

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称： 缓冲区溢出攻击**

**院 系 ：计算机科学与技术**

**专业班级 ： CS2201**

**学 号 ： U202215365**

**姓 名 ： 叶俊江**

**指导教师 ： 朱虹**

**2024 年 4 月 26 日**

**一、实验目的与要求**

通过分析一个程序（称为“缓冲区炸弹”）的构成和运行逻辑，加深对理论课中关于程序的机器级表示、函数调用规则、栈结构等方面知识点的理解，增强反汇编、跟踪、分析、调试等能力，加深对缓冲区溢出攻击原理、方法与防范等方面知识的理解和掌握；

实验环境：Ubuntu，GCC，GDB等

**二、实验内容**

**任务** 缓冲区溢出攻击

**程序运行过程中，需要输入特定的字符串，使得程序达到期望的运行效果。**

对一个可执行程序“bufbomb” 实施一系列缓冲区溢出攻击(buffer overflow attacks)，也就是设法通过造成缓冲区溢出来改变该程序的运行内存映像(例如将专门设计的字节序列插入到栈中特定内存位置)和行为，以实现实验预定的目标。bufbomb 目标程序在运行时使用函数 getbuf读入一个字符串。根据不同的任务，学生生成相应的攻击字符串。

实验中需要针对目标可执行程序bufbomb,分别完成多个难度递增的缓冲区溢出攻击(完成的顺序没有固定要求)。按从易到难的顺序，这些难度级分别命名为smoke (level 0)、fizz (level 1)、bang (level 2)、boom (level 3)和kaboom (level 4)。

**1、第0级 smoke**

正常情况下，getbuf函数运行结束，执行最后的ret指令时，将取出保存于栈帧中的返回（断点）地址并跳转至它继续执行（test函数中调用getbuf处）。要求将返回地址的值改为本级别实验的目标smoke函数的首条指令的地址， getbuf函数返回时，跳转到smoke函数执行，即达到了实验的目标。

**2、第1级 fizz**

要求getbuf函数运行结束后，转到 fizz函数处执行。与smoke的差别是，fizz函数有一个参数。 fizz函数中比较了参数val 与 全局变量cookie的值，只有两者相同（要正确打印val）才能达到目标。

**3、第2级 bang**

要求getbuf函数运行结束后，转到 bang 函数执行，并且让全局变量global\_value 与 cookie相同（要正确打印global\_value）。

**4、第3级 boom**

无感攻击，执行攻击代码后，程序仍然返回到原来的调用函数继续执行，使得调用函数（或者程序用户）感觉不到攻击行为。

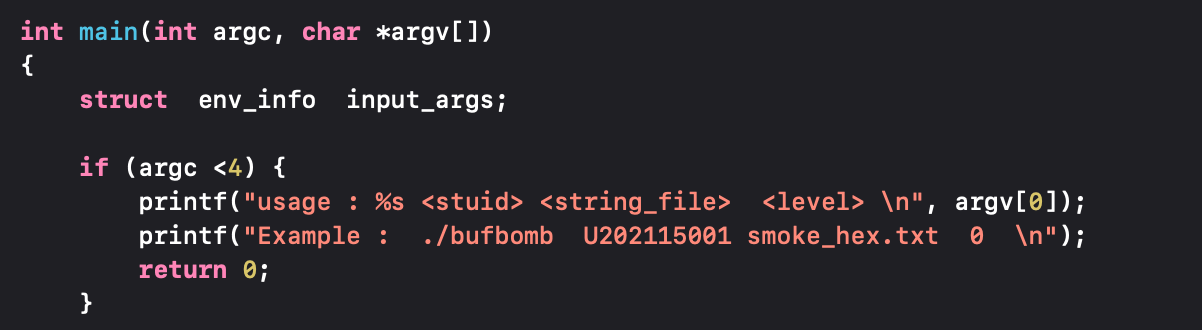
构造攻击字符串，让函数 getbuf将cookie值返回给 test函数，而不是返回值 1 。还原被破坏的栈帧状态，将正确的返回地址压入栈中，并且执行 ret 指令，从而返回到 test函数。

**三、实验记录及问题回答**

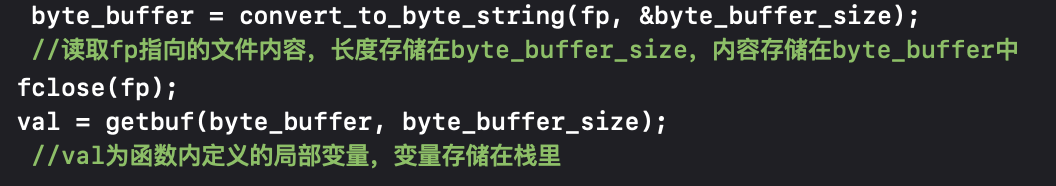
**（1）实验任务的实验记录**

**1、第0级 smoke**

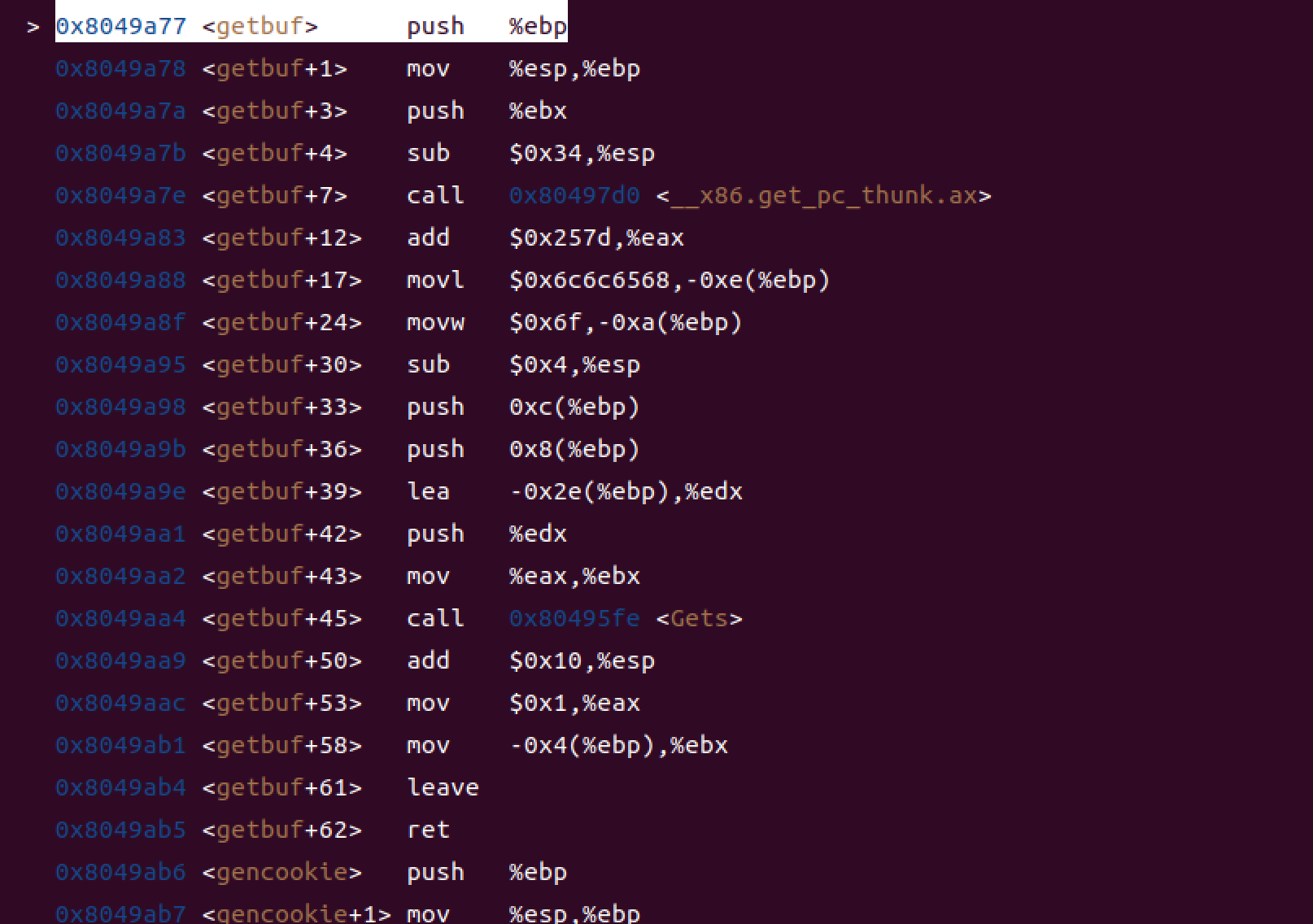
首先，命令行传参时，函数会对我们传入的参数进行检查，参数通过检查时才会继续进行函数。



即我们传入的参数要满足argc<4的时候才满足检查。随后进行initialize\_bomb(input\_args.userid)函数，传入我们输入的argv[1]参数进行初始化，随后又进行 test(&input\_args)函数进入测试。在test函数中注意以下函数语句。

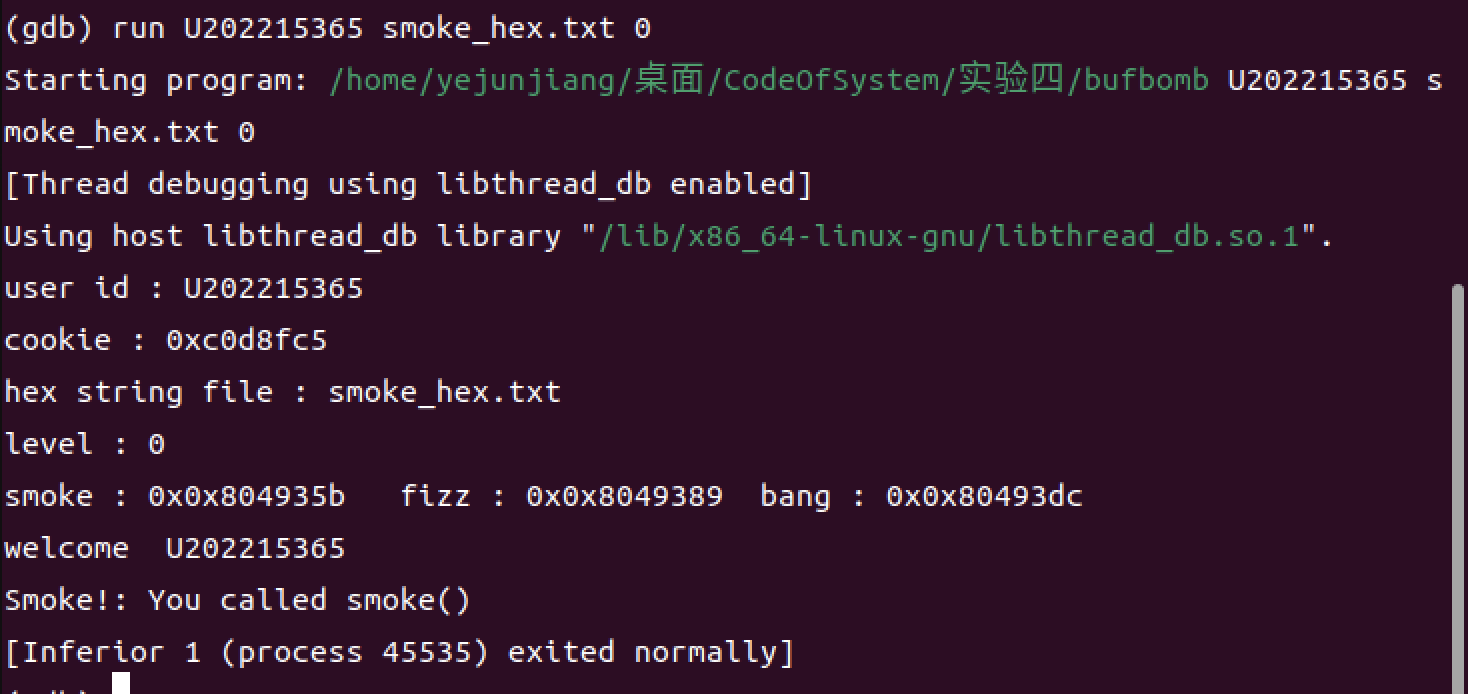


由此可见，本次实验中的关键点之一便是getbuf函数，那么我们调出汇编窗口查看getbuf函数体。



分析可知，我们在外部定义的smoke\_hex.txt文件里面需要从传入的dest开始的$ebp-0x32开始覆盖，需将之前传入getbuf之前保存的返回地址给覆盖，这样退出getbuf的时候才能跳转到smoke函数。然后由命令行运行时给出的信息可知smoke函数的地址为0x804975b，那么也就是说我们要将getbuf之前保存的地址覆盖成0x804975b即可。此时注意，我们采用小端存储的形式，故而需要将低字节存入低地址。

还有一点便是在getbuf函数中我们传入$ebp-0x2e的地址为起始的copy地址，这里起始地址和ebp的距离为0x2e，也就是46个字节，我们需要修改ebp下八字节，所以文件里面需要表示出54个字节，并且文件末尾放入smoke函数的地址即可。

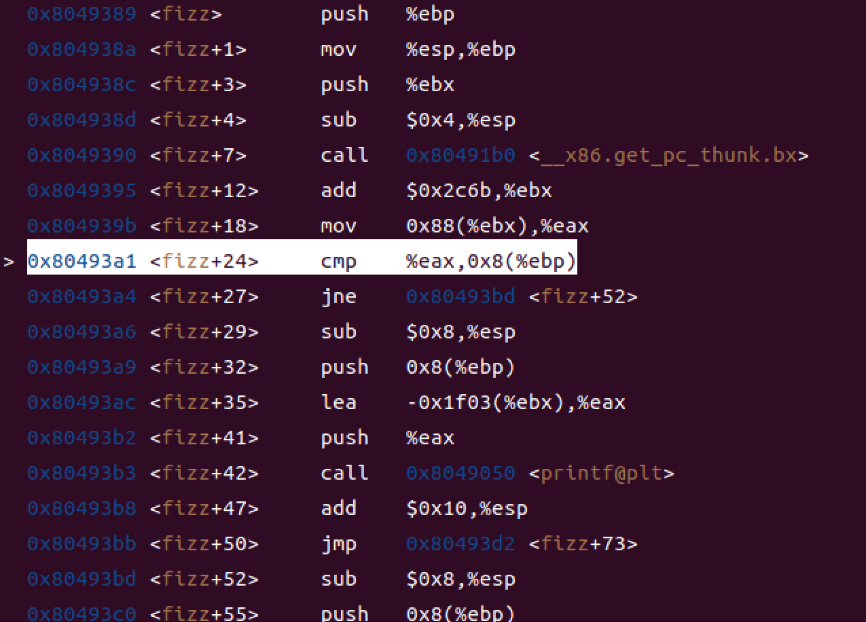


**2、第1级 fizz**

进入getbuf函数后，执行mov %esp,%ebp指令后，ebp的值变为0xffffceac，然后传入Gets函数的背拷贝地址为$ebp-0x2e，即0xffffce7e。

然后我们进入Gets函数分析，刚开始时ebp的值为0xffffcea8，在执行完mov %esp,%ebp指令后ebp的值变为0xffffce58，即此时esp的值为0xffffce58，在leave时esp的值会被还原为0xffffce58。

然后进入fizz函数体分析，函数与ebp之前的值无关，故而可以不恢复ebp的值。因为我们的参数存储在刚刚进入函数时的$esp+0x4中，也就是存储在执行完mov %esp,%ebp后的$ebp+0x8中，故而我们只需要将$ebp+0x8的位置赋值为cookie即可。

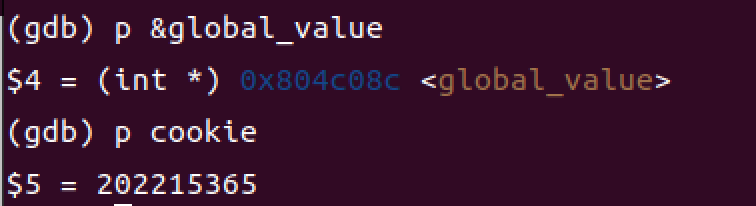


cmp %eax,0x8(%ebp)语句便是比较val与cookie的值，此时查看cookie的值为0xc0dbfc5，ebp+0x8的值为0xffffcea1。然后我们从文件中拷贝的字符会拷贝到地址0xffffce7e，而0xffffcea1-0xffffce7e=0x23=35，所以在文件fizz\_hex.txt中第35到38字节的数值写入cookie的值即可。

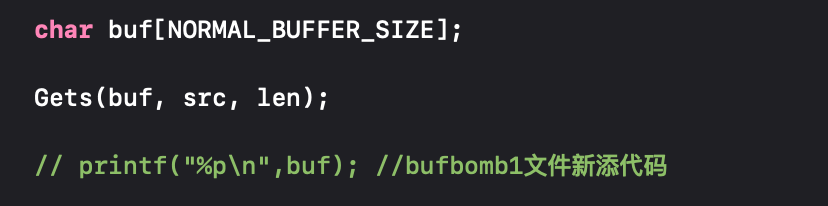


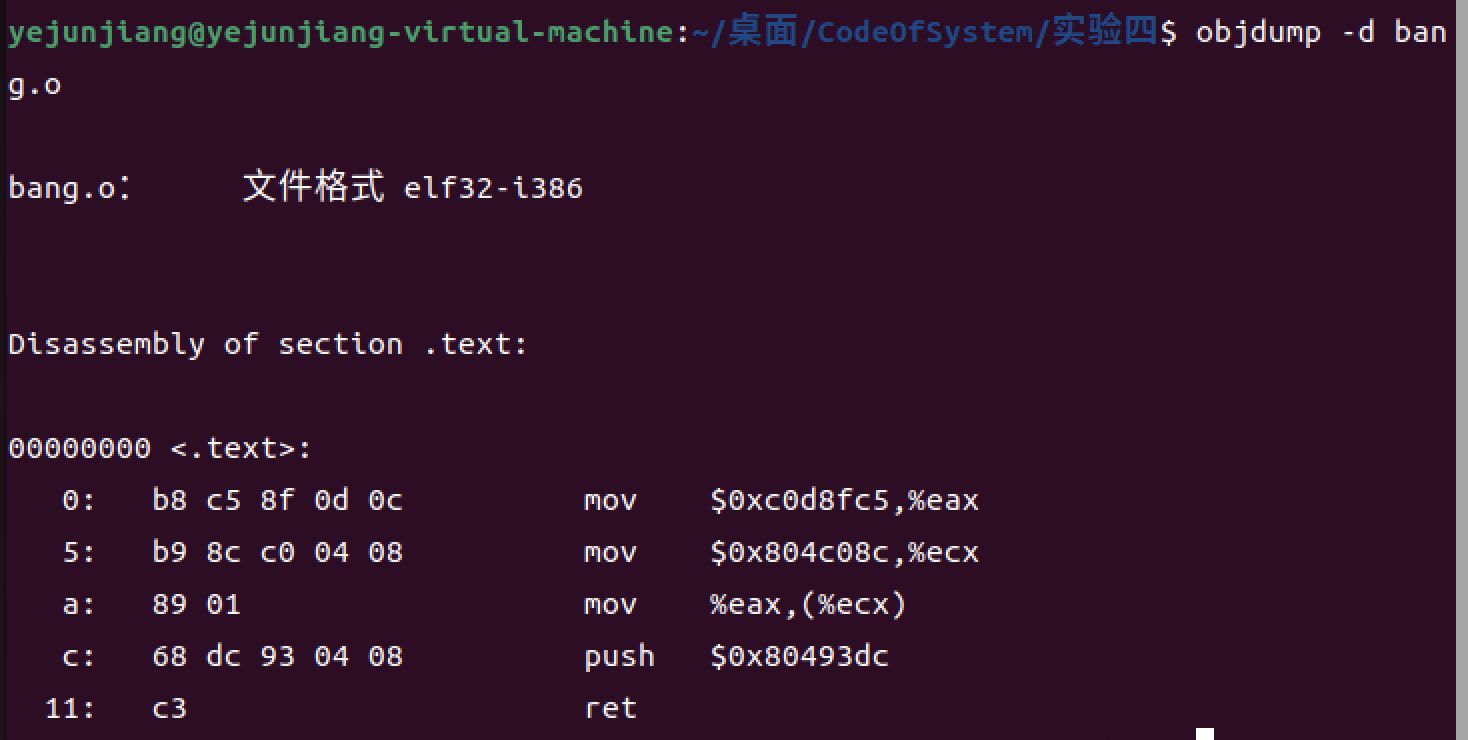
**3、第2级 bang**

首先打印输出global\_value地址和cookie的地址。



然后我们需要修改文件中的返回地址，使得我们可以跳转到bang函数，而bang函数的地址为0x80493dc。在编写bang.s文件时，第一步要将cookie值传给global\_value，然后就要处理ret返回地址的问题，第一种方法是通过esp指向地址，第二种方法是在对应编写的bang.s函数体后输入地址ret返回。我们采用第二种方法，发现一个问题：gdb调试与命令行运行时buf的首地址不一样，我们可以在源代码中写入printf(“\p\n”,buf)指令打印出buf的地址。获取buf地址后写入bang\_hex.txt文件中ret指令机器码后即可，这样我们就可以正常运行了。





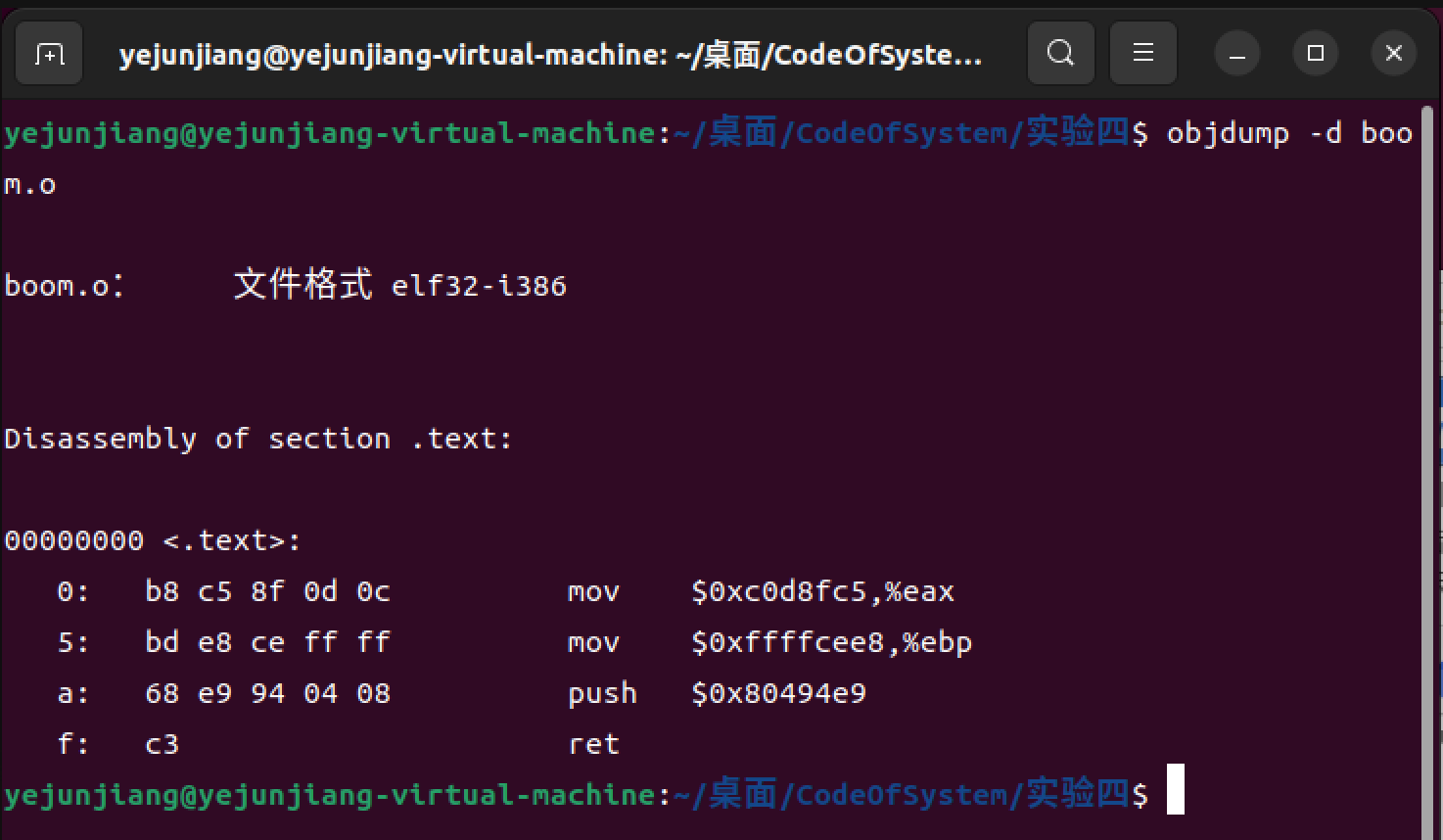


1. **第3级 boom**

这里我们要找到cookie的值，并且把cookie的值传给eax，由eax作为返回值返回，并且我们还要恢复ebp的值，故而也要查看我们要恢复ebp的值。

在刚进入getbuf函数时ebp的值为0xffffcee8，即我们需要复原ebp的值。并且查看返回地址$esp+0x4的值为0x80494e9，此为ret地址。然后cookie值与上一题一致为输入的学号202215365。

故编写的boom.o代码如下:



但是写入boom\_hex.txt文件运行时发现还是有段错误，其实这是ebx的值未恢复的问题，我们在push ebx处也要覆盖恢复ebx的值，故而在txt文件中第43到46字节中要写入ebx之前的值，至此结束。



**（2）缓冲区溢出攻击中字符串产生的方法描述**

**要求：一定要画出栈帧结构 （包括断点的存放位置，保存ebp的位置，局部变量的位置等等）**

**四、体会**