**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： 刘刚**

**报告人： XXX 学号： XXXXXXXX 班级： 高性能班**

**实 验 时 间： 2024年5月21日~5月25日**

**实验报告提交时间： 2024年5月25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。

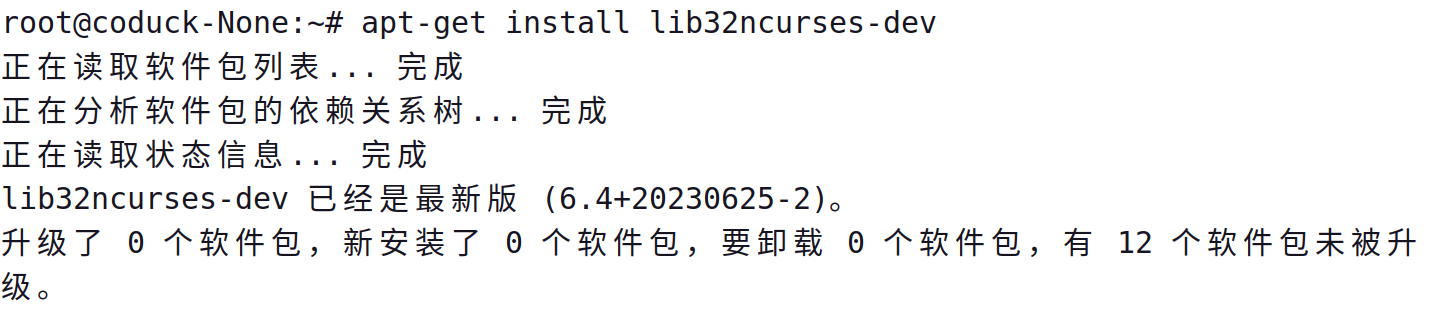
实验代码bufbomb和相关工具（sendstring/makecookie）的更详细内容请参考“实验四 缓冲区溢出攻击实验.pptx”。

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**

**1、安装可执行文件所需库**

安装lib32ncurses-dev、lib32z1、sendmail。



**图 1 安装lib32ncurses-dev**

文本

描述已自动生成

**图 2 安装lib32z1**

图形用户界面, 文本, 应用程序

描述已自动生成

**图 3 安装sendmail**

**2、用objdump获得bufbomb的反汇编文件**

**图 4 获得bufbomb的反汇编文件**

**3、返回到smoke()**

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。

**3.1、解题过程**

分析getbuf汇编代码，可以看到在调用Gets函数前向%esp所指向的地址处存入%ebp-0x18，这可能是为调用Gets所准备的参数，根据Gets的作用，我们可以猜测%ebp-0x18地址为输入时缓冲区的起始地址。

表格

低可信度描述已自动生成

**图 5 分析getbuf汇编代码**

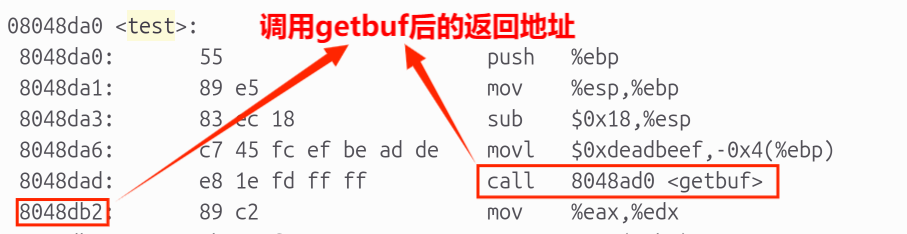
在0x8048ad9和0x8048ae1处设置断点，用gdb调试bufbomb，查看条用Gets函数前后getbuf过程栈帧的变化情况。



**图 6 调试getbuf汇编代码**

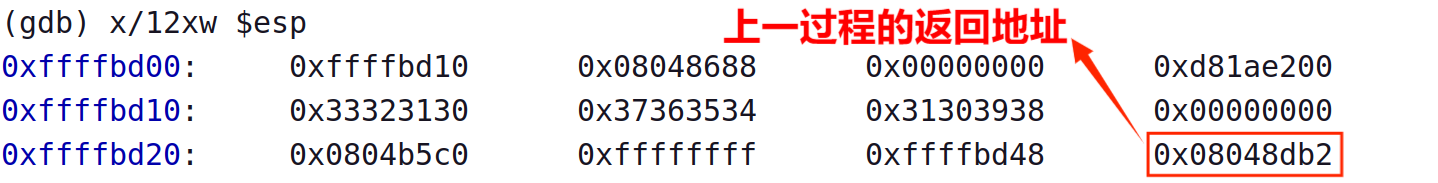
可以看到，%ebp-0x18的地址为0xffffbd10，同时在输入字符串“012345678901”（对应的十六进制数值为“0x30313233 0x34353637 0x38393031”，但因为此处是小端存储，读取数据时是从低地址开始，所以输出对应的十六进制为“0x33323130 0x37363534 0x31303938”），从0xffffbd10开始能看到我们刚刚输入字符串的在内存中的值，所以可以肯定，%ebp-0x18即0xffffbd10为输入缓存区的起始地址。

同时我们去查看调用完getbuf后返回到test的地址，可以看到是0x8048db2。



**图 7 检查getbuf返回地址（1）**

回到getbuf的栈帧处，可以看到在0xffffbd2c地址处存有调用完getbuf后返回到test的地址，由此我们可以确定了返回地址的位置。



**图 8 检查getbuf返回地址（2）**

综上，我们知道，只有在输入缓存区起始地址0xffffbd10后28字节处0xffffbd2c用smoke的起始地址覆盖原本的返回地址，才能使得getbuf返回时返回到smoke，即输入内容十六进制值为任意28字节的内容+08048eb0，由于是小端存储，所以输入内容十六进制值实际为任意28字节的内容+0xb08e0408。



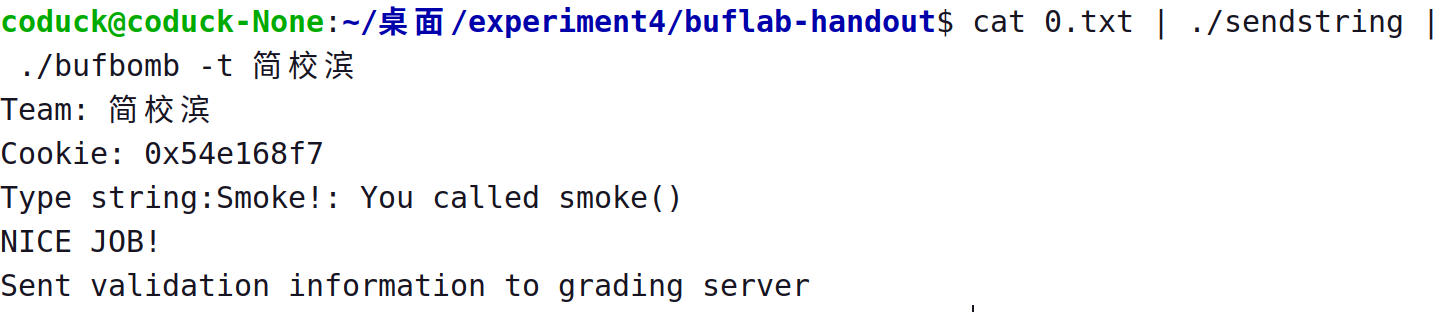
**图 9 查看smoke起始地址**

将输入内容十六进制值

00112233445566778899001122334455667788990011223344556677b08e0408

存入文件0.txt，通过sendstring工具将以上内容转换为字符串再传入bufbomb程序，即可成功返回到smoke。

**3.2、最终结果截图**

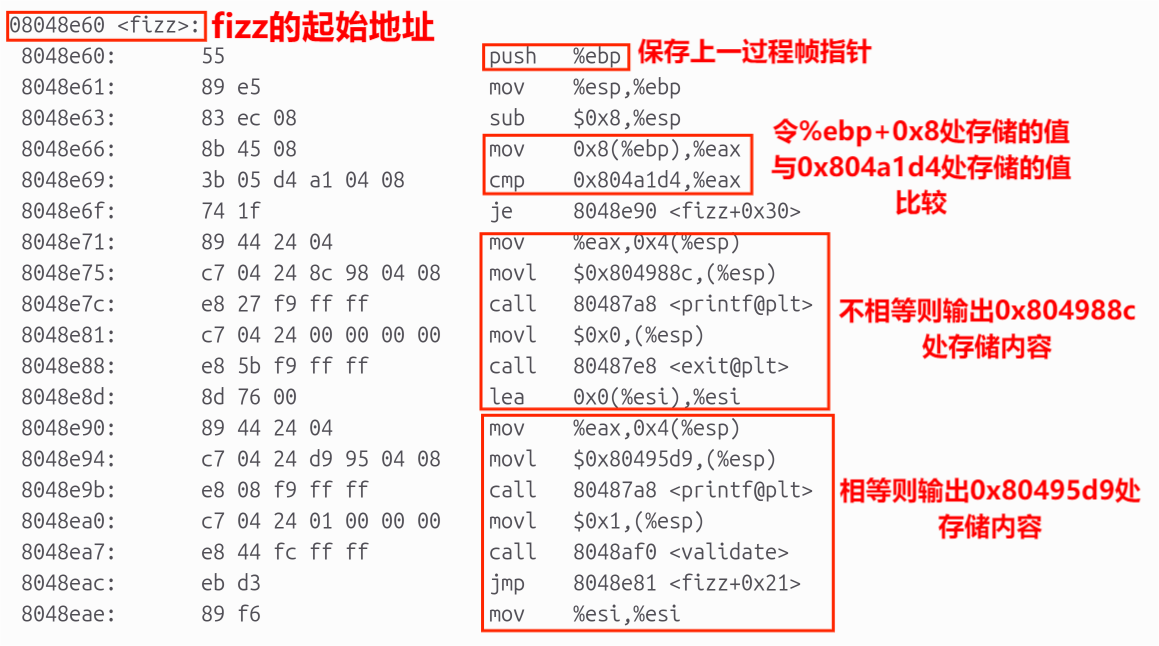


**图 10 关卡1通关截图**

1. **返回到fizz()函数并传参**

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fizz函数，并传入相应参数使得fizz函数执行成功。

**4.1、解题过程**



**图 11 分析fizz的汇编代码**

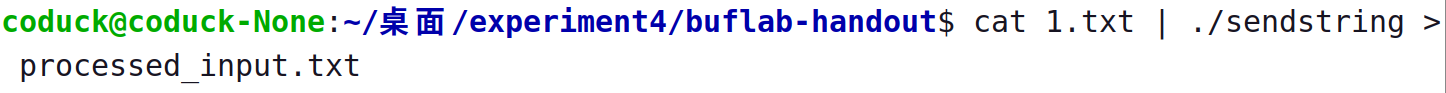
分析fizz的汇编代码，可以知道进入fizz函数后，会令%ebp+0x8处存储的值与0x804a1d4处存储的值相比较，相等则执行一段代码，不相等则执行另一段代码。

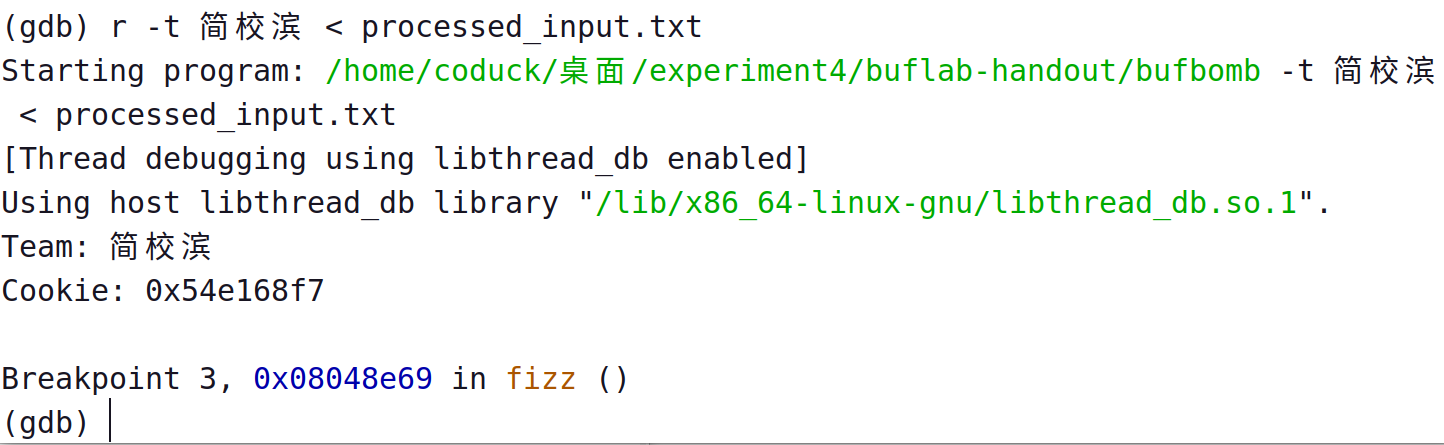
我们通过之前的分析可知，通过输入内容的十六进制为任意28字节值+fizz起始地址，即

00112233445566778899001122334455667788990011223344556677608e0408

就能返回到fizz函数，在0x8048e69处设置断点，用gdb调试bufbomb做进一步分析。

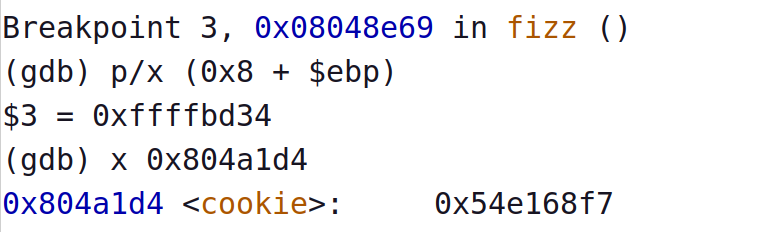
我们可以把输入内容的十六进制值存在1.txt先，在通过sendstring工具转换为字符串存在processed\_input.txt文件，在gdb调试bufbomb时，通过r -t 简校滨 < processed\_input.txt 命令即可完成返回到fizz函数的操作。





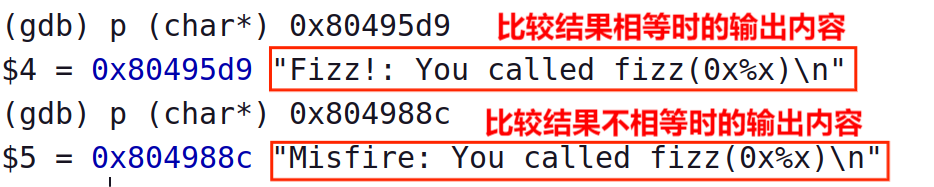
**图 12 调试fizz的汇编代码（1）**

通过调试，我们可以看到%ebp+0x8的地址为0xffffbd34，即存返回地址起始地址0xffffbd2c后8字节处。而0x804a1d4地址处存储的值为我们的cookie值。

****

**图 13 调试fizz的汇编代码（2）**

再分析不同比较结果的输出内容，可以知道如果%ebp+0x8地址存储的值与0x804a1d4地址处存储的值相等，则执行成功；若不相等，则执行失败。



**图 14 调试fizz的汇编代码（3）**

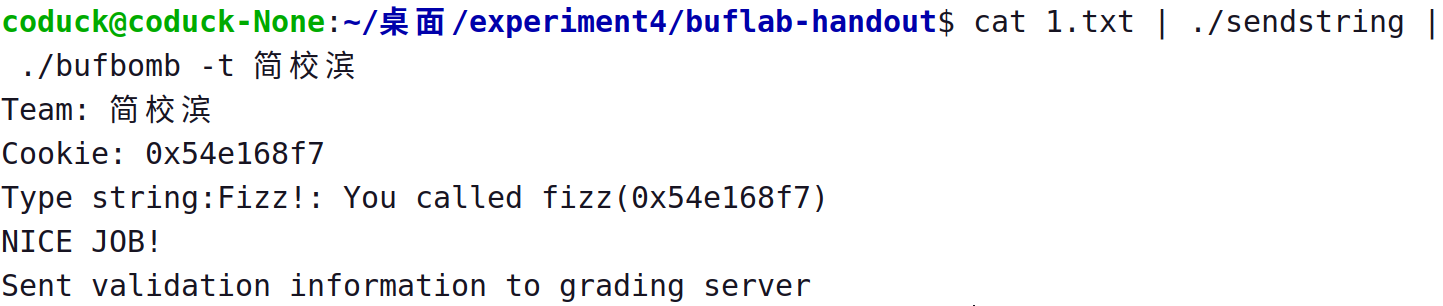
综上，我们可以确定，只有当我们在输入内容后使得0xffffbd34处存有cookie值，才能使得fizz函数执行成功。因此，输入内容十六进制值为任意28字节的内容+fizz的起始地址+任意4字节的内容+cooike值，即

00112233445566778899001122334455667788990011223344556677608e040800112233f768e154

如此，便能保证返回地址被覆盖为fizz函数起始地址的同时，完成向fizz函数的传参，保证fizz函数执行成功。

将输入内容的十六进制值存入文件1.txt，通过sendstring工具将以上内容转换为字符串再传入bufbomb程序，即可成功返回到fizz()函数并传参。

**4.2、最终结果截图**

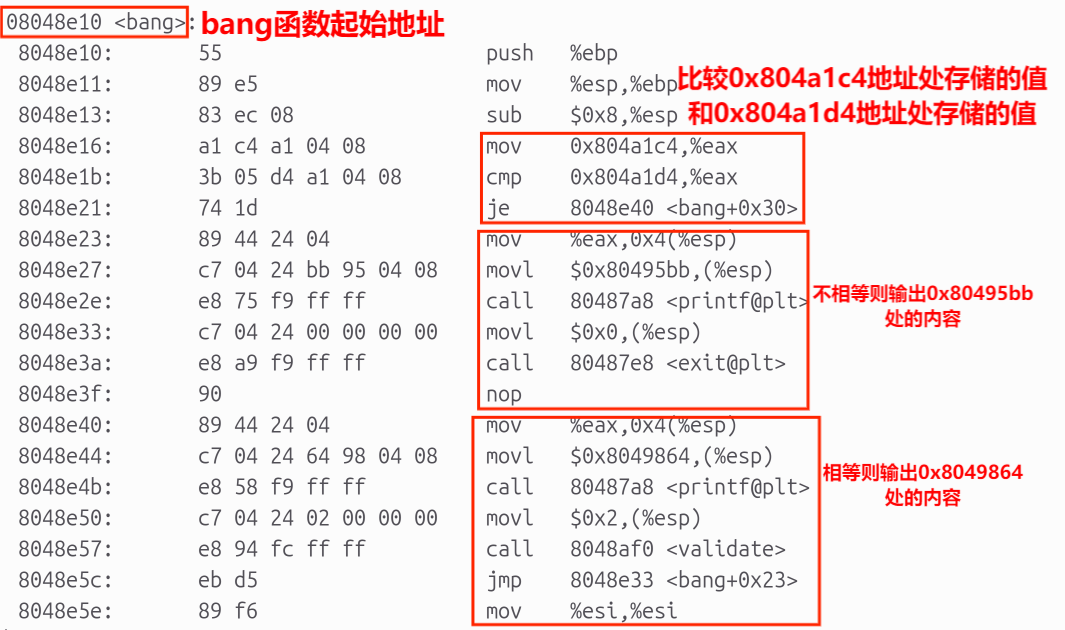
****

**图 15 关卡2通关截图**

1. **返回到bang()且修改global\_value**

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bang函数，并修改global\_value使得bang函数执行成功。

**5.1、解题过程**



**图 16 分析bang的汇编代码**

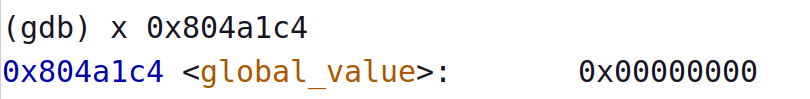
分析bang的汇编代码，可以知道进入bang函数后，会令0x804a1c4处存储的值与0x804a1d4处存储的值相比较，相等则执行一段代码，不相等则执行另一段代码。由上面的分析可知，0x804a1d4处存储的值为cookie值。

我们通过之前的分析可知，通过输入内容的十六进制为任意28字节值+bang起始地址，即

00112233445566778899001122334455667788990011223344556677108e0408

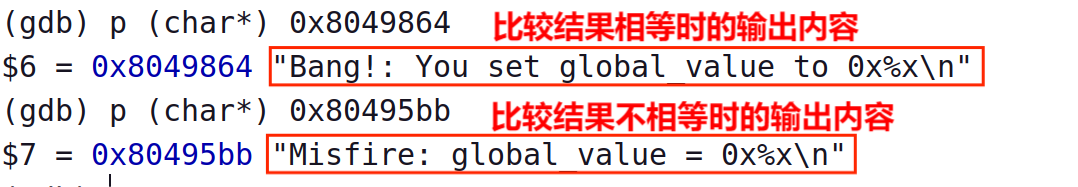
就能返回到bang函数，在0x8048e1b处设置断点，用gdb调试bufbomb做进一步分析。

通过调试，我们可以看到0x804a1c4地址处存的是global\_value的值。



**图 17 调试bang的汇编代码（1）**

再分析不同比较结果的输出内容，可以知道如果0x804a1c4地址存储的值与0x804a1d4地址处存储的值相等，则bang函数执行成功；若不相等，则执行失败。



**图 18 调试bang的汇编代码（2）**

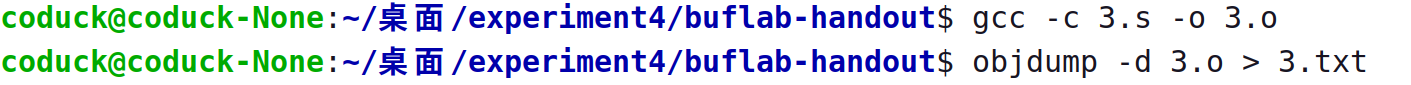
综上，我们可以确定，只有当我们将0x804a1c4处即global\_value的值修改为cookie值，才能使得bang函数执行成功。但由于0x804a1c4距离输入缓存区太远，我们无法像关卡2一样通过输入缓存溢出覆盖原本值的方法来修改global\_value的值。

因此，我们可以先编写修改global\_value的代码段，获得该代码段的十六进制代码后转换为字符串输入存到缓存区中，通过输入的字符串改写返回地址为这段修改global\_value代码段的起始地址，进行global\_value的修改，修改完毕后再重新跳转至bang函数，即可使得bang函数成功执行。

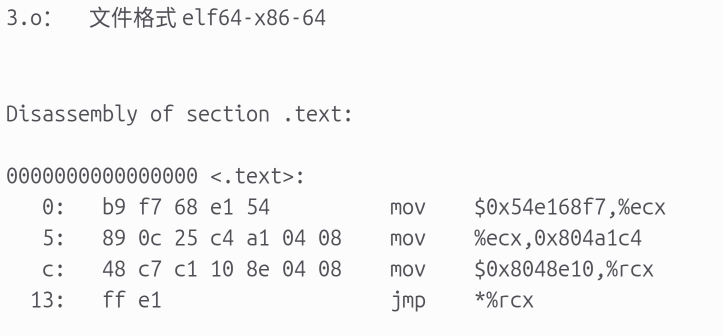
修改global\_value的汇编代码为

|  |
| --- |
| movl $0x54e168f7,%ecx // 将cookie值存入%ecx  movl %ecx,0x804a1c4 // 修改global\_value的值为cookie值  movq $0x08048e10,%rcx  jmp \*%rcx // 跳转到bang函数的起始地址 |

然后通过汇编生成可执行文件，再通过objdump反汇编得到以上代码段的十六进制代码。



**图 19 获取修改global\_value代码段的十六进制代码（1）**



**图 20 获取修改global\_value代码段的十六进制代码（2）**

因此，修改global\_value代码段的十六进制代码为

b9f768e154890c25c4a1040848c7c1108e0408ffe1

由于修改global\_value代码的十六进制值只有21字节，所以需要在补充任意7字节的内容，最后加上该代码段起始地址，即输入缓存区的起始地址0xffffbd10，就能完成返回到bang()且修改global\_value的工作。

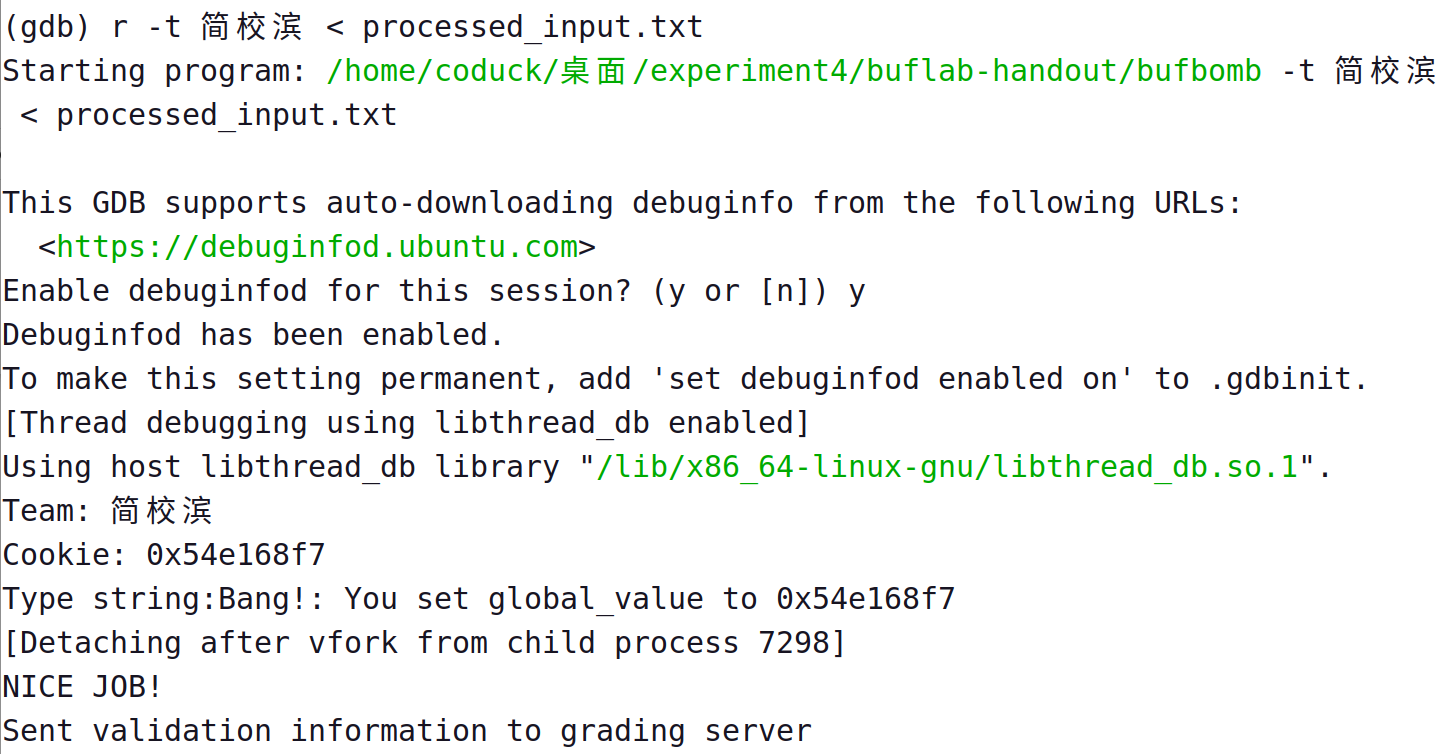
综上，输入内容十六进制值为修改global\_value代码的十六进制值+任意7字节的内容+输入缓存区的起始地址，即

8b0c25d4a10408890c25c4a10408ff2425108e04080000000000000010bdffff

如此，便能保证修改global\_value值为cookie值的同时，返回到bang函数，保证bang函数执行成功。

将输入内容的十六进制值存入文件2.txt，通过sendstring工具将以上内容转换为字符串存储在processed\_input.txt文件中，再传入bufbomb程序，即可成功返回到bang()且修改global\_value。

**5.2、最终结果截图**



**图 21 关卡3通关截图**

**五、实验总结**

在这次实验经历中，我进一步熟悉了GDB调试工具与objdump反编译工具的应用，强化了对缓冲区溢出原理的认知。通过实践，我掌握了如何利用缓冲区溢出漏洞修改返回地址及变量值以实施代码攻击的方法，同时也意识到对输入缓冲区保护的重要性。

|  |
| --- |
| **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： |