**武汉大学计算机学院**

**本科生实验报告**

**CSAPP: Data Lab, Bomb Lab, Attack Lab解题过程与心得**

专 业 名 称 ：软件工程

课 程 名 称 ：系统级程序设计

指 导 教 师 ：李清安

学 生 学 号 ：2017302580098

学 生 姓 名 ：姚晓璐

二○一九年十二月

**郑 重 声 明**

本人呈交的实验报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

本人签名： 姚晓璐 日期： 2019.12.8

摘 要

通过数据实验、二进制炸弹、缓冲区溢出攻击三个实验，熟悉和掌握整型和浮点型数据的二进制表示及基本运算，理解汇编语言并学习如何使用调试器，掌握栈的使用约定并深刻体会如果代码中存在缓冲区漏洞会导致怎样的风险。

**关键词：** 数据实验 二进制炸弹 缓冲区溢出攻击

目 录

[1 Data Lab 1](#_Toc26824244)

[1.1 实验概述 1](#_Toc26824245)

[1.2 实验过程 1](#_Toc26824246)

[1.2.1 bitAnd 1](#_Toc26824247)

[1.2.2 getByte 1](#_Toc26824248)

[1.2.3 logicalShift 2](#_Toc26824249)

[1.2.4 bitCount 2](#_Toc26824250)

[1.2.5 bang 3](#_Toc26824251)

[1.2.6 tmin 3](#_Toc26824252)

[1.2.7 fitsBits 4](#_Toc26824253)

[1.2.8 divpwr2 4](#_Toc26824254)

[1.2.9 negate 5](#_Toc26824255)

[1.2.10 isPositive 5](#_Toc26824256)

[1.2.11 isLessOrEqual 5](#_Toc26824257)

[1.2.12 ilog2 6](#_Toc26824258)

[1.2.13 float\_neg 6](#_Toc26824259)

[1.2.14 float\_i2f 6](#_Toc26824260)

[1.2.15 float\_twice 7](#_Toc26824261)

[1.3 实验测试 8](#_Toc26824262)

[1.4 实验心得 8](#_Toc26824263)

[2 Bomb Lab 9](#_Toc26824264)

[2.1 实验概述 9](#_Toc26824265)

[2.2 实验过程 9](#_Toc26824266)

[2.2.1 字符串比较 9](#_Toc26824267)

[2.2.2 循环 10](#_Toc26824268)

[2.2.3 条件/开关 10](#_Toc26824269)

[2.2.4 递归调用和栈约定 11](#_Toc26824270)

[2.2.5 指针 12](#_Toc26824271)

[2.2.6 链接表/指针/结构类型 13](#_Toc26824272)

[2.2.7 隐藏关卡 14](#_Toc26824273)

[2.3 实验测试 15](#_Toc26824274)

[3 Attack Lab 16](#_Toc26824275)

[3.1 实验概述 16](#_Toc26824276)

[3.2 实验过程 16](#_Toc26824277)

[3.2.1 attack1 2](#_Toc26824278)

[3.2.2 attack2 2](#_Toc26824279)

[3.2.3 attack3 3](#_Toc26824280)

[3.2.4 attack4 3](#_Toc26824281)

[3.2.5 attack5 5](#_Toc26824282)

# 1 Data Lab

## 1.1 实验概述

在这个实验中，学生需要修改一个名为bits.c的C文件。该文件中包含多个空的方法体。学生需要在这些方法体中填写代码，使得每个方法都能够分别实现一个特定的数学函数，比如“绝对值”。在实现过程中，还有一些别的约束。比如，整型函数的实现中，学生只能使用顺序执行的C代码和限定的算术及逻辑操作符。

Linux下一些可能需要的命令：

sudo apt-get install build-essential：安装gcc。

./dlc bits.c：用于检查学生的代码实现是否满足指定的约束。

make。

./btest：用于检查学生的代码实现是否正确。

## 1.2 实验过程

此实验共有十五个函数需要补充。

### 1.2.1 bitAnd

要求：只用 ~ 和 | 来求x & y。例如：bitAnd(6, 5) = 4。

思路：使用德摩根定律： 来解决。

代码：如图1.1所示。

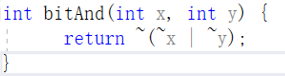


图1.1 bitAnd代码

### 1.2.2 getByte

要求：从字x里取出第n个字节（0 <= n <= 3）。例如：getByte(0x12345678,1) = 0x56。

思路：将n左移3位，即扩大8倍，得到一个字节大小的m。将x右移m位，即将要取出的字节放在低八位，然后和0xFF做与运算，求得结果。

代码：如图1.2所示。

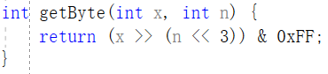


图1.2 getByte代码

### 1.2.3 logicalShift

要求：将x逻辑右移n位（0 <= n <= 31）。例如，logicalShift(0x87654321,4) = 0x08765432。

思路：需要知道逻辑右移最高位补0，而算术右移最高位补符号位。可以将1左移31位后与1做与运算，得到符号位和32位数m。然后将m算数右移n-1位后取反，将此结果和x算术右移n-1位后的结果做与运算，即可将高位变为0.

如某数：sxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx

右移n位后得到：ssss … ss(n-1个s) sxxx xxxx … xxx（32-n个x）

0x1左移31位后与x做与运算：s000 0000 … 0000

右移n-1位：ssss … ss（n-1个s）0000 0000 … 0000

取反：s’s’s’s’… s’s’ 1111 1111 … 1111

x右移n位后：ssss ssss … sxxx xxxx … xxxx

做与运算：0000 0000 … sxxx xxxx … xxxx

代码：如图1.3所示。

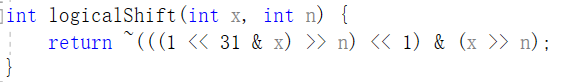


图1.3 logicalShift代码

### 1.2.4 bitCount

要求：返回字中1的个数。例如，bitCount(5) = 2。

思路：字有32位，可以将其分成四个字节，每八位进行测试。用0x1111与x做与运算得到val，分别测每个字节的最低位，然后将x右移一位，重复此步骤。这样val上每个字节的值即对应x每个字节1的数量，对val做移位操作，就可以得到1的个数。

代码：如图1.4所示。

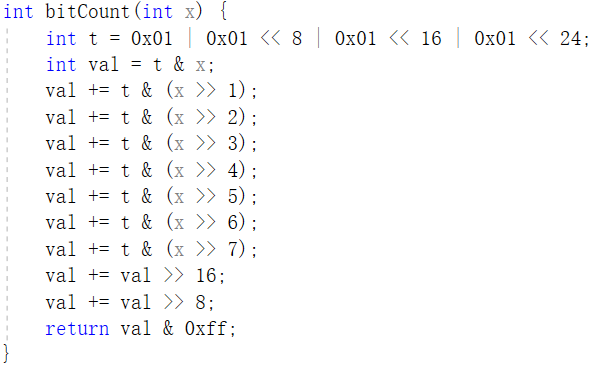


图1.4 bitCount代码

### 1.2.5 bang

要求：不使用符号 ! 求 !x。例如，bang(3) = 0。

思路：求非，需要考虑 0 的情况。将 x 取反加一，则只有 x 为0时两个数的符号位才都为 0，否则总会有一个的符号位为1。再将其右移31位，与 0x01做与运算，得到结果。

代码：如图1.5所示。

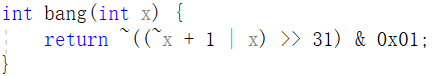


图1.5 bang代码

### 1.2.6 tmin

要求：求补码编码的最小值。

思路：补码编码的最小值可理解为正数溢出的第一个值，即0x80000000。

代码：如图1.6所示。

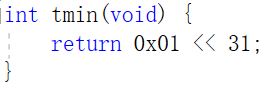


图1.6 tmin代码

### 1.2.7 fitsBits

要求：如果x可以被n（1 <= n <= 32）位补码表示，则返回1，否则，返回0。例如，fitsBits(5,3) = 0。

思路：分 x 为负数或非负数两种情况考虑。x 为负数时，符号位为1；反之为0。x = 0 时，x >> n-1 结果为 0；反之 ~x >> n-1 结果为0。

代码：如图1.7所示。

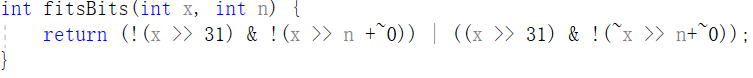


图1.7 fitsBits代码

### 1.2.8 divpwr2

要求：计算x/(2^n)。例如，divpwr2(15,1) = 7。

思路：分 x 为负数或非负数两种情况考虑。x 为非负数时，直接右移 n 位；

x 为负数时，如果 x 移出的位中不全为 0，则结果为右移 n 位后的结果加一。x为负数时的证明如图1.8所示。

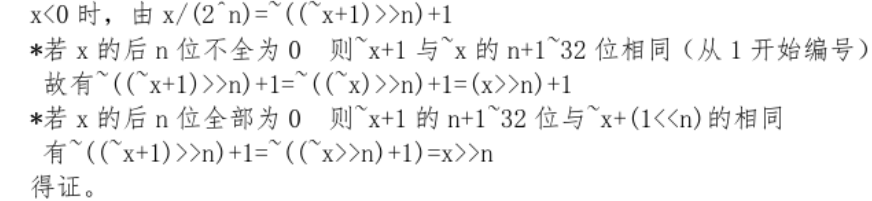


图1.8 x为负数时的证明

代码：如图1.9所示。

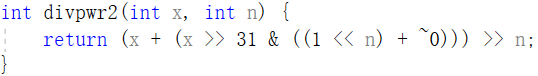


图1.9 divpwr2代码

### 1.2.9 negate

要求：返回 -x。例如，negate(1) = -1。

思路：取反加一。

代码：如图1.10所示。

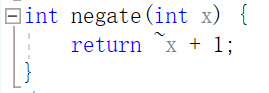


图1.10 negate代码

### 1.2.10 isPositive

要求：如果x大于0，返回1；否则，返回0。例如，isPositive(-1) = 0。

思路：取符号位即可判断正负。在此题中需要考虑 0 的情况。x的符号位和 !x进行异或操作，非0的数一定为1。

代码：如图1.11所示。

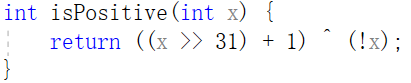


图1.11 isPositive代码

### 1.2.11 isLessOrEqual

要求：如果x小于等于y，则返回1；否则，返回0。例如，isLessOrEqual(4,5) = 1。

思路：比较 x 和 y 的符号位。得到 y – x 的符号位。注意 x、y 异号时的可能溢出情况。具体的溢出情况可参照课本第二章的内容。

代码：如图1.12所示。

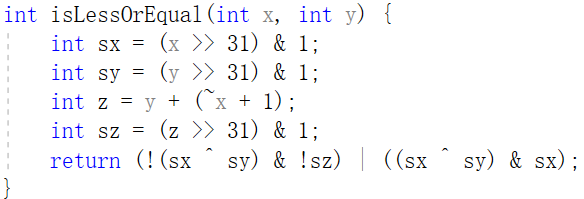


图1.12 isLessOrEqual代码

### 1.2.12 ilog2

要求：返回中的x。例如，ilog2(16) = 4。

思路：二分法。先检查第16位的情况；再检查高（低）8位的情况；……依此类推。例如，数0000 0000 0000 0001 0001 0001 0001 0001，先检查第16位的情况，为1；此时，再检查第24位的情况，为0；再检查第22位的情况，同样为0；检查第21位的情况，同样为0。所以结果为16。

代码：如图1.13所示。

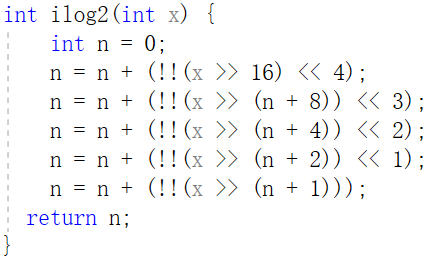


图1.13 ilog2代码

### 1.2.13 float\_neg

要求：返回浮点数的负数。

思路：在单精度浮点数中，第31位为符号位，30~23位为阶码，22~0位为尾数。阶码不全为 1 或尾数不全为0时，取符号位得到结果；否则返回NaN。这是根据IEEE浮点数的规则而来的，阶码全为1，尾数不全为0就是NaN；阶码全为1，尾数全是0就是正负无穷；阶码尾数全为0就是正负0。

代码：如图1.14所示。

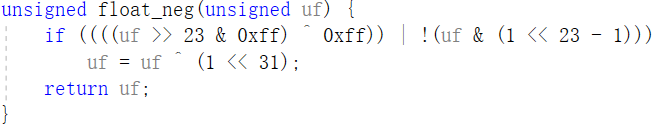


图1.14 float\_neg代码

### 1.2.14 float\_i2f

要求：将整形x返回为浮点类型。

思路：取符号位。初始化阶码：(x>>31) ? (127 + 31) : 0。尾数部分将前缀 0 去除，取高23位，加上后面剩余位四舍五入的精度位。具体内容可以参照书本第二章。

代码：如图1.15所示。

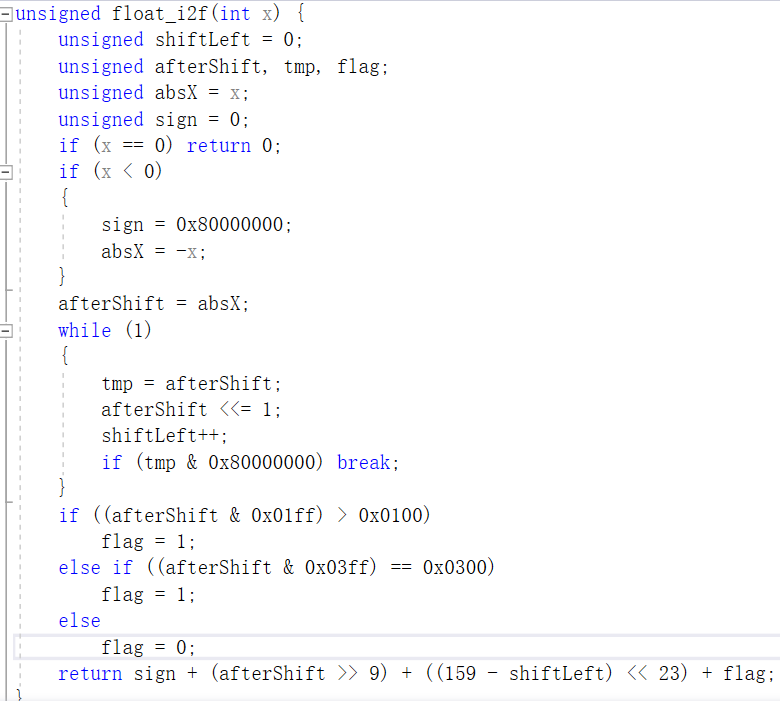


图1.15 float\_i2f代码

### 1.2.15 float\_twice

要求：计算浮点数的两倍。

思路：计算 uf \* 2，分两种情况考虑：阶码部分为 0，尾数部分左移一位；阶码部分不为0，阶码加一。

代码：如图1.16所示。

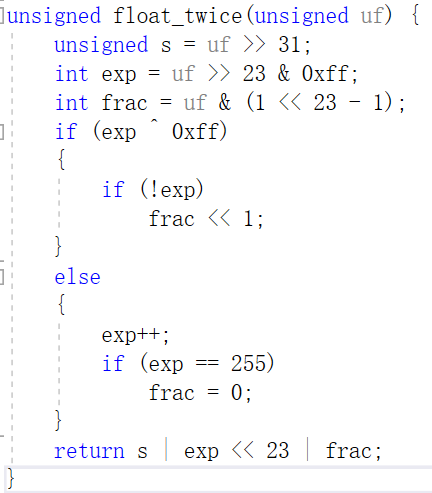


图1.16 float\_twice代码

## 1.3 实验测试

在linux下进行测试。如图1.17所示。

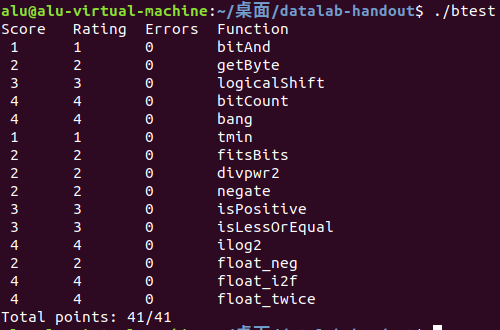


图1.17 实验一测试结果

## 1.4 实验心得

即使很多都只有一行代码，但是写起来依然十分困难。常常处于一种毫无头绪的状态。甚至第一个题目，求x & y，看起来极其简单的题目，都想了很久，没有想到可以用德摩根定理去求。只是在草稿纸上涂涂画画，试着通过各种运算拼凑出想要的答案。很多题目似乎都是这么来的，就在纸上自己举例子，然后通过各种运算，比如与、或、非，等等，卡能否求出答案。但即便如此，有些题也不能答对。比如0的情况，溢出的情况，等等，都没有考虑到。有的题目，看了答案之后，还是一头雾水，不知是什么意思。后又将书本第二章的内容仔细阅读了一遍，才恍然大悟，原来是在这个意思。也是很有意思了。这个实验写了很久吧，前前后后写了两天左右，加上还需要做PPT之类，花了挺多功夫的。不过还是挺满足的吧，至少通过这些题目把书上的内容熟悉的差不多了。然后发现CSAPP真的很不错，是我读过的最好的专业书了。写得很仔细，很细心，把我之前模模糊糊，模棱两可的东西全解决了，有一种茅塞顿开的感觉。想来即使期末只能读一部分，在考完后这本书也要尽力去读，去深挖，对个人提高和代码能力应该会有很大的提升。

# 2 Bomb Lab

## 2.1 实验概述

一个“二进制炸弹”是指一个Linux下的可执行程序，包含六个关口。每个关口需要学生从控制台（stdin）输入一个猜测的特定的字符串（可以理解为拆弹密钥）。如果猜对了，则拆弹成功；如果猜错了，则炸弹爆炸并输出“BOOM!!!”。学生的目标是尽可能多次拆弹成功。

## 2.2 实验过程

### 2.2.1 字符串比较

反汇编。得到如图2.1所示代码。

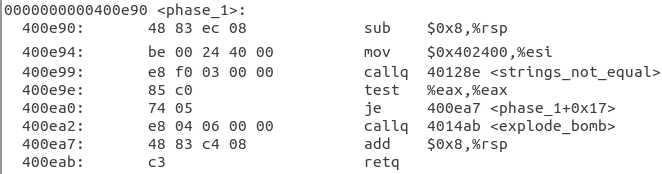


图2.1 反汇编phase1后的代码

可以发现，在第2行中，将一个数移进了寄存器 %esi 中，而第三行中调用了字符串比较的函数，如果不相等，则爆炸。因此可以知道，第二行的数即为答案。

### 2.2.2 循环

反汇编。得到图2.2所示代码。

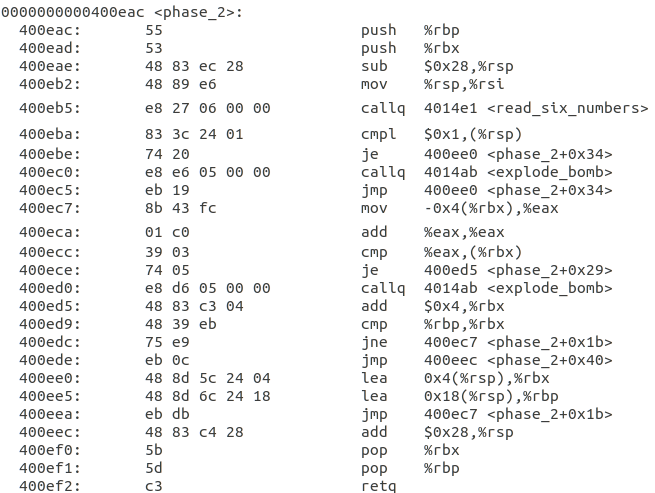


图2.2 反汇编phase2后的代码

可以知道，在第5行中，调用了一个函数，需要读取六个数字。第6行中，将数字1与输入的第一个数字进行比较，如果不相等，则爆炸。之后，在0x400eca中，可以得知需要将数字乘以2，然后再进行比较。因此可以发现，此题是以1开始，以2为比值的等比数列。

### 2.2.3 条件/开关

反汇编。得到图2.3所示代码。

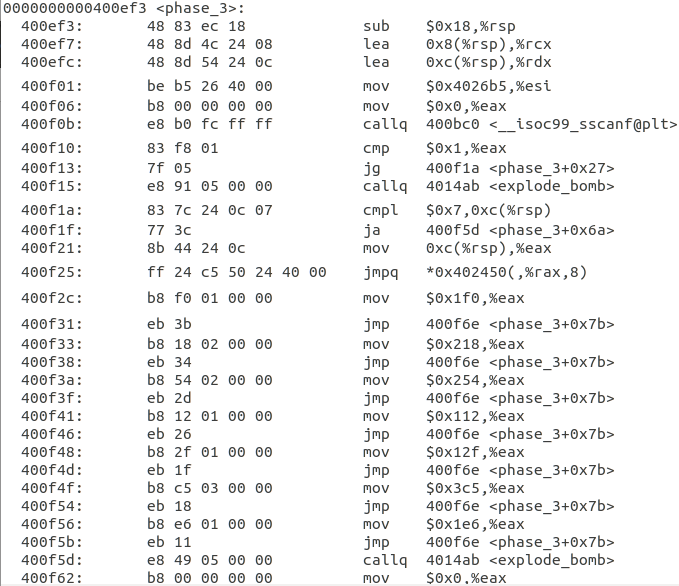


图2.3 反汇编phase3后的部分代码

在第4行中，使用gdb查看发现此处需要输入两个整数。在第6行中，扫描输入的数据。并在第7行中将1与输入的数据数量作比较，如果不大于1，则爆炸。在0x400f25处，跳转表，可以发现是一个switch…case的函数。在0x400f2c处，可以发现数字0的结果是，0x1f0，将其转换为十进制数为496即为我们需要的值。

### 2.2.4 递归调用和栈约定

反汇编。得到图2.4所示代码。

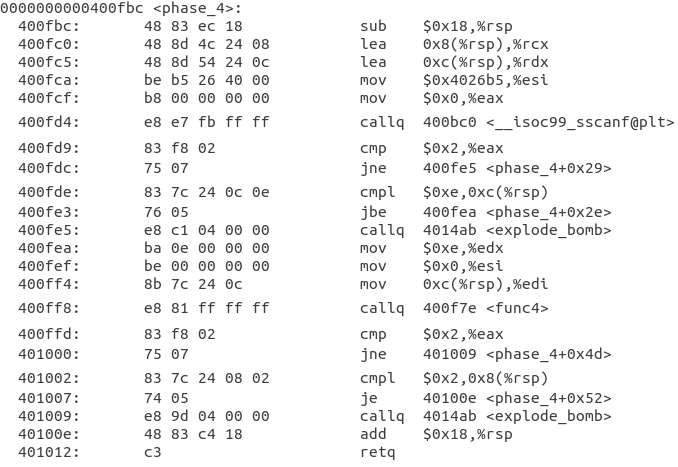


图2.4 反汇编phase4后的部分代码

在地址0x400fd9处，可以发现，需要输入两个整数。在0x400fde处，发现输入的第一个数不大于14。在0x400ffd处，发现func4返回值为2。在0x401002处可以发现，输入的第二个数必须为2。而第一个数字，需要一个个试，得到正确结果。

### 2.2.5 指针

反汇编。得到图2.5所示代码。

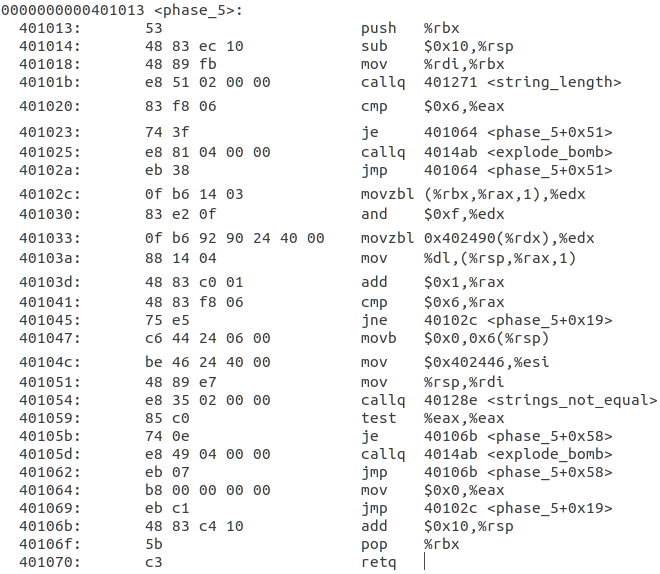
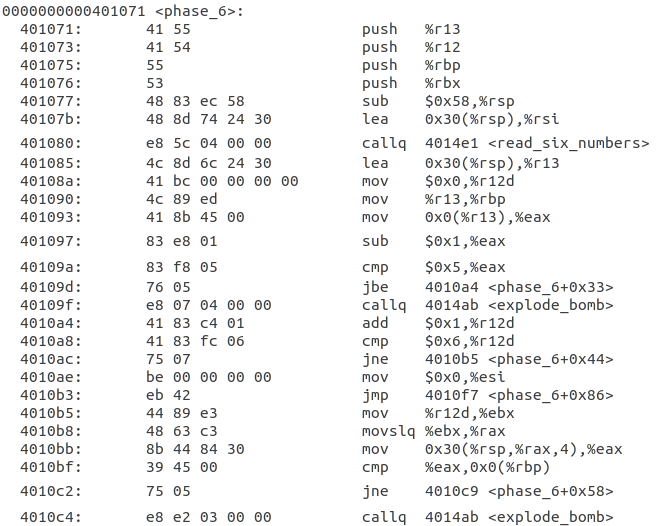


图2.5 反汇编phase5后的代码

在地址0x401020处，可以发现，需要输入的字符串长度为6。用gdb反编译地址0x401033处，可以得到“maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?”。再反编译地址0x40104c处，得到“oilers”。其每个字符在0x401020处的字符数组“maduiersnfotvbyl”中的对应的偏移量为10，4，15，5，6，7。查阅ASCII表格可以得到结果为jdoefg。

### 2.2.6 链接表/指针/结构类型

反汇编。得到图2.6所示代码。



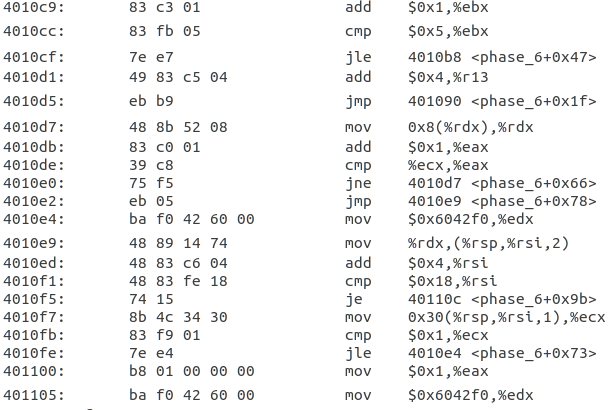


图2.6 反汇编phase6后的部分代码

在地址0x401080可知，此题需要输入六个数字。且由0x401097可知，输入的数字需要在1~6之间，且第一个数字要小于等于5。由0x4010c2可知，输入的数字每个都不能重复，否则爆炸。在地址0x401105处，反编译，(gdb) x /24xw 0x6042f0，得到六个节点。如图2.7所示。

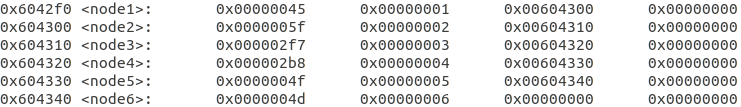


图2.7 0x6042f0的结果

按照降序排列，可以发现，node3 > node4 > node2 > node5 > node6 > node1。所以输入的值为3 4 2 5 6 1。

### 2.2.7 隐藏关卡

反汇编。得到图2.8，图2.9所示代码。

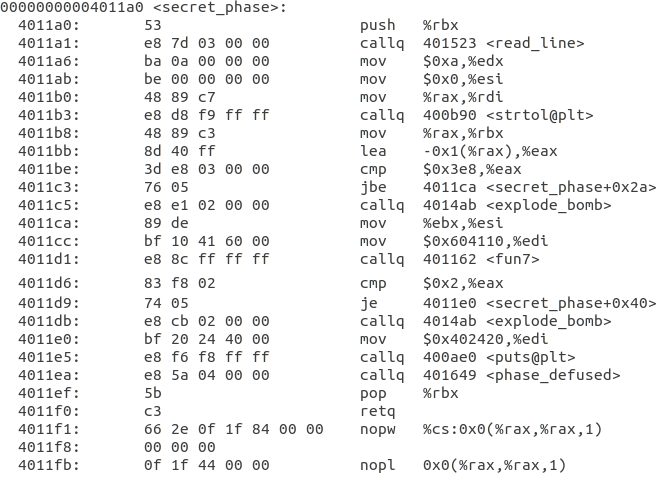


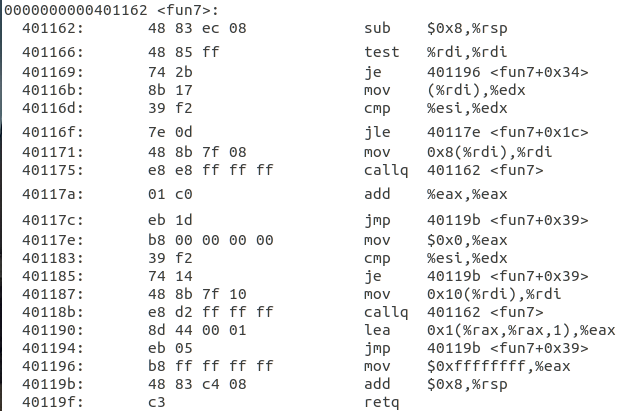
图2.8 反汇编secrete phase后的代码

图2.8 反汇编secrete phase后的func7代码

在地址0x4011d6处，比较func7返回的是否是两个值。而通过对func7的分析可知，此汇编代码对应的C语言代码大致如图2.9所示。

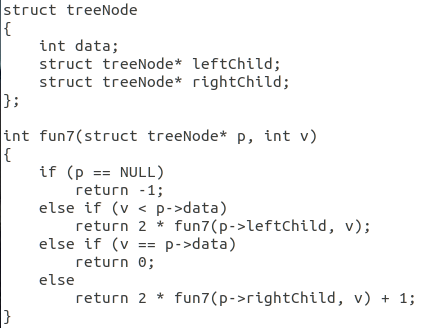


图2.9 func7对应的C语言代码

因此，func2返回2 2\*(1+2\*0)。所以输入的数字须得等于n32节点中的值0x16(22)。

## 2.3 实验测试

实验结果由2.10所示。

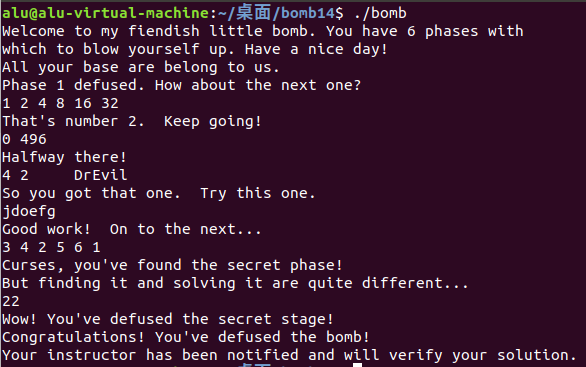


图2.10 实验结果

# 3 Attack Lab

## 3.1 实验概述

实验中为每个学生提供了两个定制的二进制文件。其中一个很容易被代码注入攻击，另一个很容易被基于return的编程方法攻击。学生的任务就是利用这两种攻击思路，设计程序输入，以实现缓冲区漏洞攻击。

## 3.2 实验过程

### 3.2.1 attack1

反汇编得到3.1所示touch1代码。

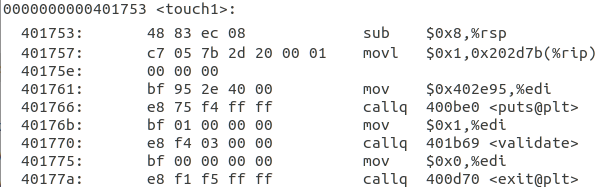


图3.1 touch1汇编代码

gebuf得到缓冲区大小为0x38，即56字节。

将缓冲区溢满后跳转到touch1首地址，即为答案，如图3.2所示。

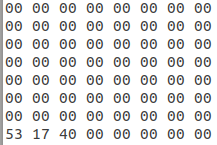


图3.2 attck1答案

### 3.2.2 attack2

反汇编得3.3所示代码。

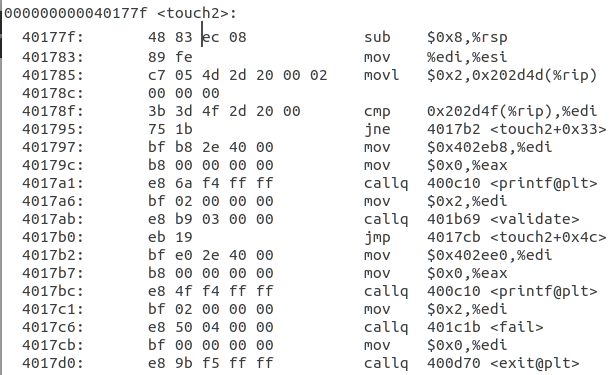


图3.3 touch2汇编代码

需要注入的代码如图3.4所示。



图3.4 需要注入的代码

将上述的汇编代码转化为计算机可以执行的指令序列，得到如图3.5所示结果。

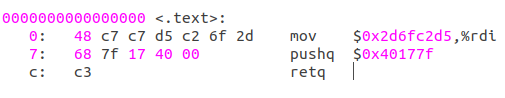


图3.5 将汇编转成机器可执行的指令序列

通过gdb得到缓冲区地址。

即可得出attack2答案。如图3.6所示。

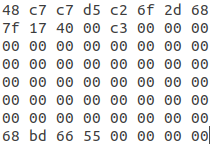


图3.6 attack2答案

### 3.2.3 attack3

与attack2相似，也是需要在输入的字符串中注入一段代码，但是不同于第二阶段的是，在这一阶段中需要传递字符串作为参数。

得到attack3的答案，如图3.7所示。

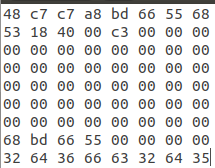


图3.7 attack3答案

### 3.2.4 attack4

这一关需要知道一些指令的编码。如图3.8所示。

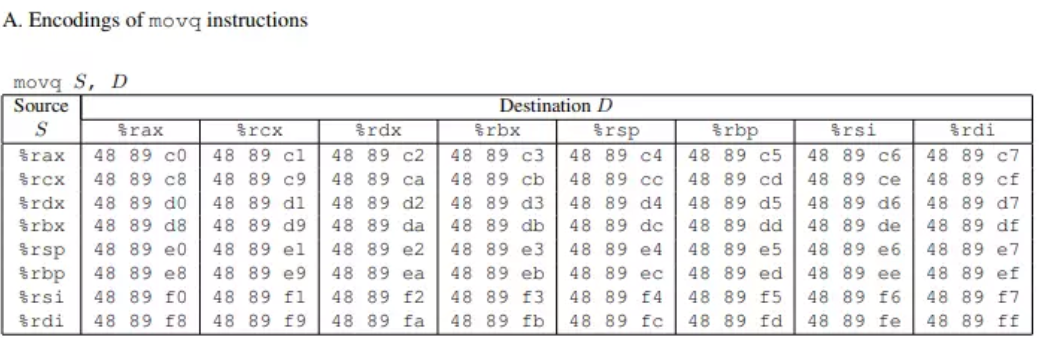


图3.8 movq的指令编码

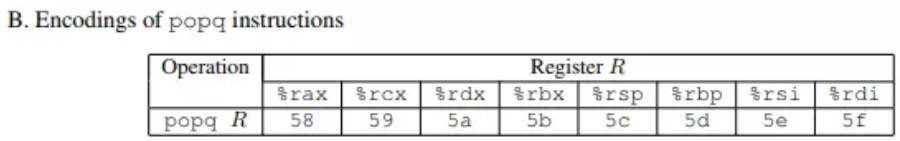


图3.8 popq指令编码

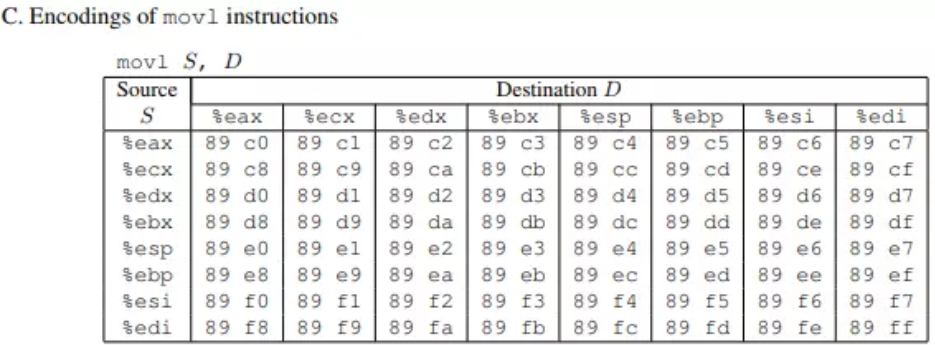


图3.9 movl指令编码

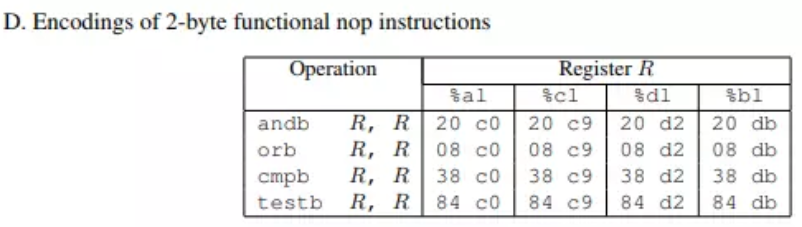


图3.10 其他的一些指令编码

这一阶段中，其实是重复代码注入攻击中第二阶段，劫持程序流，返回到touch2函数。只不过这个我们要做的是ROP攻击，无法再像上一阶段中将指令序列放入到栈中，所以需要到现有的程序中，找到我们需要的指令序列。我们需要的代码序列：

popq %rax

movq %rax, %rdi

从farm.d文件中找出可以用到的函数，即可得到结果。popq的可用函数如图3.11所示。



图3.11 popq可用函数

movq %rax, %rdi可用函数如图3.12所示。



图3.12 movq %rax, %rdi可用函数

结果如图3.11所示。

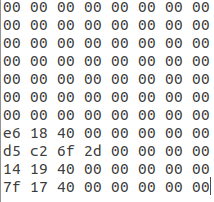


图3.11 attack4

### 3.2.5 attack5

把字符串的起始地址，传送到%rdi,然后调用touch3函数。因为每次栈的位置是随机的，所以无法直接用地址来索引字符串的起始地址，只能用栈顶地址 + 偏移量来索引字符串的起始地址。从farm中可以获取到这样一个gadget，lea (%rdi,%rsi,1),%rax，这样就可以把字符串的首地址传送到%rax。过程与phase4相似。结果如图3.12所示。



图3.12 attack5答案

教师评语评分

评语：

评分：

评阅人：

年 月 日

（备注：对该实验报告给予优点和不足的评价，并给出百分之评分。）