**深 圳 大 学 实 验 报 告**

**课 程 名 称： 计算机系统(2)**

**实验项目名称： 缓冲区溢出攻击实验**

**学 院： 计算机与软件学院**

**专 业： 计算机科学与技术**

**指 导 教 师： 马晨琳**

**报告人： 学号： 班级：**

**实 验 时 间： 2025年4月30日~5月13日**

**实验报告提交时间： 2025年5月13日**

**教务处制**

**一、 实验目标：**

1. 理解程序函数调用中参数传递机制；
2. 掌握缓冲区溢出攻击方法；
3. 进一步熟练掌握GDB调试工具和objdump反汇编工具。

**二、实验环境：**

1. 计算机（Intel CPU）
2. Linux 64位操作系统
3. GDB调试工具
4. objdump反汇编工具

**三、实验内容**

本实验设计为一个黑客利用缓冲区溢出技术进行攻击的游戏。我们仅给黑客（同学）提供一个二进制可执行文件bufbomb和部分函数的C代码，不提供每个关卡的源代码。程序运行中有3个关卡，每个关卡需要用户输入正确的缓冲区内容，否则无法通过管卡！

要求同学查看各关卡的要求，运用**GDB调试工具和objdump反汇编工具**，通过分析汇编代码和相应的栈帧结构**，**通过缓冲区溢出办法在执行了getbuf()函数返回时作攻击，使之返回到各关卡要求的指定函数中。第一关只需要返回到指定函数，第二关不仅返回到指定函数还需要为该指定函数准备好参数，最后一关要求在返回到指定函数之前执行一段汇编代码完成全局变量的修改。

实验代码bufbomb和相关工具（sendstring/makecookie）的更详细内容请参考“实验四 缓冲区溢出攻击实验.pptx”。

本实验要求解决关卡1、2、3，给出实验思路，通过截图把实验过程和结果写在实验报告上。

**四、实验步骤和结果**

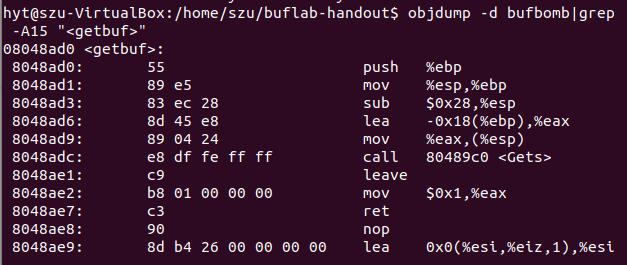
**因为本次实验用到的可执行文件是32位，而实验环境是64位的，需要先安装一个32位的库，在root权限下安装如下所示：**



**还需要安装sendmail**



首先利用反汇编命令查看getbuf函数的汇编代码，以便分析getbuf在调用<Gets>时的栈帧结构，汇编代码如下：



**步骤1 返回到smoke()**

* 1. **解题思路**

本实验中，bufbomb中的test()函数将会调用getbuf()函数，getbuf()函数再调用gets()从标准输入设备读入字符串。

系统函数gets()未进行缓冲区溢出保护。其代码如下：

***int getbuf()***

***{***

***char buf[12];***

***Gets(buf);***

***return 1;***

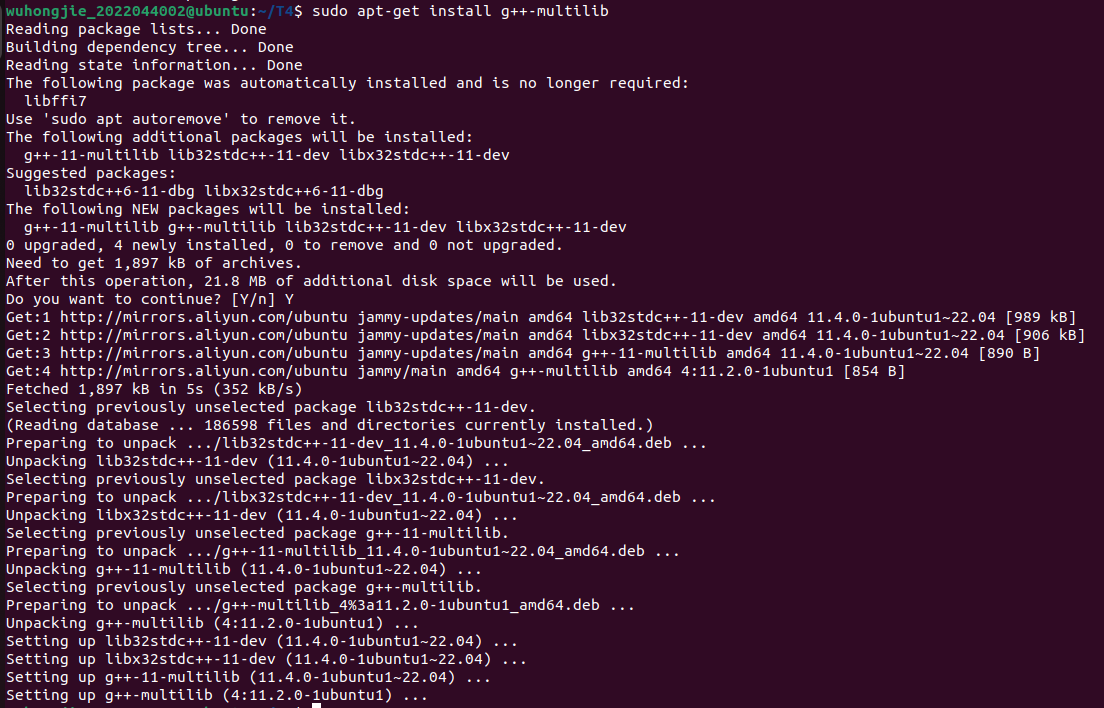
***}***

我们的目标是使getbuf()返回时，不返回到test()，而是直接返回到指定的smoke()函数。

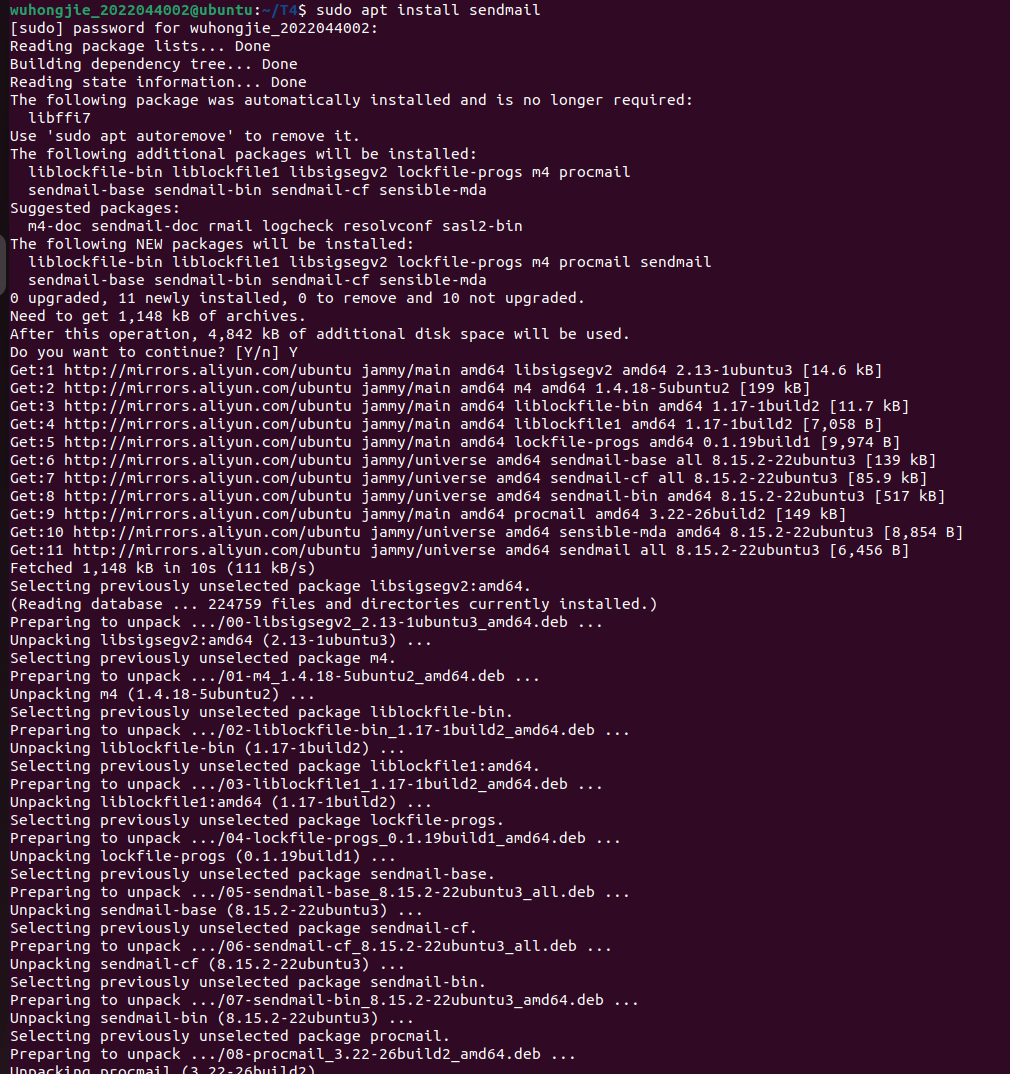
为此，我们可以通过构造并输入大于getbuf()中给出的数据缓冲区的字符串而破坏getbuf()的栈帧，替换其返回地址，将返回地址改成smoke()函数的地址。

* 1. **解题过程**

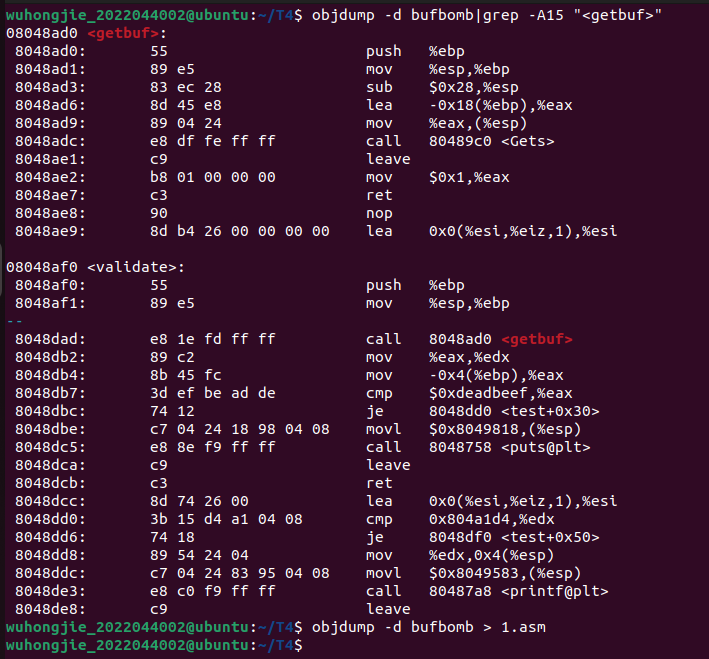
1.安装库



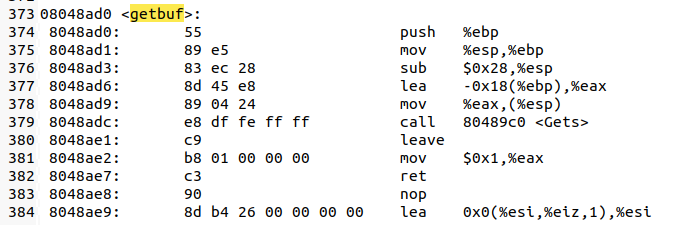
2.安装sendmail



3.生成反汇编文件：



4.查看getbuf函数

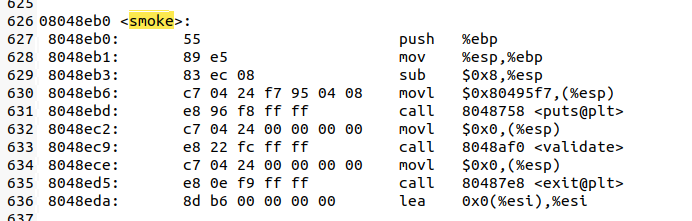


阅读可知，该函数没有缓冲区溢出保护。

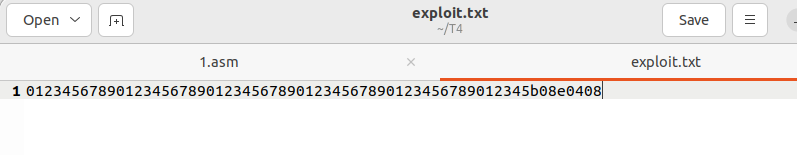
377行ebp-0x18的地址为Gets函数的参数，Gets函数将以该地址为起点向地址增大的方向保存字符，共0x18（24）个字节，即buf缓冲区为0x18（24）个字节，因此需要将 getbuf 函数的返回地址覆盖为smoke函数第一条语句的地址。

按照要求，我们得让getbuf 函数的返回地址覆盖为smoke函数第一条语句的地址。即输入24+4=28个字节没用的东西，再加上smoke的地址，我们可以直接看到smoke地址开头的地址：

即08 04 8e b0

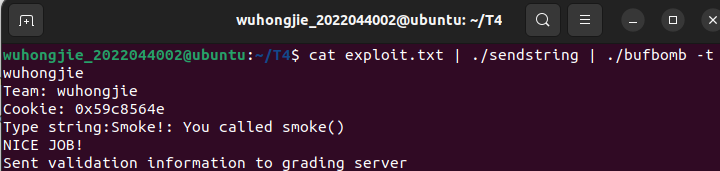


4.创建文件exploit.txt，用56个随意的数字加上用小端法表示的smoke函数地址，如图所示：



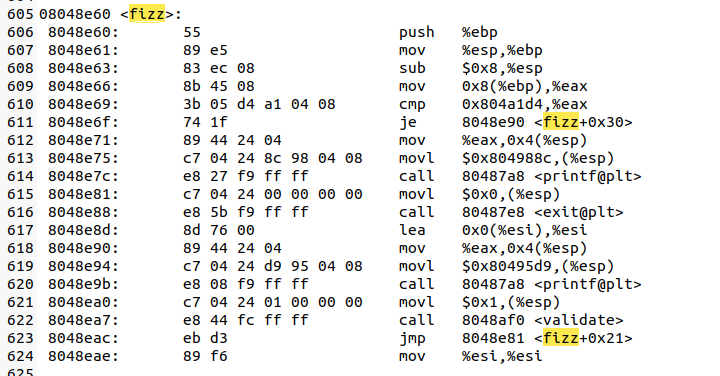
5.执行，获得结果

**1.3 最终结果截图**



**步骤2 返回到fizz()并准备相应参数**

查看并分析fizz()函数的汇编代码：

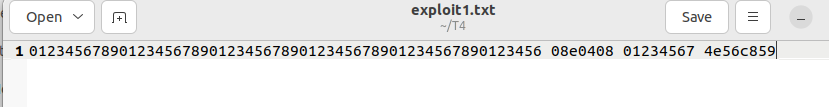


从汇编代码可知，fizz函数被调用时首先保存%ebp旧值并分配新的空间，然后读取%ebp-0x8地址处的内容作为传入的参数，要求传入的参数是自己的cookie值。所以，可以对应到getbuf函数中有如下这样的结构：

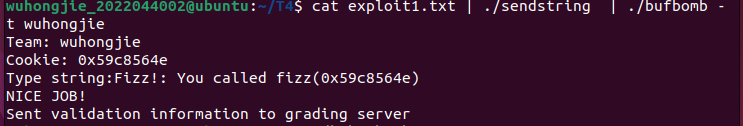
|  |  |
| --- | --- |
| **栈帧** |  |
|  | 需要替换成cookie传入fizz() |
|  | 任意替换 |
| 返回地址 | 属于调用者的栈帧 |
| 保存的%ebp旧值 | %ebp，需要替换成fizz()的地址 |
|  | 任意替换 |
|  | 任意替换 |
|  | 任意替换 |
| [11][10][9][8] |  |
| [7][6][5][4] |  |
| [3][2][1][0] | buf,%ebp-0x18 |
|  |  |
|  |  |
|  |  |
|  | %esp，%ebp-0x24 |

由此判断，我们可以判断输入28个任意字节+fizz函数的地址+4个任意的字节+自己的cookie值，前边的图已经得知fizz函数的地址为08 04 8e 60，我的cookie为0x59c8564e。

所以，按照此输入：

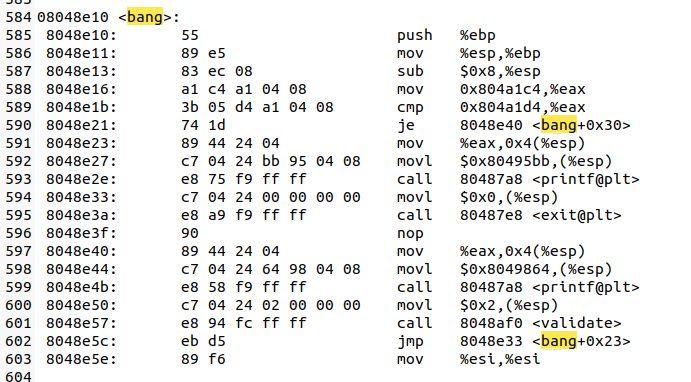


运行，得到成功的结果：

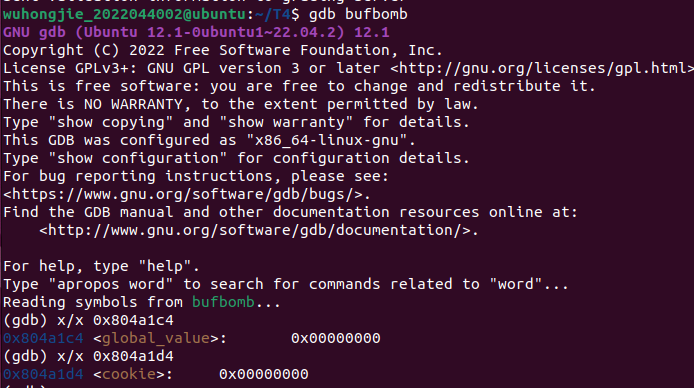


**步骤3 返回到bang()且修改global\_value**

1.找到bang函数的地址：

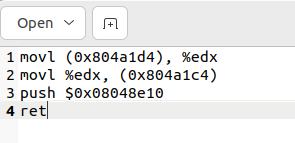


为08 04 8e 10。观察代码，发现函数首先读取0x804a1c4和0x804a1d4的地址的内容并进行比较，要求两个地址中的内容相同。我们再用gdb查看他们俩：



一个是global\_value，一个是cookie，所以这就说明我只要在自己写的代码里把自己的cookie写进global\_value里就行了。然后与前边的题类似地，把bang函数的入口压入栈中使其被调用。

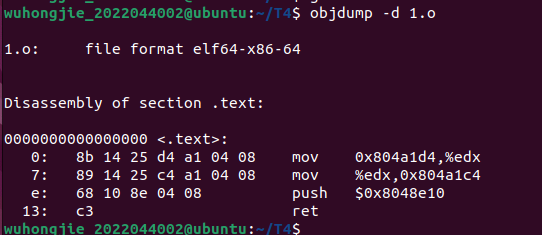
2.写上述功能的代码：



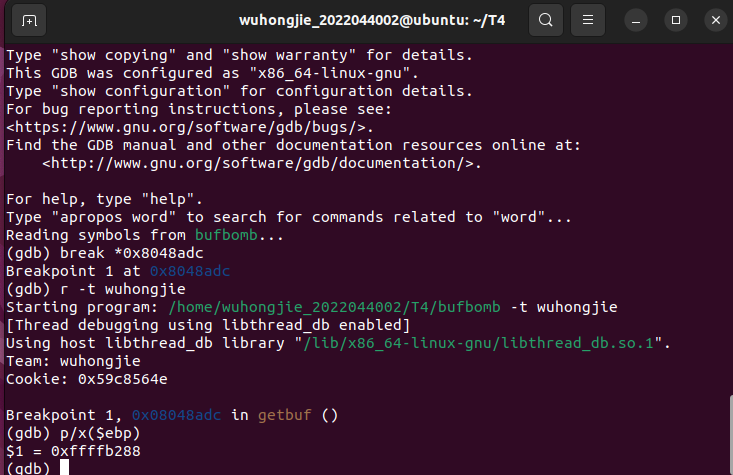
编译，转为机器码：



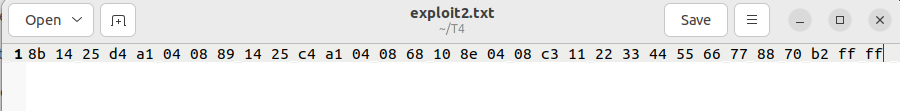
用objdump查看机械代码：



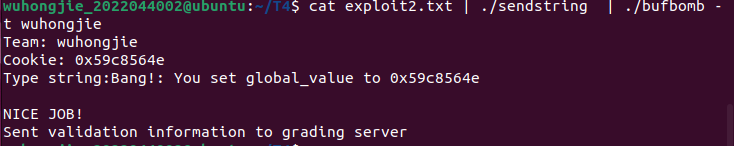
所以，最后我们要构造的字符串为自己写的汇编代码生成的机器码(20个字节)+8个任意字节+buf数组的首地址，我们查看：



打上getbuf()断点发现ebp寄存器内的地址是0xffffb288，则buf的数组为ebp-0x18=0xffffb270,故构造数如下：



执行，发现成功：



**五、实验总结与体会**

这次实验让我对缓冲区溢出攻击有了更深入的了解。通过分析汇编代码和栈帧结构，我明白了攻击的原理和步骤。虽然过程有些复杂，但通过查阅资料和反复尝试，最终我还是成功完成了实验。

通过分析getbuf函数的汇编代码，我掌握了函数调用栈的结构 —— 从ebp寄存器定位缓冲区位置（ebp-0x18），到理解Gets函数因缺乏边界检查导致的溢出风险，清晰认识到栈帧中返回地址被篡改的关键机制。这让我对程序执行时内存空间的分配、函数调用的参数传递有了直观认知，深刻体会到汇编语言作为 “桥梁” 对理解高级语言与底层硬件交互的重要性。

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  指导教师签字： 马晨琳    2024年 5月 日 |
| 备注： |