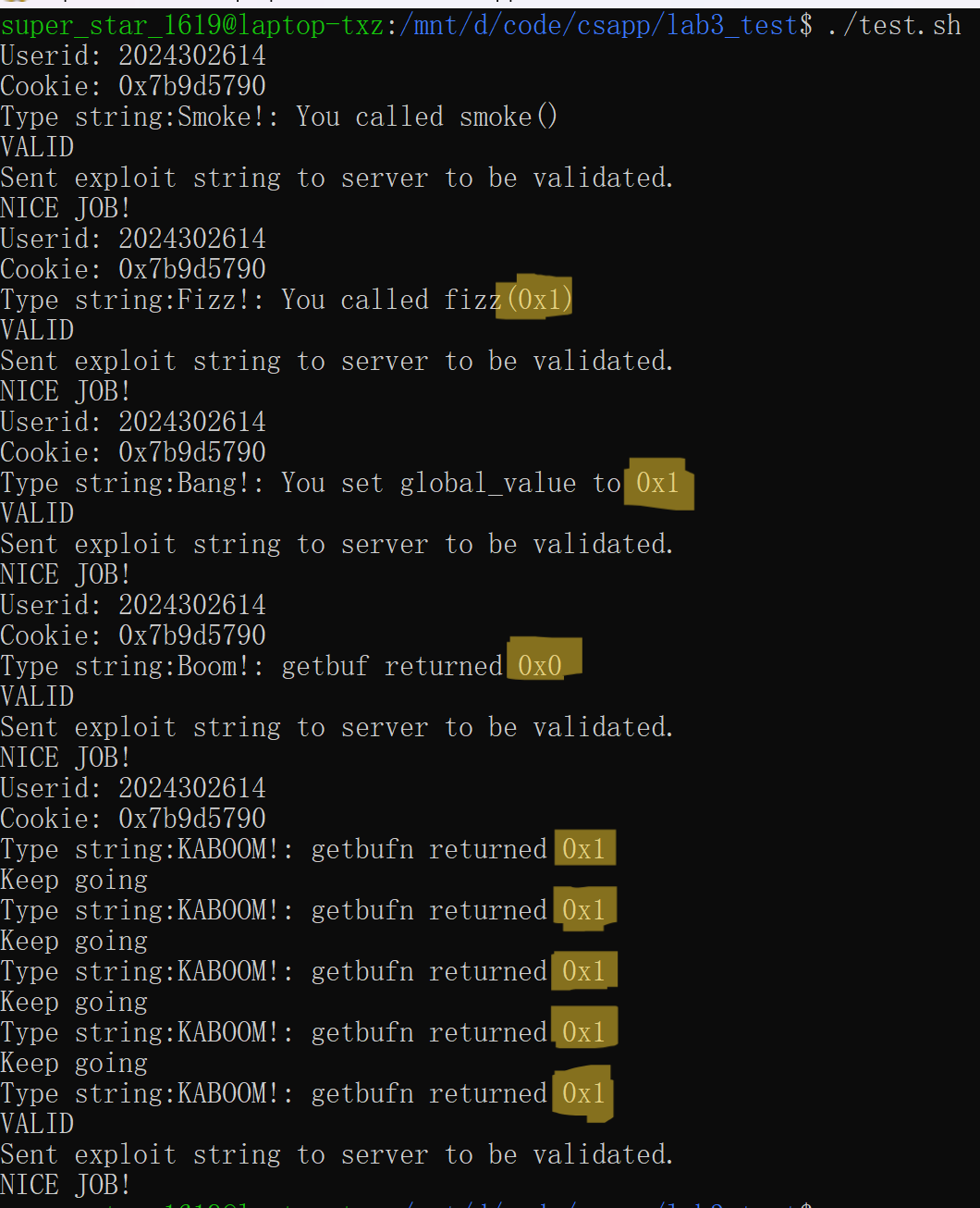
（反正有时间可以试试）

注：逃课流是有风险的

以下是正常做法的截图



以下是逃课流的截图



你会发现输出的这些结果很明显都是没改的

所以有风险

但是提交能成功就是了