**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（创新班）**

**指 导 教 师： 刘刚**

**报告人： 何泽锋 学号： 2022150221 班级： 高性能特色班**

**实 验 时 间： 2024年5月25日~5月26日**

**实验报告提交时间： 2024年5月26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。

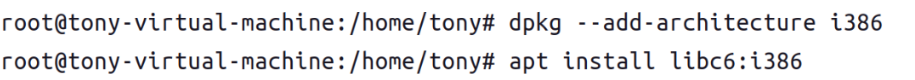
实验代码bufbomb和相关工具（sendstring/makecookie）的更详细内容请参考“实验四 缓冲区溢出攻击实验.pptx”。

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**

**实验预处理**

1)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，因为lib32ncureses5库在新版本ubuntu中已经删除，此处使用其他库替代，安装指令如下

****

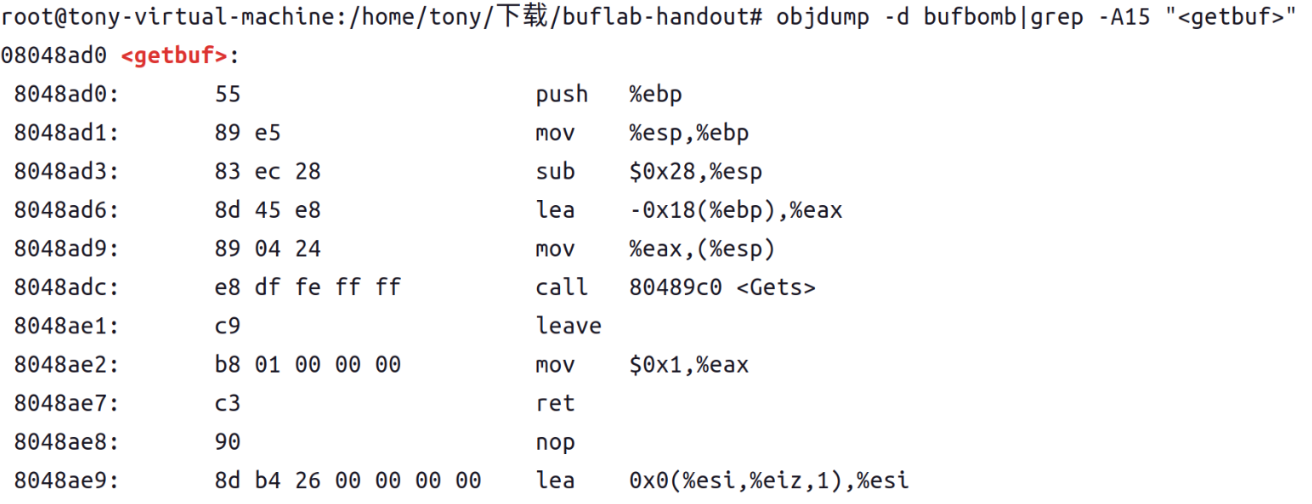
2)继续安装sendmail

****

**关卡1 返回到smoke()**

1)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。

2)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：

****

3)将汇编代码写成c语言形式，发现系统函数gets()未进行缓冲区溢出保护。其代码如下：

*int getbuf(){*

*char buf[12];*

*Gets(buf);*

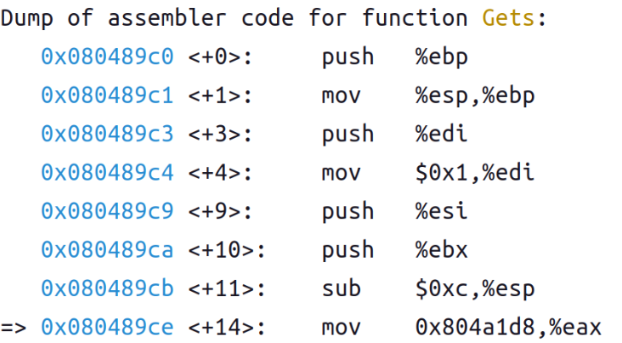
*return 1;*

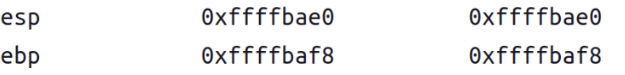
*}*

4)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。

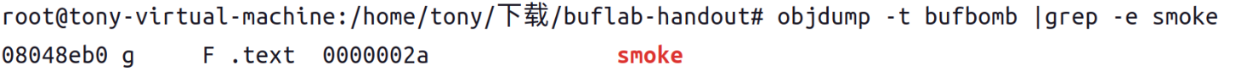
5)分析地址空间，可以知道首先在getbuf()开设了0x28大小的栈，即%ebp-0x28（%esp-40），进入从开设的空间中取出0x18给buf数组,此时位置为%ebp-0x18（%esp-24）

6)运行Gets至此，此时观察寄存器中的值可以发现，%ebp与%esp刚好相差0x18





7)查询smoke()的地址，方便后续使用



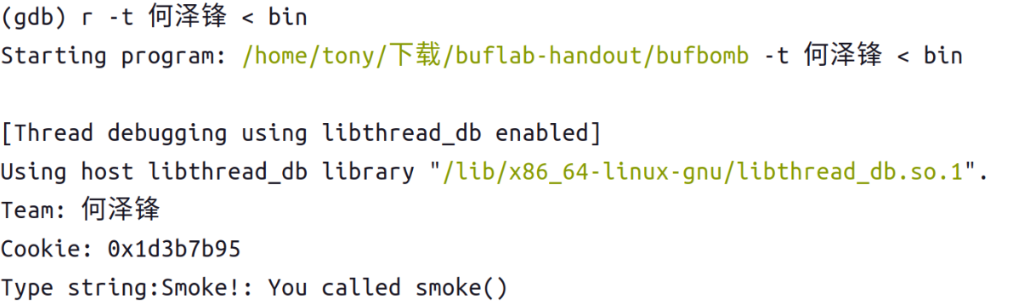
8)图解表示如下,地址中首先有部分空白的地方，开始存入数据的位置为%ebo-0x18，已知每个地址单元大小为8位，因此要修改返回值需要将前面的内容都填满直到返回值存放位置。已知一位16进制的数占4位，因此使用16进制的二位数刚好占8位（1个字节），此处共需要占24+4个字节，因此在改后的地址前要输入28个16进制的二位数。要注意返回值要先输入低位再输入高位，因为栈的存储方式是高位在上。



9)构建答案文档如图所示：

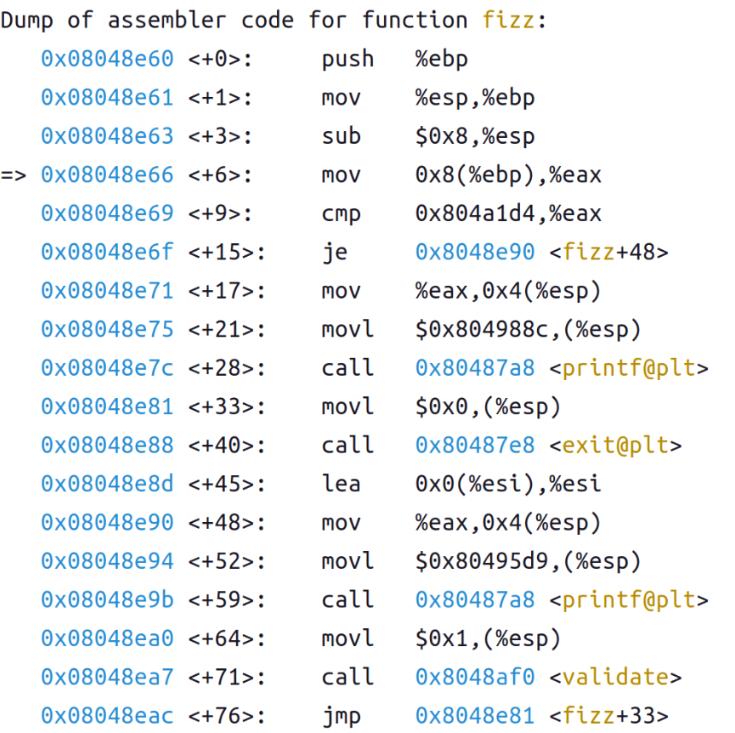


10)结果验证，将个人信息和答案输入后成功跳转到smoke()

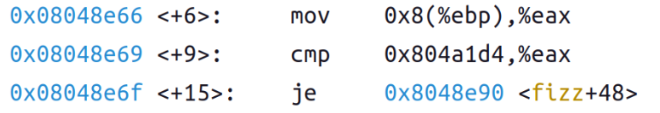


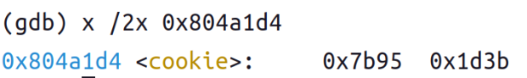
**关卡2 返回到fizz()并准备相应参数**

1)修改%eip寄存器至fizz()处，使用指令参看汇编代码

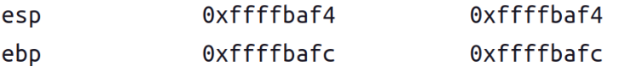
****

2) 阅读代码可知，此处进行了比较，一个是栈中的数据，即输入的部分数据。另一个是内存中的数，查看该部分数据，该数据为自己的cookie，因此可知，输入的数据要包含自己的cookie。





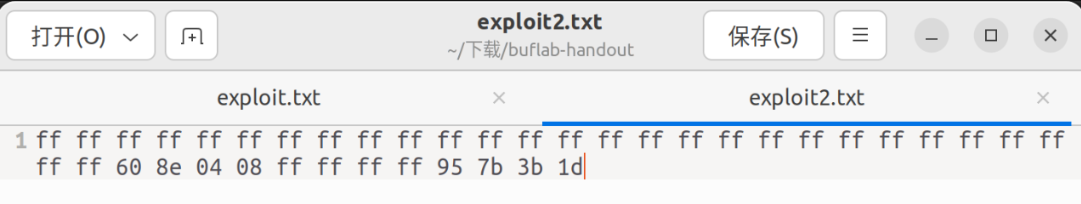
3)查看比较时%ebp指向的位置，进而可以确认是输入的哪一个数据开始存放cookie，结合前面的Gets()时的%ebp位置可以分析得知，存放cookie的位置在返回值加四个字节之后。



4)图表表示如下，因此输入的答案为28个ff+fizz()地址+ff ff ff ff+cookie



5)构建答案文档如图所示：

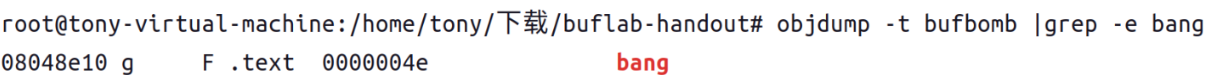


6) 结果验证，将个人信息和答案输入后成功跳转到fizz()



**关卡3 返回到bang()且修改global\_value**

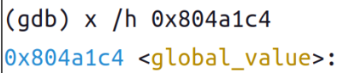
1)首先查询bang()的地址，方便后续跳转时使用



2)查看bang()的反汇编代码



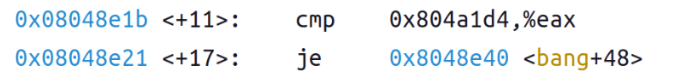
3)可以看到此处主要包括两个地址，分别为0x804a1c4和0x804a1d4，检查0x804a1c4内存地址发现存放的是一个<global\_value>，当前尚未有数据存入



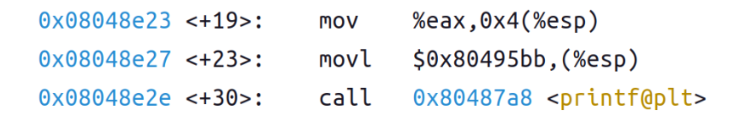
4)检查0x804a1d4，发现此处存放的是自己的<cookie>

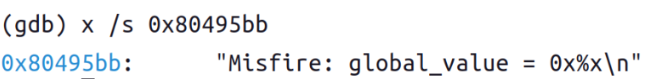


5)将这两个地址的内容进行比较，若相等则跳转

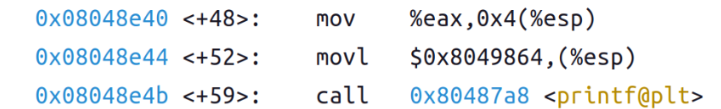


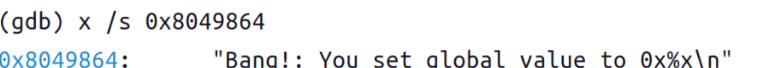
6)输出该全局变量的内容





7)若global\_value与自身cookie相等则输出该值，此处尚未输入，仅展示输出的语句

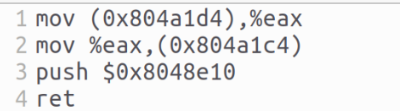




8)结合以上代码分析可知，bang()是比较<global\_value>与<cookie>是否相等，但在函数内并没有对<global\_value>的值进行修改，因此需要在跳转到bang()前将<global\_value>的值设置为<cookie>

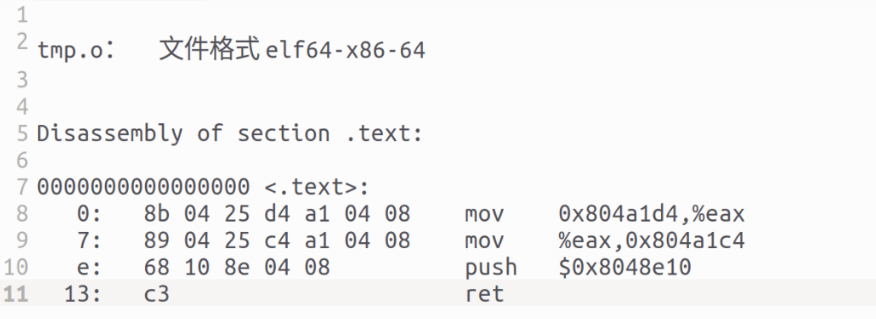
9)此处手动输入赋值的汇编代码，方便后续转为机器码得到16进制内容，代码的意思是将<cookie>的内容放入<global\_value>对应的地址，然后将bang()的起始地址入栈，作为返回值



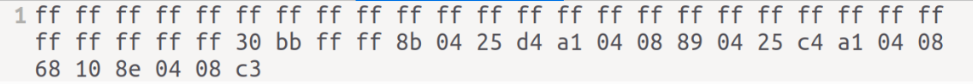


10)将刚刚输入的汇编代码转为16进制机器码，具体代码如下

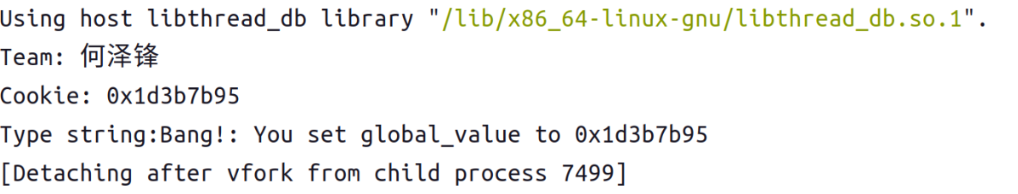




11)构造答案，首先需要使用28个ff占位使得能够修改<Gets>的返回值，此处返回值指向自己输入的代码的首地址，然后接着将自己的代码往后存放。



12)将构造好的答案转成二进制文件在gdb调试的时候进行输入，可以看到此处将<global\_value>设置成了自己的<cookie>

****

**五、实验总结**

通过本次实验基本了解了如何使用缓冲区溢出进行攻击，本次实验攻击都是基于buf数组进行，通过往非数组空间里存放数据使得返回值发生了改变，进而进入不同的函数，或者是调用自己的代码。经过本次实验，对数据存储有了更清晰的了解，在x86中，数据是小端存储，即数据低位存在低地址，因此实验时要注意输入数据和存储数据的顺序刚好是相反的。通过实验也了解到了缓冲区的重要性，要严格注意缓冲区溢出的问题，否则程序很有可能会受到攻击

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