**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： 李文俊 学号： 2023150001 班级： 高性能班**

**实 验 时 间： 2024年4月30日~5月14日**

**实验报告提交时间： 2024年5月9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.反汇编可执行程序bomb\_64.dat

**1.1输入反汇编命令查看汇编代码**（保存在1txt文件中）:

***$ objdump -d bufbomb > 1.txt***

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：

图形用户界面, 文本

AI 生成的内容可能不正确。

图 1

2.返回到smoke()

2.1解题思路

本实验中，bufbomb通过先调用<launch>函数，进而调用<test>,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;***

***}***

**目标：使getbuf()返回时，不返回到test()，而是直接返回到指定的smoke()函数。**

图形用户界面, 应用程序

AI 生成的内容可能不正确。

图 2

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。

2.2 解题过程

test()调用getbuf()返回地址：0x8048db2，smoke()的起始地址是0x8048eb0，

分析栈帧结构后可知，buf第一个元素的地址是ebp-0x18，32位系统中地址是4字节，因此”返回地址”第一个字节的地址是ebp+0x04，两个位置的相差换算成换算成十进制就是0x04 - (-0x18) = 4 + 24 = 28，也就是说输入到缓冲区中的数据，要构造28个任意字符，然后加上smoke()的地址（注意小端表示）就能准确覆盖到”返回地址”，完成溢出攻击返回到smoke()。

图示

AI 生成的内容可能不正确。

图 3

无关紧要的28字节区域可以用00~99来填充，两个十六进制位表示一个字节，因此要输入56位。故输入的字符串保存到ans.txt，用cat查看ans.txt内容：$ cat ans.txt

文本

AI 生成的内容可能不正确。

图 4

2.3 最终结果截图

输入 $ cat ans.txt | ./sendstring | ./bufbomb -t lwj

文本

AI 生成的内容可能不正确。

图 5

3.返回到fizz()并准备相应参数

3.1解题思路

fizz()的起始地址是0x08048e60，根据 ./makecookie lwj生成的cookie为0x3756556d

文本

AI 生成的内容可能不正确。

图 6

通过覆盖test的返回地址换成fizz()的返回地址和之前一样，这一次更进一步需要将传入的参数替换为cookie，需要得到fizz()参数存储的位置，通过Gets继续覆盖传入参数。

3.2 解题过程

分析栈帧结构后可知，test()的返回地址在%ebp，而fizz()调用%ebp+0x8，结果会根据与0x804a1d4是否相等来决定输出哪一种结果，输出0x804a1d4发现是cookie值，因此需要传入的参数需要放在%ebp+0x8

文本

AI 生成的内容可能不正确。

图 7

图示, 工程绘图

AI 生成的内容可能不正确。

图 8

所以如果要把cookie传入fizz()，需要先构造8个任意字符，然后加上cookie（注意小端表示）就能准确覆盖到传入参数，完成溢出攻击返回到含参数的fizz()。

3.3 最终结果截图

用cat查看ans.txt内容：$ cat ans.txt



图 9

输入 $ cat ans.txt | ./sendstring | ./bufbomb -t lwj

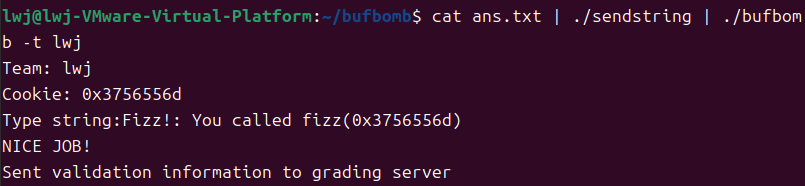


图 10

4.返回到bang()且修改global\_value

4.1解题思路

这道题和fizz()函数的判断流程相同，通过比较cookie和global\_value相等来validate，因此怎么修改global\_value的值是关键，根据提示，需要在输入数据中包含黑客构造的代码，因此需要通过编写C或汇编代码并通过GCC产生目标文件，从而得到所需的代码。

文本

AI 生成的内容可能不正确。

图 11

4.2 解题过程

**注意：关闭 Linux 的内存地址随机化，才能够准确跳转到给定的地址：**

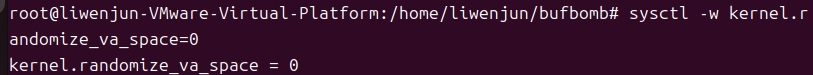


图 12

**问题是修改代码要放到什么位置呢？系统在什么情况下才会执行这段代码？**如果按照之前的思路，系统只会在ret时候读覆盖后的返回地址去到bang，不会理会缓冲区前面的代码，所以如果汇编代码只是单纯地将 global\_value=cookie + 覆盖为<bang>地址，系统不会执行修改。

**因此，需要将返回地址调整为缓冲区的开头，然后执行修改代码最后再跳转到bang()。**

编写将cookie赋值给global\_value的汇编代码（两个常量都已知存储地址），gcc编译后再objdump得到对应的机器代码：

文本

AI 生成的内容可能不正确。

图 13

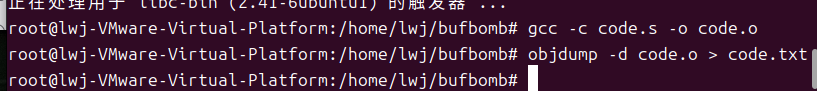


图 14 文本

AI 生成的内容可能不正确。

图 15

因此前面的部分是 机器代码（25个字节），中间需要3字节任意数字，然后覆盖4字节的缓冲区首地址（%ebp-0x18）。

通过b getbuf 设置断点 ， r -t lwj 开始运行，p $ebp 查看此时%ebp-0x18是0xffffafd0（注意小端序）

文本

AI 生成的内容可能不正确。

图 16

文本

AI 生成的内容可能不正确。

图 17

4.3 最终结果截图

ans.txt：488b1425d4a1040848891425c4a1040848c7c2108e0408ffe2**080808**d0afffff

用cat查看ans.txt内容：$ cat ans.txt



图 18

输入 $ cat ans.txt | ./sendstring | ./bufbomb -t lwj

文本

AI 生成的内容可能不正确。

图 19

**五、实验总结与体会**

**本次实验围绕bufbomb程序的栈溢出漏洞展开，通过gdb调试工具深入分析了getbuf()函数的栈帧布局，精确计算出缓冲区大小为24字节，并定位返回地址位于缓冲区后第28-31字节处。针对三个目标，分别构造了不同的攻击载荷：**

​劫持返回地址至smoke()：在输入中填充24字节后直接覆盖返回地址为smoke()的函数地址（小端序），利用info frame验证栈帧修改成功，实现控制流跳转；

​向fizz()注入参数：结合调用约定规则，在覆盖fizz()地址后添加4字节占位符及自定义的cookie值（通过makecookie生成），利用x/xw $esp+8验证参数传递正确性；

​篡改全局变量并执行bang()：通过gdb定位global\_value的内存地址及bang()入口，构造ROP链（pop %eax; ret与mov [global\_value], %eax; ret）将cookie写入全局变量，最终跳转至bang()。

实验过程中，深入理解了call指令的压栈逻辑、ret指令对程序流的控制机制，以及栈溢出漏洞利用的核心原理（如覆盖返回地址、参数注入、ROP链构造）。同时，熟练掌握了gdb的disas反汇编、watch监控变量、x查看内存等调试技巧，提升了二进制程序分析与漏洞攻防的实践能力，为后续深入理解内存安全机制奠定了基础。

|  |
| --- |
| **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： |