

**课 程 实 验 报 告**

**课程名称： 计算机系统基础实验**

**专业班级： CS1705**

**学 号： U201714726**

**姓 名： 王明明**

**指导教师： 胡侃**

**报告日期： 2019.6.2**

**计算机科学与技术学院**

目 录

[2.1实验概述 3](#_Toc10403893)

[2.2 实验过程 4](#_Toc10403894)

[2.2.1 阶段一 字符串比较 4](#_Toc10403895)

[2.2.2 阶段2 循环 5](#_Toc10403896)

[2.2.3 阶段三 条件/分支 7](#_Toc10403897)

[2.2.4 阶段四 递归调用和栈 9](#_Toc10403898)

[2.2.5 阶段五 指针 12](#_Toc10403899)

[2.2.6 阶段六 链表 13](#_Toc10403900)

[2.2.7 阶段七 隐藏阶段 14](#_Toc10403901)

# 实验二 Binary Bombs

# 2.1 实验概述

本实验中，你要使用课程所学知识拆除一个“Binary Bombs”来增强对程序的机器级表示、汇编语言、调试器 和逆向工程等方面原理与技能的掌握。 一个“Binary Bombs”（二进制炸弹，简称炸弹）是一个 Linux可执行C程序，包含phase1~phase6共6个阶段。 炸弹运行的每个阶段要求你输入一个特定的字符串，若你 的输入符合程序预期的输入，该阶段的炸弹就被“拆除”， 否则炸弹“爆炸”并打印输出 "BOOM!!!"字样。 实验的目标是你要拆除尽可能多的炸弹阶段。

每个炸弹阶段考察了机器级语言程序的一个不同方面， 难度逐级递增：

阶段1：字符串比较

阶段2：循环

阶段3：条件/分支：含switch语句

阶段4：递归调用和栈

阶段5：指针

阶段6：链表/指针/结构

另外还有一个隐藏阶段，但只有当你在第4阶段的解之后附加一特定字符串后才会出现。

拆弹技术：为了完成二进制炸弹拆除任务，你需要

① 使用gdb调试器和objdump来反汇编炸弹的可执行文件；

② 单步跟踪调试每一阶段的机器代码

③ 理解每一汇编语言代码的行为或作用，

④ 进而设法“推断”出拆除炸弹所需的目标字符串。

⑤ 这可能需要你在每一阶段的开始代码前和引爆炸弹的函数前设置断点，以便于调试。

实验语言：C语言

实验环境：linux

# 2.2 实验过程

反汇编bomb，得到反汇编代码，根据汇编代码完成拆炸弹任务。

## 2.2.1 阶段一 字符串比较

1. 任务描述：找到与输入的字符串进行比较的存储的字符串的首地址，进而得到存储的字符串，得到结果。

2. 实验设计： 根据反汇编代码一步一步分析，具体步骤见实验过程。

3. 实验过程： 将bomb反汇编输出到asm.txt中，在反汇编代码中查找main、phase\_1的位置：

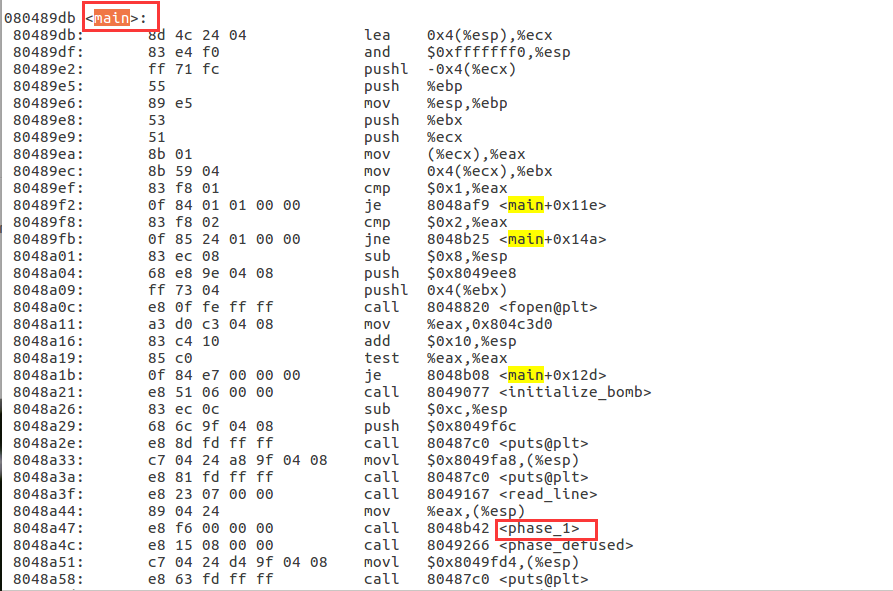


图2-1 main位置截图

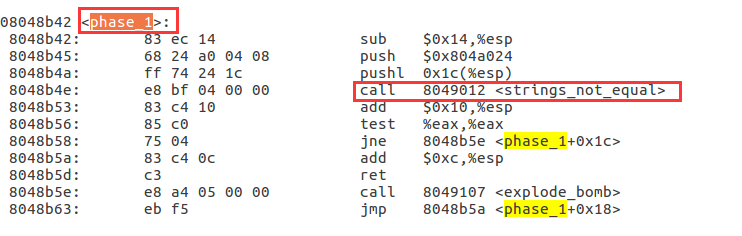


图2-2 phase\_1位置截图

发现在调用strings\_not\_equal对比字符串之前,有一个地址送入了堆栈,大胆猜测该地址就是正确字符串的首地址，即0x804a024，可以查看该地址存储的数据，如图2-3所示：

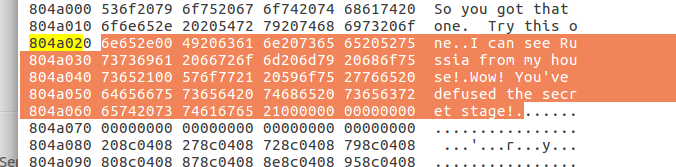


图2-3 猜测的字符串数据

故猜测“I can see Russia from my house!”就是所需字符串。重新执行该程序，直接输入该字符串，观察结果，如图2-4所示：

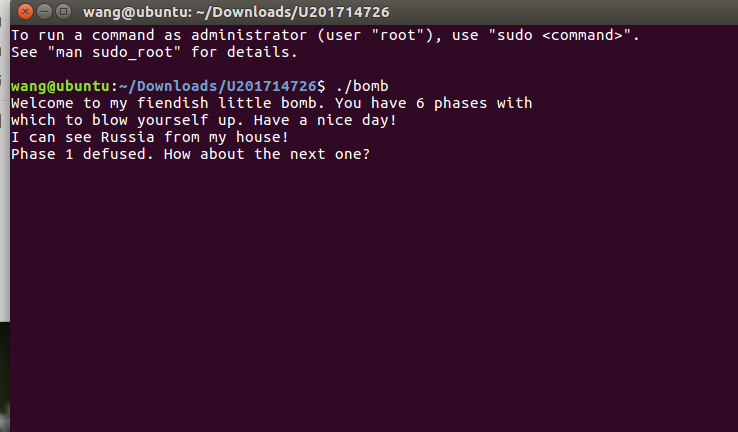


图2-4 结果截图

可见正确答案为“I can see Russia from my house!”。

## 2.2.2 阶段2 循环

1. 任务描述： 完成炸弹2的拆除

2. 实验设计： 观察分析phase\_2代码，使用gdb调试分析结果

3. 实验过程： 找到phase\_2代码段

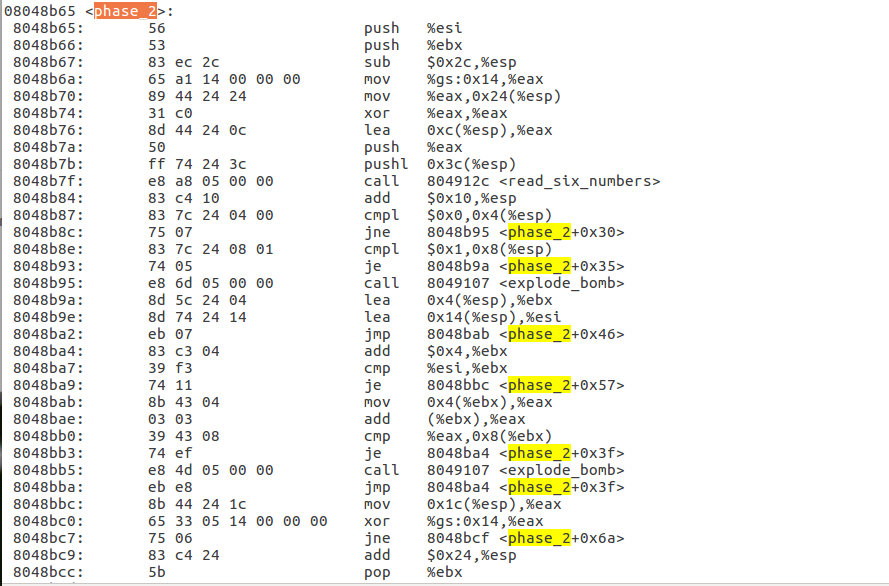


图2-5 phase\_2代码段截图

由read\_six\_numbers知是要输入六个数字，观察：

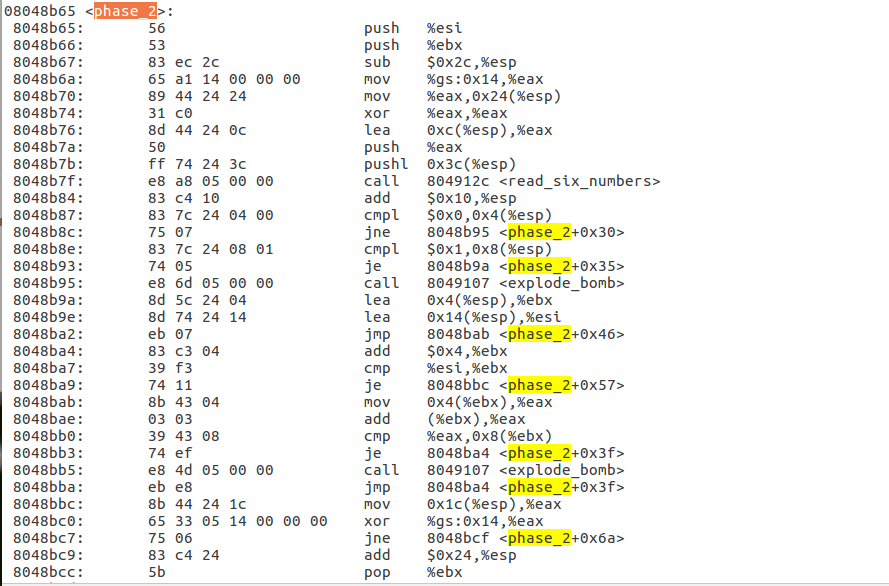


图2-6 read\_six\_numbers代码段截图

分析可知read\_six\_numbers功能为使输入为六个数字，否则会爆炸。

对phase\_2分析，可知：

let arr = int[6]

read\_six\_numbers(arr)

if arr[0] {[%esp+4]} != 0 => bomb

if arr[1] {[%esp+8]} == 0 => bomb

%ebx = arr {%esp+4}

%esi = arr + 6 {%esp+0x14}

L1:

%eax = \*(ebx+1) {[%ebx+4]}

%eax += \*(ebx) {[%ebx]}

if [%eax] != \*(ebx+2) {[%ebx+8]} => bomb

%ebx += 4

if %ebx == %esi => return

else => goto L1

即要求输入序列满足a[n] = a[n-1] + a[n-2] 且a[0] == 0, a[1] != 0。

故构造输入序列0 1 1 2 3 5。

输入序列0 1 1 2 3 5，测试结果如图2-7 所示：

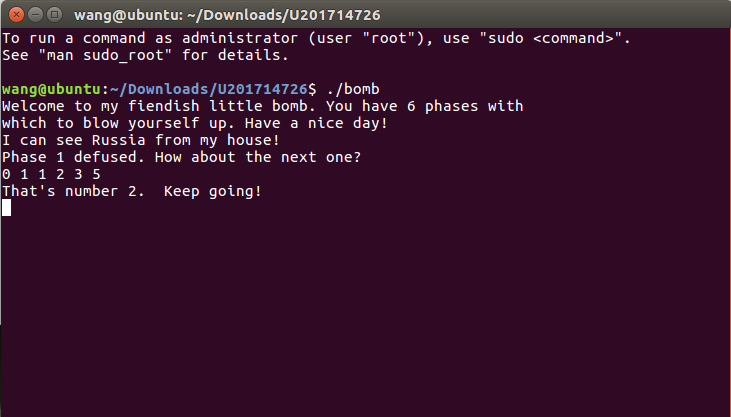


图2-7 阶段二测试截图

可知输入的序列正确。

## 2.2.3 阶段三 条件/分支

1. 任务描述： 完成炸弹三的拆除

2. 实验设计： 观察分析phase\_3代码段，使用gdb调试分析结果

3. 实验过程： 找到phase\_3代码段如下：

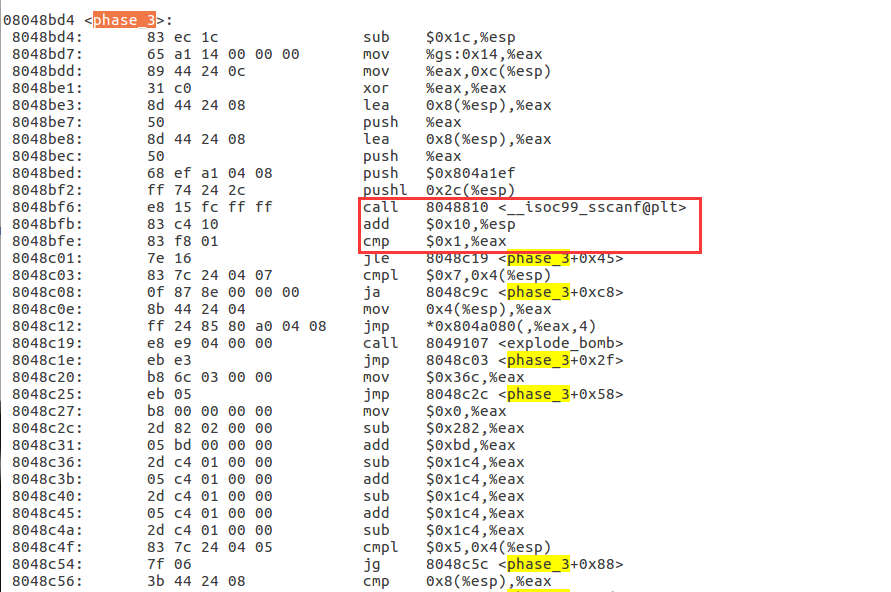


图2-8 phase\_3代码段截图

发现输入的应该要求\_isoc99\_sscanf@plt的返回值大于1，于是观察函数\_isoc99\_sscanf@plt，发现需要输入两个数字。继续观察分析phase\_3的反汇编代码，如图2-9所示：

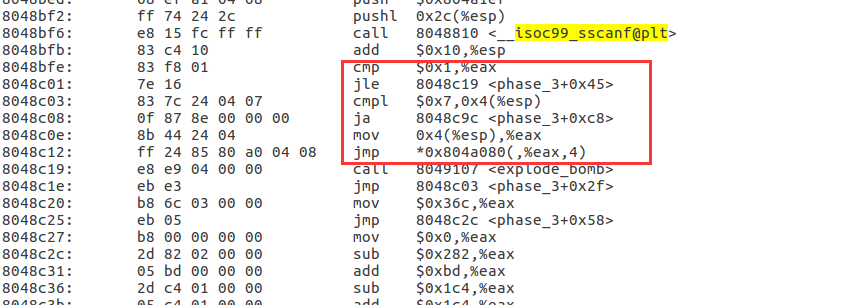


图2-9 phase\_3部分反汇编代码

不难发现，第一个数字不能大于7，并将其存放在eax中，然后跳转到0x804a080+eax\*4地址处，取第一个数字为0，那么，找到该地址存放的值，即为下条指令的地址，如图2-10所示：

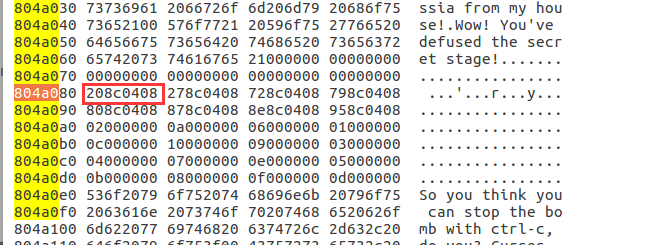


图2-10 下一条指令地址

由图2-10可知，指令地址为：08048c20，找到该地址对应的指令，如图2-11 所示：

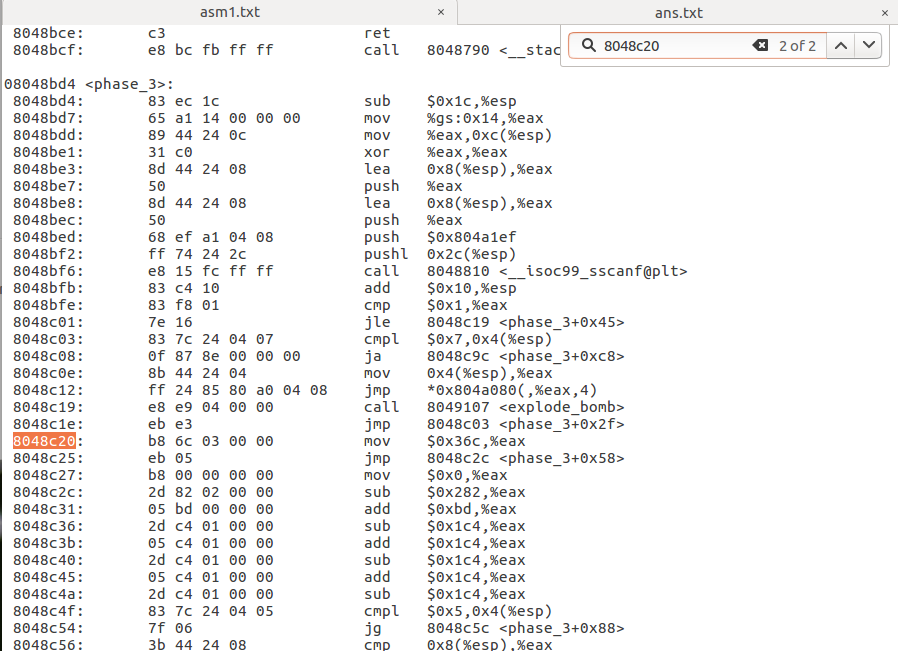


图2-11 之后的指令截图

进行计算，可知与0对应的第二个数字为：-29，输入测试数据，测试结果如图2-12所示：

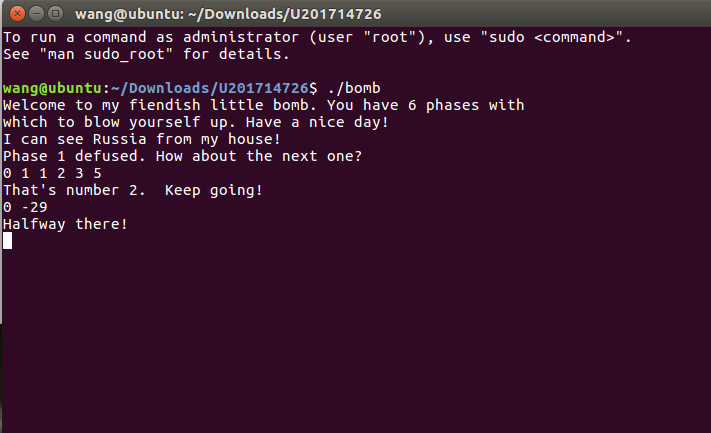


图2-12 阶段三拆弹测试结果

可见，第三阶段用到了分支，只需要沿着一条分支走就可以，因此第三阶段输入应有多个结果，其中一组结果为0，-29

## 2.2.4 阶段四 递归调用和栈

1. 任务描述： 通过phase\_4以及fun4的反汇编代码推断第四阶段要输入的数据。

2. 实验设计： 利用gdb结合断点来动态地分析

3. 实验过程： 观察phase\_4的前一部分反汇编代码，如图2-13所示：

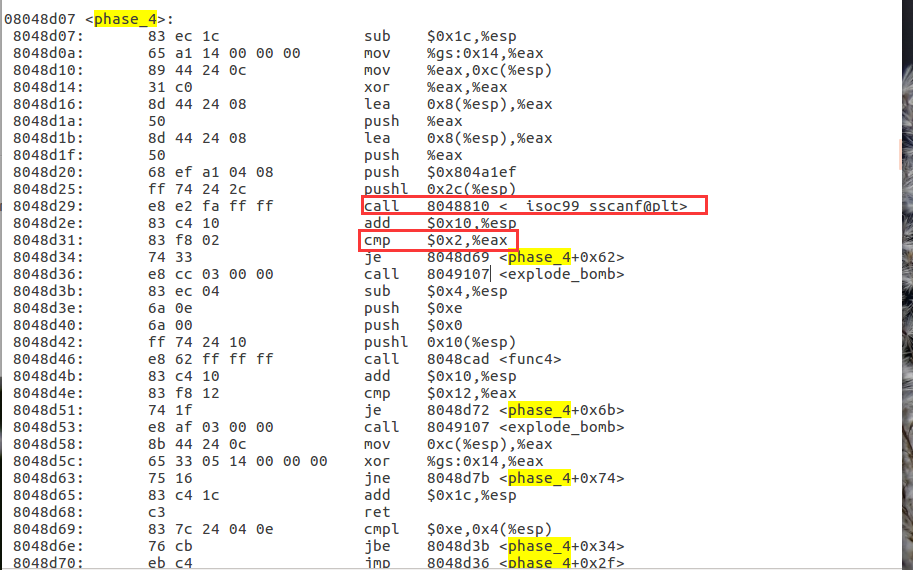


图2-13 phase\_4部分反汇编代码

发现输入的要求仍然是两个数字，而且要求第二个数字小于等于18，观察后半段代码可知返回值要为18，且第二个数字为18。

继续观察fun4的反汇编代码，如图2-14所示：

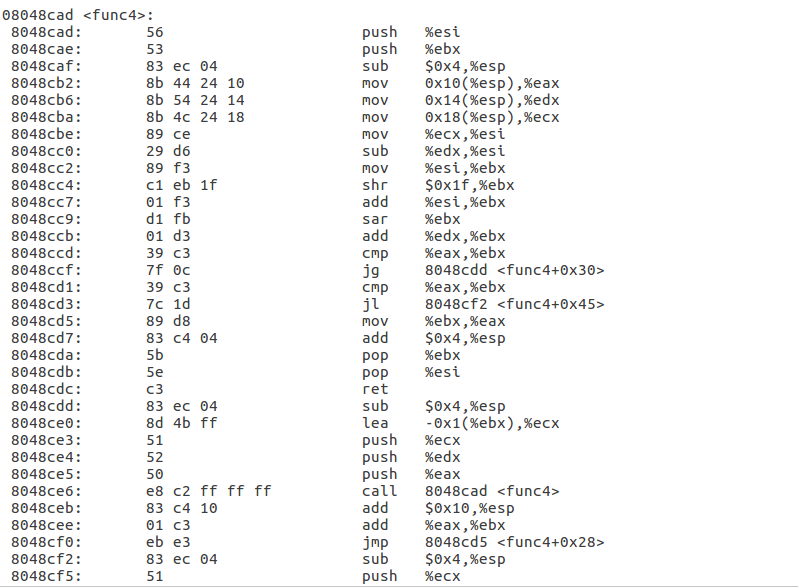


图2-14 fun4反汇编代码

可用C语言表述上述逻辑：

#include <stdio.h>

#include <stdlib.h>

int fun4(int eax, int edx, int ecx);

int main() {

int i, j;

for (i = 1; i <= 14; i++) {

j = fun4(i, 0, 14);

if (j == 18) {

printf("第一个值是：%d\n", i);

}

}

}

int fun4(int eax, int edx, int ecx) {

int ebx;

ebx = ecx - edx;

ebx = ebx / 2;

ebx = ebx + edx;

if (ebx > eax) {

ecx = ebx - 1;

return ebx + fun4(eax, edx, ecx);

}

if (ebx < eax) {

edx = ebx + 1;

return ebx + fun4(eax,edx,ecx);

}

if (ebx == eax) {

return ebx;

}

}

测试得到第一个数字为：

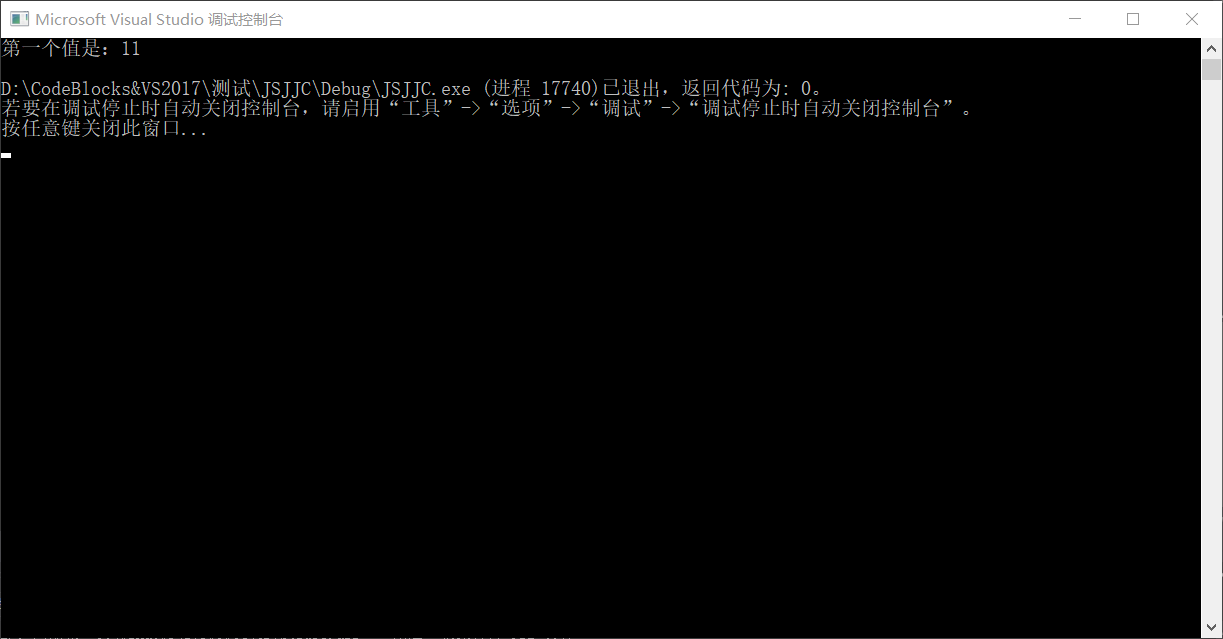


图2-15 测试得到第一个数字

因此输入数据应为11，18

进行测试，结果如图2-16所示：

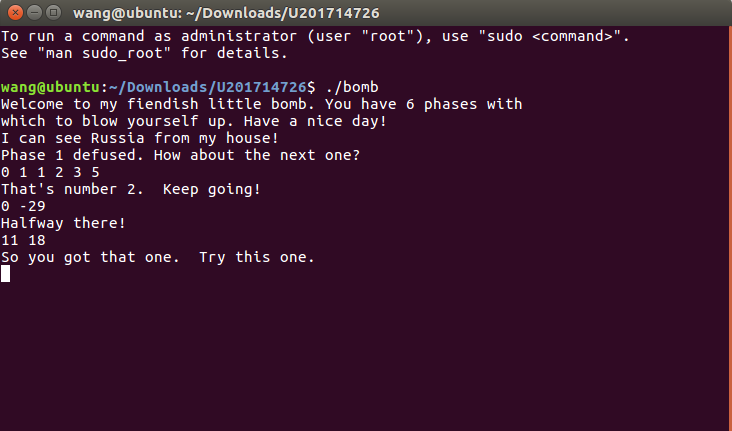


图2-16 测试结果截图

因此阶段四调用了别的函数，而且函数中还有递归调用，要求能够看懂函数功能，在进行反推。

## 2.2.5 阶段五 指针

1. 任务描述： 通过phase\_5的反汇编代码推断第五阶段需要输入的数据

2. 实验设计： 利用gdb结合断点来动态分析

3. 实验过程： 观察phase\_5的前一部分反汇编代码，如图2-17 所示

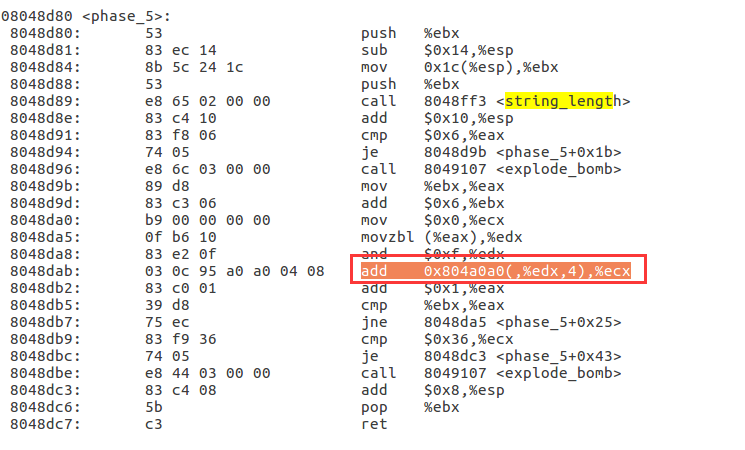


图2-17 phase\_5代码段截图

分析：string\_length要求输入为6个字符，否则爆炸。

需要将一片数组区域存储的16进制字符进行相加，使其得到的结果为0X36

该数组存放的数据如图2-18所示：

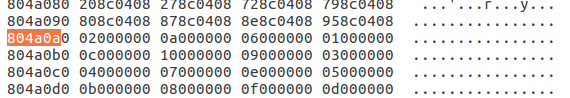


图2-18 存放数据截图

可知数据依次为：02h,0ah,06h,01h,0ch,10h,09h,03h,04h,07h,0eh,05h,0bh,08h,0fh,0dh

因此，若要从上述数据中选出六个相加结果为0x36，可选6个09h，对应下标为666666

因此输入数据可为666666，测试如图2-19 所示：

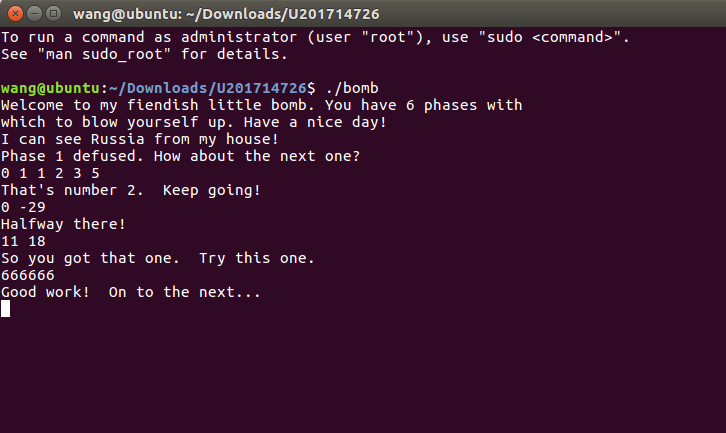


图2-19 阶段五测试截图

## 2.2.6 阶段六 链表

1. 任务描述： 通过phase\_6的反汇编代码推断第五阶段要输入的数据

2. 实验设计： 利用IDA反汇编进行分析

3. 实验过程： 观察逻辑框图，找出突破点，如图2-20所示：

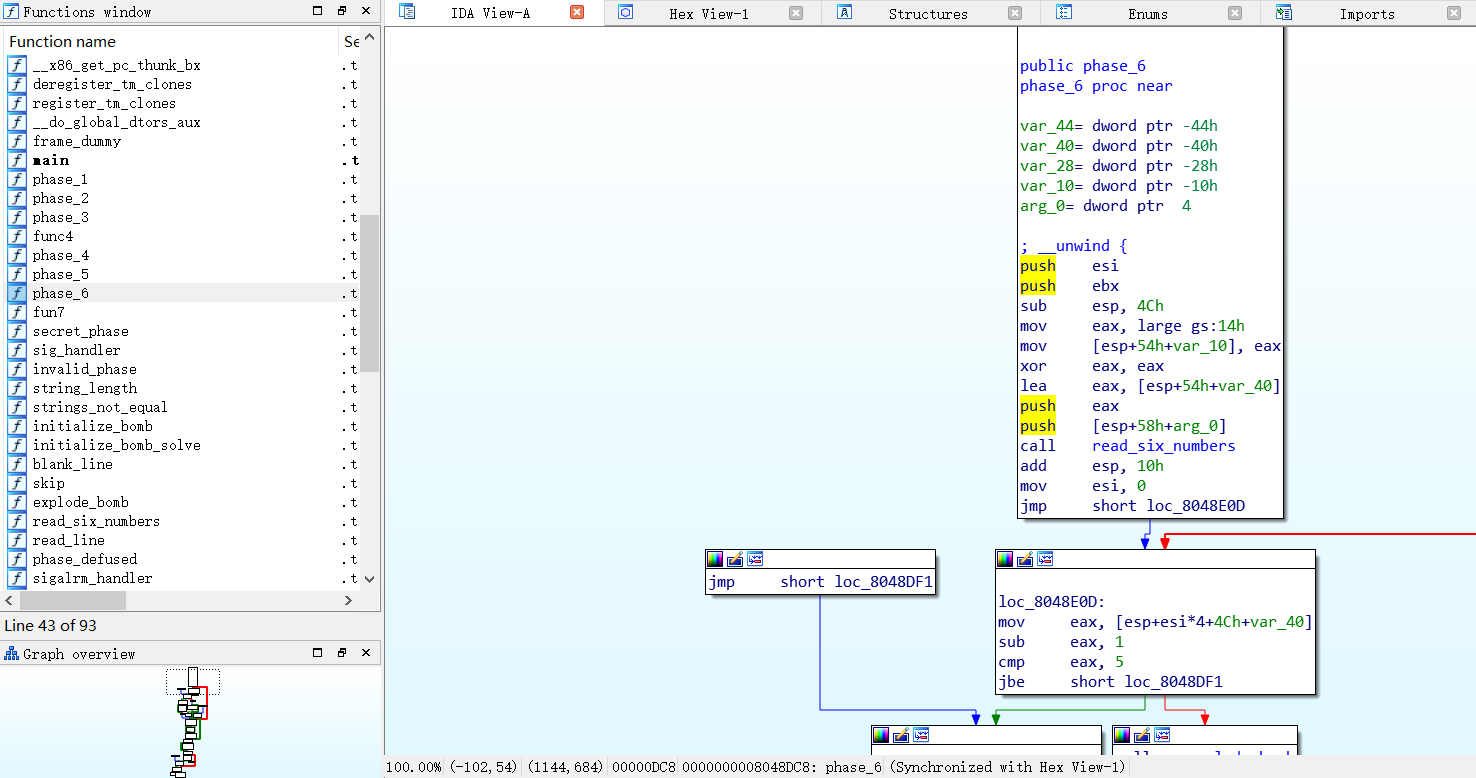


图2-20 phase\_6 IDA反汇编截图

观察phase\_6的反汇编代码，如图2-21所示：

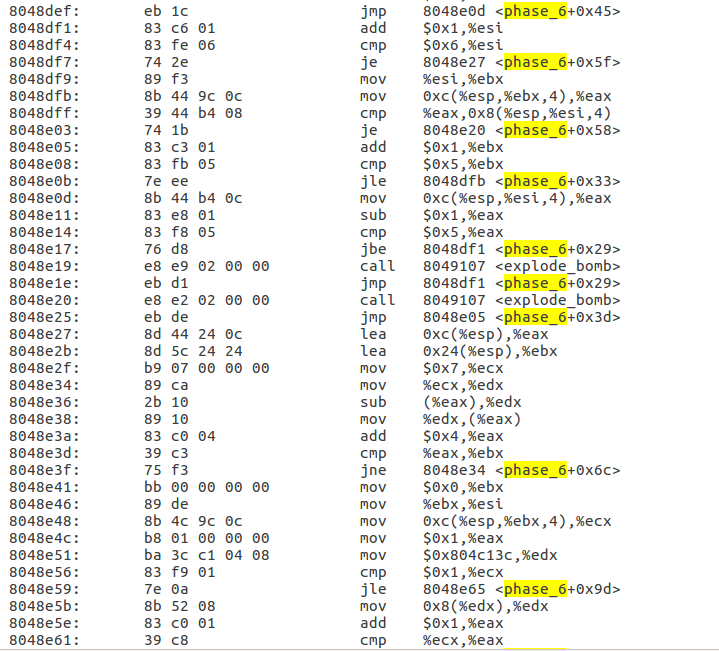


图2-21 phase\_6的反汇编代码

分析可知，根据输入的序列，将链表的顺序进行改排，并且要将输入的序列对7取补，然后根据改排后的链表中的数据是否满足从大到小进行排序的。

节点数据如图2-22所示：

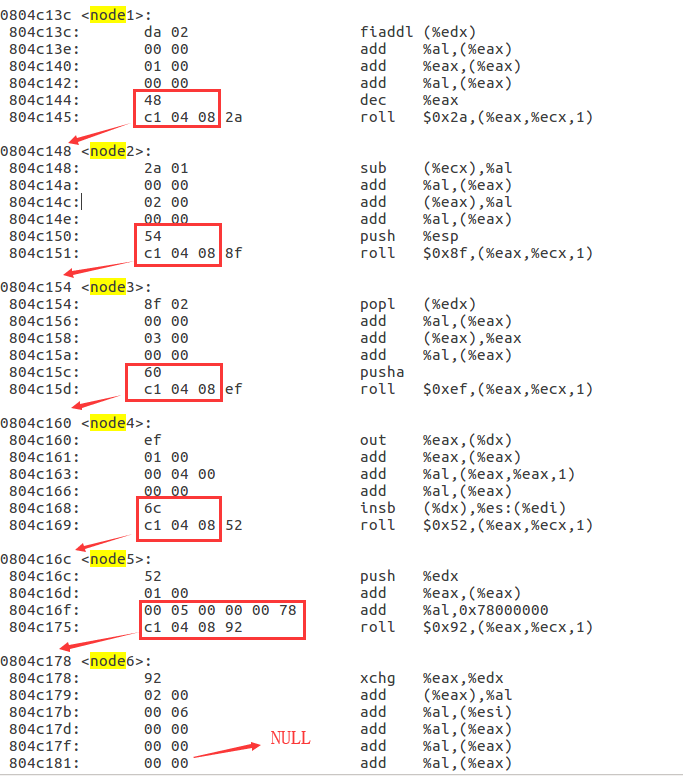


图2-22 链表节点值

因此，链表节点的顺序是node1🡪node2🡪node3🡪node4🡪node5🡪node6🡪NULL：

根据节点的值进行排序为：

node1(02daH)>node6(0292H)>node3(028fH)>node4(01efH)>node5(0152H)>node2(012aH)

所以输入节点顺序对应的下标为1、6、3、4、5、2

对7取补后为：6、1、4、3、2、5

测试结果如图2-23所示：

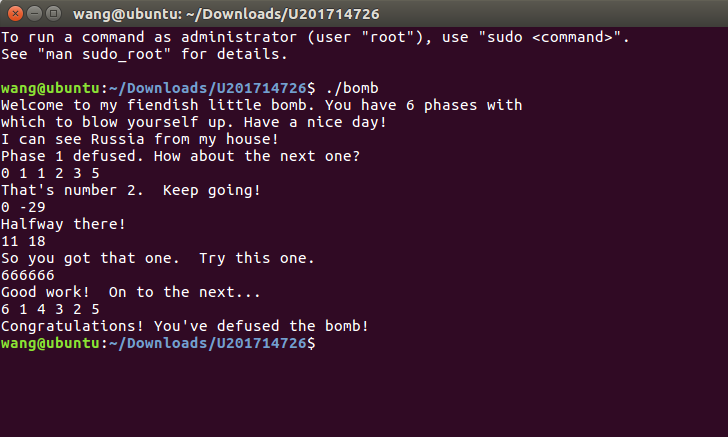


图2-23 阶段六测试截图

## 2.2.7 阶段七 隐藏阶段

由于phase\_4中没有指向隐藏任务的代码，故从phase\_4调用后的下一个函数phase\_defused入手，在其中找到strings\_not\_equal函数调用，与任务一一样，根据地址参数找到所在的数据段的值，为"DrEvil"。在任务四答案后附上该字符串，成功进入隐藏任务。如图2-24和2-25所示：

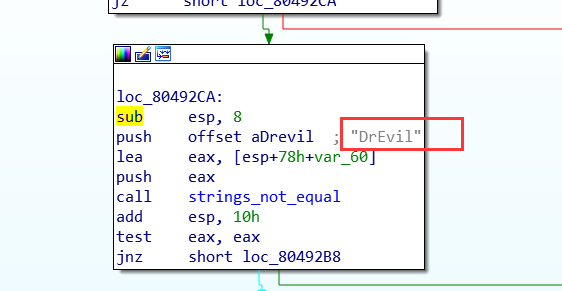


图2-24 隐藏阶段字符串

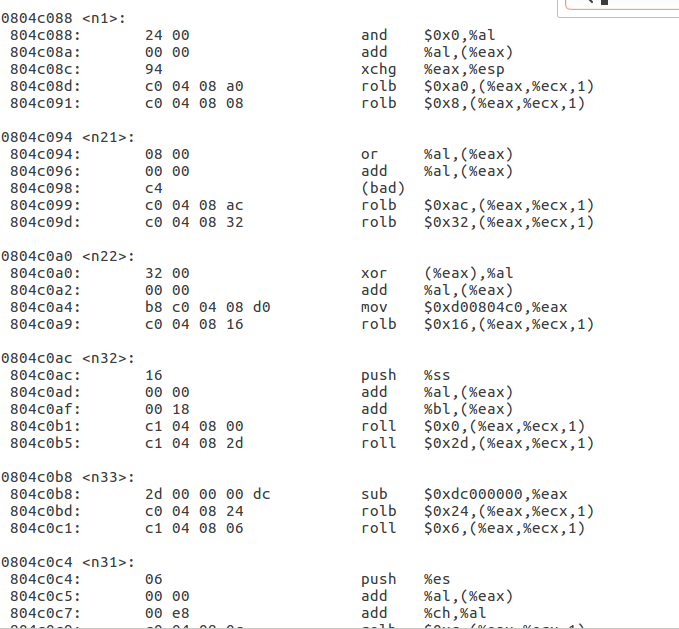


图2-25 部分节点数据

对fun7进行分析，得出的C代码如下：

int fun7(node\* n,int num) {

if (n == nullptr)return -1;

if (n.x > num)

return fun7(n.left,num) \* 2;

if (n.x == num)

return 0;

return fun7(n.right,num) \* 2 + 1;

}

进入隐藏任务截图，输入正确节点值，如图2-26所示：

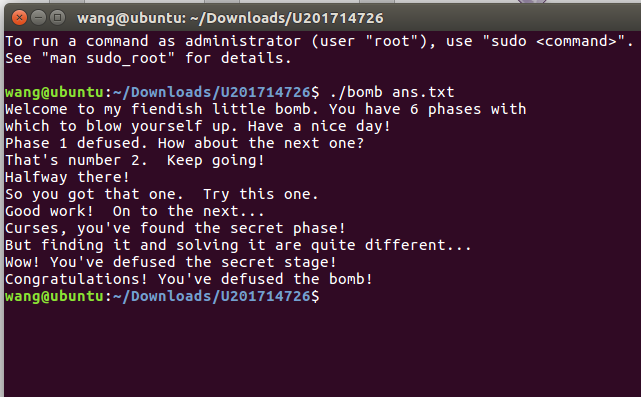


图2-26 通过测试截图

# 2.3 总结

# 附录 ans.txt

I can see Russia from my house!

0 1 1 2 3 5

0 -29

11 18 DrEvil

666666

6 1 4 3 2 5

107