**深 圳 大 学 实 验 报 告**

**课 程 名 称： 计算机系统(2)**

**实验项目名称： 逆向工程实验**

**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： 洪继耀 学号： 2014150120**

**班级： 2**

**实 验 时 间： 2016年4月25日**

**实验报告提交时间： 2016年5月19日**

**教务处制**

**一、 实验目标：**

1. 理解程序（控制语句、函数、返回值、堆栈结构）是如何运行的
2. 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. Linux32位操作系统（Fedora 13）
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拆解二进制炸弹的游戏。我们仅给黑客（同学）提供一个二进制可执行文件bomb和主函数所在的源程序bomb.c，不提供每个关卡的源代码。程序运行中有6个关卡（6个phase），每个关卡需要用户输入正确的字符串或数字才能通关，否则会引爆炸弹（打印出一条错误信息，并导致评分下降）！

要求同学运用**GDB调试工具和objdump反汇编工具**，通过分析汇编代码**，**找到在每个phase程序段中，引导程序跳转到“explode\_bomb”程序段的地方，并分析其成功跳转的条件，以此为突破口寻找应该在命令行输入何种字符串来通关。本实验要求解决Phase\_1(**15分**)、Phase\_2(**10分**)、Phase\_3(**10分**)、Phase\_4(**10分**)、Phase\_5(**10分**)。通过截图把结果写在实验报告上。

1. **实验步骤和结果**

**各个部分的汇编代码如下：**

**phase\_1**

08048b20 <phase\_1>:

8048b20: 55 push %ebp

8048b21: 89 e5 mov %esp,%ebp

8048b23: 83 ec 08 sub $0x8,%esp

8048b26: 8b 45 08 mov 0x8(%ebp),%eax

8048b29: 83 c4 f8 add $0xfffffff8,%esp

**//** 准备两个入口参数 一个是**%eax**即用户输入的字符串 另外一个是$0x80497c0上的字符串

8048b2c: 68 c0 97 04 08 push $0x80497c0

8048b31: 50 push %eax



**//** 比对**R[%eax]**和**M[0x80497c0]**  结果放在**%eax**

8048b32: e8 f9 04 00 00 call 8049030 <strings\_not\_equal>

**//** 跳转条件**eax==0** 即两字符串相等

8048b37: 83 c4 10 add $0x10,%esp

8048b3a: 85 c0 test %eax,%eax

8048b3c: 74 05 je 8048b43 <phase\_1+0x23>

8048b3e: e8 b9 09 00 00 call 80494fc <explode\_bomb>

**//** 否则引爆炸弹

8048b43: 89 ec mov %ebp,%esp

8048b45: 5d pop %ebp

8048b46: c3 ret

8048b47: 90 nop

第一道题是送分题，它将%eax上保存的用户输入的字符串和保存在0x80497c0上的字符串比较，如果相等则%eax为0， test %eax %eax后%eax 为0 跳出函数。于是我们用gdb设置好断点，然后打印0x80497c0上的内容，可以得到: “Public speaking is very easy.”此为答案。

**phase\_2**

08048b48 <phase\_2>:

8048b48: 55 push %ebp

8048bi49: 89 e5 mov %esp,%ebp

8048b4b: 83 ec 20 sub $0x20,%esp

**// esi ebx**是调用者负责保存的 要使用他们的值 就必须先保存旧值

8048b4e: 56 push %esi

8048b4f: 53 push %ebx

**//** 用户输入一个字符串 它放在**%ebp+8**上

8048b50: 8b 55 08 mov 0x8(%ebp),%edx

8048b53: 83 c4 f8 add $0xfffffff8,%esp

**//** 准备入口参数

8048b56: 8d 45 e8 lea -0x18(%ebp),%eax

**// 0x18=24=4\*6**

8048b59: 50 push %eax

**//** 入口参数二**%eax** 内容是 **-0x18(%ebp)**的引用

**//** 也就是返回结果存在**-0x18(%ebp)**

8048b5a: 52 push %edx

**//** 入口参数一**%edx**内容是用户输入的字符串

**//** 调用**read\_six\_numbers** 从用户输入的字符串中提取**6**个数字

**//** 数字放到长度为**6**的数组中，首地址为参数二即-**0x18(%ebp)**

8048b5b: e8 78 04 00 00 call 8048fd8 <read\_six\_numbers>

**//** 把第一个数字和**1**比较 如果第一个数不为**1** 引爆炸弹

8048b60: 83 c4 10 add $0x10,%esp

8048b63: 83 7d e8 01 cmpl $0x1,-0x18(%ebp)

8048b67: 74 05 je 8048b6e <phase\_2+0x26>

8048b69: e8 8e 09 00 00 call 80494fc <explode\_bomb>

//否则检查后面的数

8048b6e: bb 01 00 00 00 mov $0x1,%ebx **// ebx=n=1**

8048b73: 8d 75 e8 lea -0x18(%ebp),%esi

**// %esi**是数组首地址 也就是源变址

**//** 一维数组寻址： **%esi**源变址**+%ebx**变址\*比例因子

**//** 以下是递推公式 **%esi+4\*i-4 == （%esi+4\*i-8）\*i i**从**2**开始

**//** 也就是 **a[n]==（n+1） \* a[n-1]**

**// do**

8048b76: 8d 43 01 lea 0x1(%ebx),%eax

**// eax=ebx+1=n+1=i**

8048b79: 0f af 44 9e fc imul -0x4(%esi,%ebx,4),%eax

**// imul**有符号乘法

8048b7e: 39 04 9e cmp %eax,(%esi,%ebx,4)

**// cmp a[n-1]\*(n+1)与a[n]**

8048b81: 74 05 je 8048b88 <phase\_2+0x40>

8048b83: e8 74 09 00 00 call 80494fc <explode\_bomb>

**//** 不相等引爆

8048b88: 43 inc %ebx

**// %ebx++ (n++)**

8048b89: 83 fb 05 cmp $0x5,%ebx

**// while ebx<=5**

8048b8c: 7e e8 jle 8048b76 <phase\_2+0x2e>

8048b8e: 8d 65 d8 lea -0x28(%ebp),%esp

8048b91: 5b pop %ebx

8048b92: 5e pop %esi

8048b93: 89 ec mov %ebp,%esp

8048b95: 5d pop %ebp

8048b96: c3 ret

8048b97: 90 nop

这段程序调用了read\_six\_numbres，也就是说要读取6个数。而0x8048b63 行说明第一个数一定为1。再接下去几行是个递推公式，它的代码意思是数组的数服从a[n]=a[n-1]\*(n+1)

那么数组里的数就是1 2 6 24 120 720了

**phase\_3**

08048b98 <phase\_3>:

8048b98: 55 push %ebp

8048b99: 89 e5 mov %esp,%ebp

8048b9b: 83 ec 14 sub $0x14,%esp

**//**调用者保存**ebx**的旧值

8048b9e: 53 push %ebx

**//**传形式参数给临时变量（实际参数）

8048b9f: 8b 55 08 mov 0x8(%ebp),%edx

8048ba2: 83 c4 f4 add $0xfffffff4,%esp

**//**调用者准备入口参数

8048ba5: 8d 45 fc lea -0x4(%ebp),%eax //第5个参数 用户输入的第二个数

8048ba8: 50 push %eax

8048ba9: 8d 45 fb lea -0x5(%ebp),%eax //第4个参数 用户输入的字母

8048bac: 50 push %eax

8048bad: 8d 45 f4 lea -0xc(%ebp),%eax //第3个参数 用户输入的第一个数

8048bb0: 50 push %eax

**//**第二个参数是 **$0x80497de**  (格式字符串)

8048bb1: 68 de 97 04 08 push $0x80497de



**//**第一个参数是**%edx** 用户输入的字符串

8048bb6: 52 push %edx

**// sscanf@plt**的作用见印象笔记

8048bb7: e8 a4 fc ff ff call 8048860 <sscanf@plt>

8048bbc: 83 c4 20 add $0x20,%esp

**// %eax**为**sscanf**的返回值

**//**用户输入的字符串格式正确**（**符 合 **%d %c %d）**的话这里为3 大于2 否则输入不足 爆炸

8048bbf: 83 f8 02 cmp $0x2,%eax

8048bc2: 7f 05 jg 8048bc9 <phase\_3+0x31>

8048bc4: e8 33 09 00 00 call 80494fc <explode\_bomb>

**//** 以下代码说明第一个数小于等于**7**

8048bc9: 83 7d f4 07 cmpl $0x7,-0xc(%ebp)

8048bcd: 0f 87 b5 00 00 00 ja 8048c88 <phase\_3+0xf0>

8048bd3: 8b 45 f4 mov -0xc(%ebp),%eax

**//**将第一个数放到**%eax**

**//** 以下是跳转语句：**0x8048bd6（0x80497e8**为起始地址**，%eax×4**为偏移**）**

**//** 这道题第一个数只要**0-7**就行 而后面的字母和数字应该输什么取决于第一个数

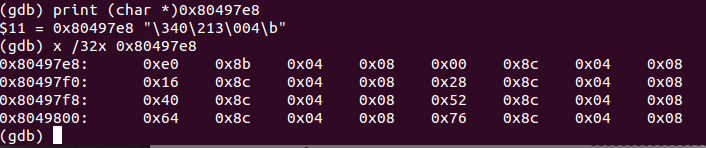
**//** 这里假定用户输入的数为**%eax=0** 总共有**8**种可能

**//** 跳转向**\*(0x80497e8+ %eax\*4)=0x80497e8**

**//**用**gdb** 调试**0x80497e8**的作为字符串的值为**"\340\213\004\b"**

**//**不知所云

**//**接着换成**32**位数字格式解析**0x80497E8**



**//** 发现恰好是**8**个**phrase3**的代码地址 恰好第一个数字的取值有**8**种

**//** 四个一组 （从右往左读因为是小端法）

**// 0x08 04 8b e0 =0x8048be0 (eax==0)**

**// 0x8048c00 (eax==1)**

**// 0x8048c16 (eax==2)**

**// ...**以下略

**//** 由此推测  **0x8048bd6**是根据输入的第一个参数跳转到对应位置

**//** 也就是一个使用**0x80497e8**作为跳转表的**switch**语句

**//** 当**%eax==0**时 跳转到**0x08048be0**的位置

8048bd6: ff 24 85 e8 97 04 08 jmp \*0x80497e8(,%eax,4)

8048bdd: 8d 76 00 lea 0x0(%esi),%esi

**// eax=0** 跳转到这

8048be0: b3 71 mov $0x71, %bl

**//** 参考**0x8048c8f**的代码得出第二个参数应与**%bl**相等 此处**%bl='q'**

8048be2: 81 7d fc 09 03 00 00 cmpl $0x309,-0x4(%ebp)

**//** 第三个参数应为**0x309 (777)**

8048be9: 0f 84 a0 00 00 00 je 8048c8f <phase\_3+0xf7>

8048bef: e8 08 09 00 00 call 80494fc <explode\_bomb>

8048bf4: e9 96 00 00 00 jmp 8048c8f <phase\_3+0xf7>

8048bf9: 8d b4 26 00 00 00 00 lea 0x0(%esi,%eiz,1),%esi

8048c00: b3 62 mov $0x62,%bl

**// eax=1 bl='b'**

8048c02: 81 7d fc d6 00 00 00 cmpl $0xd6,-0x4(%ebp)

**//** 第三个参数**214**

8048c09: 0f 84 80 00 00 00 je 8048c8f <phase\_3+0xf7>

8048c0f: e8 e8 08 00 00 call 80494fc <explode\_bomb>

8048c14: eb 79 jmp 8048c8f <phase\_3+0xf7>

8048c16: b3 62 mov $0x62,%bl **// eax=2 bl='b'**

8048c18: 81 7d fc f3 02 00 00 cmpl $0x2f3,-0x4(%ebp)

**//** 第三个参数 **755**

8048c1f: 74 6e je 8048c8f <phase\_3+0xf7>

8048c21: e8 d6 08 00 00 call 80494fc <explode\_bomb>

8048c26: eb 67 jmp 8048c8f <phase\_3+0xf7>

8048c28: b3 6b mov $0x6b,%bl

8048c2a: 81 7d fc fb 00 00 00 cmpl $0xfb,-0x4(%ebp)

8048c31: 74 5c je 8048c8f <phase\_3+0xf7>

8048c33: e8 c4 08 00 00 call 80494fc <explode\_bomb>

8048c38: eb 55 jmp 8048c8f <phase\_3+0xf7>

8048c3a: 8d b6 00 00 00 00 lea 0x0(%esi),%esi

8048c40: b3 6f mov $0x6f,%bl **//** 以下略

8048c42: 81 7d fc a0 00 00 00 cmpl $0xa0,-0x4(%ebp)

8048c49: 74 44 je 8048c8f <phase\_3+0xf7>

8048c4b: e8 ac 08 00 00 call 80494fc <explode\_bomb>

8048c50: eb 3d jmp 8048c8f <phase\_3+0xf7>

8048c52: b3 74 mov $0x74,%bl

8048c54: 81 7d fc ca 01 00 00 cmpl $0x1ca,-0x4(%ebp)

8048c5b: 74 32 je 8048c8f <phase\_3+0xf7>

8048c5d: e8 9a 08 00 00 call 80494fc <explode\_bomb>

8048c62: eb 2b jmp 8048c8f <phase\_3+0xf7>

8048c64: b3 76 mov $0x76,%bl

8048c66: 81 7d fc 0c 03 00 00 cmpl $0x30c,-0x4(%ebp)

8048c6d: 74 20 je 8048c8f <phase\_3+0xf7>

8048c6f: e8 88 08 00 00 call 80494fc <explode\_bomb>

8048c74: eb 19 jmp 8048c8f <phase\_3+0xf7>

8048c76: b3 62 mov $0x62,%bl

8048c78: 81 7d fc 0c 02 00 00 cmpl $0x20c,-0x4(%ebp)

8048c7f: 74 0e je 8048c8f <phase\_3+0xf7>

8048c81: e8 76 08 00 00 call 80494fc <explode\_bomb>

8048c86: eb 07 jmp 8048c8f <phase\_3+0xf7>

8048c88: b3 78 mov $0x78,%bl

8048c8a: e8 6d 08 00 00 call 80494fc <explode\_bomb>

8048c8f: 3a 5d fb cmp -0x5(%ebp),%bl

**//** 对比字母参数与**%bl**是否相同

8048c92: 74 05 je 8048c99 <phase\_3+0x101>

8048c94: e8 63 08 00 00 call 80494fc <explode\_bomb>

**//** 否则爆炸

8048c99: 8b 5d e8 mov -0x18(%ebp),%ebx

8048c9c: 89 ec mov %ebp,%esp

8048c9e: 5d pop %ebp

一开始这段程序调用了sscanf@plt，它的第二个参数即0x80497de所在的地址是sscanf解析的格式，用gdb看到这个格式为%d %c %d。也就是说用户必须输入一个数字，一个字母，一个数字格式的字符串才能被正确解析

然后分析代码可知程序会根据第一个整数的值来进行跳转，

从跳转到的地方可以推测出第二个第三个参数的值，详见注释

本题的共有八个答案，其中一个答案为0 q 777。

**phase\_4**

08048ce0 <phase\_4>:

8048ce0: 55 push %ebp

8048ce1: 89 e5 mov %esp,%ebp

8048ce3: 83 ec 18 sub $0x18,%esp **// esp-24**

8048ce6: 8b 55 08 mov 0x8(%ebp),%edx

8048ce9: 83 c4 fc add $0xfffffffc,%esp **// esp-8**

**//** 准备入口参数

8048cec: 8d 45 fc lea -0x4(%ebp),%eax

**//** 引用临时变量

8048cef: 50 push %eax

**//** 参数**3 eax** 存放解析结果 此处为用户输入的数字

8048cf0: 68 08 98 04 08 push $0x8049808

**//** 参数**2** 格式字符串**”%d”**



8048cf5: 52 push %edx

**//** 参数1 用户输入的字符串

**//** 开始解析 如果用户输入正确则返回**1**

8048cf6: e8 65 fb ff ff call 8048860 <sscanf@plt>

8048cfb: 83 c4 10 add $0x10,%esp

**//** 释放入口参数空间

**//** 检查用户是否按正确的格式输入（这里是只输入一个数字）

8048cfe: 83 f8 01 cmp $0x1,%eax

8048d01: 75 06 jne 8048d09 <phase\_4+0x29>

**//** 否则引爆炸弹

**//** 检查用户输入的值是否比0大

8048d03: 83 7d fc 00 cmpl $0x0,-0x4(%ebp)

8048d07: 7f 05 jg 8048d0e <phase\_4+0x2e>

8048d09: e8 ee 07 00 00 call 80494fc <explode\_bomb>

**//** 否则引爆炸弹

**//** 准备**func4**的参数（用户输入的数）

8048d0e: 83 c4 f4 add $0xfffffff4,%esp

8048d11: 8b 45 fc mov -0x4(%ebp),%eax

8048d14: 50 push %eax

**//** 调用**func4 func4**是个递归函数 作用是求斐波那契数列第**n**项

8048d15: e8 86 ff ff ff call 8048ca0 <func4>

8048d1a: 83 c4 10 add $0x10,%esp

**//** 释放入口参数空间

8048d1d: 83 f8 37 cmp $0x37,%eax

**//** 对比**func4**返回值与**55**的大小

**//** 仅当**func4**返回**55**的时候 也就是入口参数**%eax**为**9**的时候不引爆

**//** 也就是用户应该输入**9**

8048d20: 74 05 je 8048d27 <phase\_4+0x47>

8048d22: e8 d5 07 00 00 call 80494fc <explode\_bomb>

8048d27: 89 ec mov %ebp,%esp

8048d29: 5d pop %ebp

8048d2a: c3 ret

8048d2b: 90 nop

phase\_4调用了sscanf来解析用户输入的字符串，其第二个参数格式为”%d”，即要求用户输入一个整数。然后将这个整数传给func4，最后检查func4返回值与0x37也就是55是否相等。如果相等就通过不相等就爆炸。所以我们只要搞懂func4在做什么就可以了。以下是func4的汇编代码：

08048ca0 <func4>:

8048ca0: 55 push %ebp

8048ca1: 89 e5 mov %esp,%ebp

8048ca3: 83 ec 10 sub $0x10,%esp

**//** 调用者保存寄存器

8048ca6: 56 push %esi

8048ca7: 53 push %ebx

**//** 将参数**n**存到**ebx**

8048ca8: 8b 5d 08 mov 0x8(%ebp),%ebx

**// if (n<=1) return 1;** 递归基

8048cab: 83 fb 01 cmp $0x1,%ebx

8048cae: 7e 20 jle 8048cd0 <func4+0x30>

**// else a= func4(n-1)**

8048cb0: 83 c4 f4 add $0xfffffff4,%esp

8048cb3: 8d 43 ff lea -0x1(%ebx),%eax

**// %eax=n-1**

8048cb6: 50 push %eax

8048cb7: e8 e4 ff ff ff call 8048ca0 <func4>

8048cbc: 89 c6 mov %eax,%esi

**// %esi=a=func4(n-1)**

8048cbe: 83 c4 f4 add $0xfffffff4,%esp

8048cc1: 8d 43 fe lea -0x2(%ebx),%eax

**// %eax=n-2**

8048cc4: 50 push %eax

**// b= func4(n-2)**

8048cc5: e8 d6 ff ff ff call 8048ca0 <func4>

8048cca: 01 f0 add %esi,%eax

**// %eax = b = func4(n-1)+func4(n-2) return b;**

8048ccc: eb 07 jmp 8048cd5 <func4+0x35>

8048cce: 89 f6 mov %esi,%esi

8048cd0: b8 01 00 00 00 mov $0x1,%eax

**// return 1;**

8048cd5: 8d 65 e8 lea -0x18(%ebp),%esp

8048cd8: 5b pop %ebx

8048cd9: 5e pop %esi

8048cda: 89 ec mov %ebp,%esp

8048cdc: 5d pop %ebp

8048cdd: c3 ret

8048cde: 89 f6 mov %esi,%esi

func4非常简单 容易看出意思是求斐波那契数列第n项的值 即f(0)=f(1)=1(when n<=1)

f(n)=f(n-1)+f(n-2)(when n>=2)

然后问题等价于，第多少个斐波那契数等于0x37(55)，非常简单，答案是9。

**phase\_5**

08048d2c <phase\_5>:

8048d2c: 55 push %ebp

8048d2d: 89 e5 mov %esp,%ebp

8048d2f: 83 ec 10 sub $0x10,%esp

**//** 保存**esi ebx**旧值

8048d32: 56 push %esi

8048d33: 53 push %ebx

**//** 用户输入的字符串放入**ebx**作为入口参数 调用**string\_length**求长度

8048d34: 8b 5d 08 mov 0x8(%ebp),%ebx

8048d37: 83 c4 f4 add $0xfffffff4,%esp

8048d3a: 53 push %ebx

8048d3b: e8 d8 02 00 00 call 8049018 <string\_length>

**//** 释放空间

8048d40: 83 c4 10 add $0x10,%esp

**//** 检查输入字符串的长度**%eax**是否为**6** 不是则爆炸

8048d43: 83 f8 06 cmp $0x6,%eax

8048d46: 74 05 je 8048d4d <phase\_5+0x21>

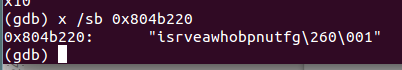
8048d48: e8 af 07 00 00 call 80494fc <explode\_bomb>

**//** 清空**edx**

8048d4d: 31 d2 xor %edx,%edx

**//** 引用局部变量

8048d4f: 8d 4d f8 lea -0x8(%ebp),%ecx



8048d52: be 20 b2 04 08 mov $0x804b220,%esi

**// %ebx**是用户输入的字符的起始地址，**%edx**初始化为**0**，所以接下来是遍历输入字符

**//** 根据用户输入的字符串**（**原文**）** 来生成新字符串**（**密文**）** 疑似加密

**//** 这里取原文的每一个字符的低四位做符号扩展(强转**int)** 作**index**

**//** 从**$0x804b220**上取位置**index**上的字符当做密文的对应位置

**//** 也就是 假设原文为**in** 密文为**mi** **$0x804b220**为s**tr**

**//** 那么**mi[n]=str[ int(in[n]&0xf) ] n=0,1,2,3,4,5 mi[6]=0='/0'**

8048d57: 8a 04 1a mov (%edx,%ebx,1),%al

8048d5a: 24 0f and $0xf,%al

**// %eax=( %al)&0xf**的操作就是取**%al**的低**4**位

8048d5c: 0f be c0 movsbl %al,%eax

**//** 符号扩展 说明**%eax**应被解读（强转）为**int**

**//**接下来 将**%esi（$0x804b220 s**首指针**）** **+ %eax**作为地址，取这个字符存到**%al**

8048d5f: 8a 04 30 mov (%eax,%esi,1),%al

**//** 又把**%al**存在局部变量**%edx + %ecx = -0x8(%ebp) （**密文首地址**）+ %edx**中

8048d62: 88 04 0a mov %al,(%edx,%ecx,1)

8048d65: 42 inc %edx

8048d66: 83 fa 05 cmp $0x5,%edx // %edx++ 依次处理6个字符

8048d69: 7e ec jle 8048d57 <phase\_5+0x2b>

**//** 补上密文字符串末尾的**0**

8048d6b: c6 45 fe 00 movb $0x0,-0x2(%ebp)

**//** 释放空间

8048d6f: 83 c4 f8 add $0xfffffff8,%esp

**//** 这个字符串内容为**“giants”**



8048d72: 68 0b 98 04 08 push $0x804980b

8048d77: 8d 45 f8 lea -0x8(%ebp),%eax

**//** 取得密文

8048d7a: 50 push %eax

**//** 调用 **strings\_not\_equal**比较密文和**”giants”**

8048d7b: e8 b0 02 00 00 call 8049030 <strings\_not\_equal>

**//** 当且仅当用户输入的原文加密后得到**“giants”** 炸弹才不爆炸 因此拆弹相当于解密

**//** 此处能加密得到**“giants”**的字符串不唯一 **Opukma**是一个例子

**//** 因此答案可以为**Opukma**

8048d80: 83 c4 10 add $0x10,%esp

8048d83: 85 c0 test %eax,%eax

8048d85: 74 05 je 8048d8c <phase\_5+0x60>

8048d87: e8 70 07 00 00 call 80494fc <explode\_bomb>

8048d8c: 8d 65 e8 lea -0x18(%ebp),%esp

8048d8f: 5b pop %ebx

8048d90: 5e pop %esi

8048d91: 89 ec mov %ebp,%esp

8048d93: 5d pop %ebp

8048d94: c3 ret

8048d95: 8d 76 00 lea 0x0(%esi),%esi

这段代码的作用是，将用户输入的字符串进行处理，得到的字符串和保存在0x804980b的字符串”giants”比较，相等就算过了这关了。

如果把处理字符串的过程看做加密，那么为了最后得到“giants”，我们需要通过giants 反推最开始应该输入什么，相当于解密。

加密过程如下：

遍历这个字符串，对于每个字符，取低4位，然后作为0x804b220地址的偏移，得到一个地址。取这个偏移后的地址上的字符作为密文对应位置的字符。比如原文‘O’的后四位是0xF,也就是要加密为0x804b220的第16个字母g

相应地，解密的做法就是，先对照”isrveawhobpnutfg”，“gaints”第一个字符是g，偏移量是0xF(16)，所以我们的第一个输入字符的后4位是0xF就可以了，即 ‘/’ 或 ‘?’ 或 ‘O’ 或 ‘\_’ 或 ‘o’ 都可以。

依此类推，第2、3、4、5、6个字符都是这么得到，此phase其中一个答案是Opukma。当然还有其它很多很多答案。

**phase\_6**

08048d98 <phase\_6>:

8048d98: 55 push %ebp

8048d99: 89 e5 mov %esp,%ebp

8048d9b: 83 ec 4c sub $0x4c,%esp

**//** 调用者保存寄存器

8048d9e: 57 push %edi

8048d9f: 56 push %esi

8048da0: 53 push %ebx

**//** 取得输入参数

8048da1: 8b 55 08 mov 0x8(%ebp),%edx

**//** 取得**$0x804b26c**的值 经过**gdb**调试发现是个结点 编号是**node1**

8048da4: c7 45 cc 6c b2 04 08 movl $0x804b26c,-0x34(%ebp)

**//** 释放空间

8048dab: 83 c4 f8 add $0xfffffff8,%esp

**//** 取得局部变量引用

8048dae: 8d 45 e8 lea -0x18(%ebp),%eax

**// 0x18=24=4\*6**

**//** 准备**read\_six\_numbers**的入口参数 参数**2**为存放读取结果的数组首地址

8048db1: 50 push %eax

**//** 参数**1**为待读取的用户输入的字符串

8048db2: 52 push %edx

**//** 读取**6**个数，存在 **%ebp - 0x18** 开始的地址中

8048db3: e8 20 02 00 00 call 8048fd8 <read\_six\_numbers>

**//** 清空**%edi**

8048db8: 31 ff xor %edi,%edi

**//** 释放空间

8048dba: 83 c4 10 add $0x10,%esp

8048dbd: 8d 76 00 lea 0x0(%esi),%esi

**//** 下面的代码检查**6**个数是否两两不同 以防后面有**bug**

**//** 也检查**6**个数里面是否有大于**6**的数

**//** 检查不通过的话引爆炸弹

8048dc0: 8d 45 e8 lea -0x18(%ebp),%eax

8048dc3: 8b 04 b8 mov (%eax,%edi,4),%eax

8048dc6: 48 dec %eax

8048dc7: 83 f8 05 cmp $0x5,%eax

8048dca: 76 05 jbe 8048dd1 <phase\_6+0x39>

8048dcc: e8 2b 07 00 00 call 80494fc <explode\_bomb>

8048dd1: 8d 5f 01 lea 0x1(%edi),%ebx

8048dd4: 83 fb 05 cmp $0x5,%ebx

8048dd7: 7f 23 jg 8048dfc <phase\_6+0x64>

8048dd9: 8d 04 bd 00 00 00 00 lea 0x0(,%edi,4),%eax

8048de0: 89 45 c8 mov %eax,-0x38(%ebp)

8048de3: 8d 75 e8 lea -0x18(%ebp),%esi

8048de6: 8b 55 c8 mov -0x38(%ebp),%edx

8048de9: 8b 04 32 mov (%edx,%esi,1),%eax

8048dec: 3b 04 9e cmp (%esi,%ebx,4),%eax

8048def: 75 05 jne 8048df6 <phase\_6+0x5e>

8048df1: e8 06 07 00 00 call 80494fc <explode\_bomb>

**// %ebx++** 说明**%ebx**为计数器

8048df6: 43 inc %ebx

8048df7: 83 fb 05 cmp $0x5,%ebx

8048dfa: 7e ea jle 8048de6 <phase\_6+0x4e>

8048dfc: 47 inc %edi

8048dfd: 83 ff 05 cmp $0x5,%edi

8048e00: 7e be jle 8048dc0 <phase\_6+0x28>

**//** 以上

**//** 清空**%edi**

8048e02: 31 ff xor %edi,%edi

**//** 取得首地址

8048e04: 8d 4d e8 lea -0x18(%ebp),%ecx

**// A[-0x30(%ebp)]->-0x3c(%ebp)**

8048e07: 8d 45 d0 lea -0x30(%ebp),%eax

8048e0a: 89 45 c4 mov %eax,-0x3c(%ebp)

8048e0d: 8d 76 00 lea 0x0(%esi),%esi

.lab3

**// .lab3 - .lab2** 根据用户输入来取得对应索引编号

**//** 取得结点1地址

8048e10: 8b 75 cc mov -0x34(%ebp),%esi

**// %ebx=1**

8048e13: bb 01 00 00 00 mov $0x1,%ebx

**//%edx=%eax=4\*%edi=4n**

8048e18: 8d 04 bd 00 00 00 00 lea 0x0(,%edi,4),%eax

8048e1f: 89 c2 mov %eax,%edx

**// cmp %eax （**即**4\*%edi）+ %ecx（**键入的数组首地址**）** 和 **%ebx**

**//** 也就是 如果**in[n] <= %ebx** 跳到 **.lab1**

8048e21: 3b 1c 08 cmp (%eax,%ecx,1),%ebx

8048e24: 7d 12 jge .lab1

**// %eax = in[n]**

8048e26: 8b 04 0a mov (%edx,%ecx,1),%eax

**// %esi=%esi+%eiz**

8048e29: 8d b4 26 00 00 00 00 lea 0x0(%esi,%eiz,1),%esi

.lab2

8048e30: 8b 76 08 mov 0x8(%esi),%esi

**//%ebx ++**

8048e33: 43 inc %ebx

**// if(%eax==%ebx)**

8048e34: 39 c3 cmp %eax,%ebx

8048e36: 7c f8 jl .lab2

.lab1:

**//** 将对应的链表结点指针按序存放**，\*(%ebp - 0x3c)**为指针数组的起始地址

**// %edi=index**

8048e38: 8b 55 c4 mov -0x3c(%ebp),%edx

8048e3b: 89 34 ba mov %esi,(%edx,%edi,4)

8048e3e: 47 inc %edi

8048e3f: 83 ff 05 cmp $0x5,%edi

**//** 总共取6次

8048e42: 7e cc jle .lab3

.lab7

8048e44: 8b 75 d0 mov -0x30(%ebp),%esi

8048e47: 89 75 cc mov %esi,-0x34(%ebp)

8048e4a: bf 01 00 00 00 mov $0x1,%edi

8048e4f: 8d 55 d0 lea -0x30(%ebp),%edx

.lab4

**//** 以下建立新链表，上面 **.lab3 - .lab7**已经将6个节点的地址按顺序存下来了

**//** 存在**\*(%ebp - 0x3c)**开头的6位数组里

8048e52: 8b 04 ba mov (%edx,%edi,4),%eax

8048e55: 89 46 08 mov %eax,0x8(%esi)

8048e58: 89 c6 mov %eax,%esi

8048e5a: 47 inc %edi

8048e5b: 83 ff 05 cmp $0x5,%edi

8048e5e: 7e f2 jle .lab4

8048e60: c7 46 08 00 00 00 00 movl $0x0,0x8(%esi)

8048e67: 8b 75 cc mov -0x34(%ebp),%esi

8048e6a: 31 ff xor %edi,%edi

8048e6c: 8d 74 26 00 lea 0x0(%esi,%eiz,1),%esi

.lab6

**//** 以下检查新链表是否按从大到小的顺序排序

8048e70: 8b 56 08 mov 0x8(%esi),%edx

8048e73: 8b 06 mov (%esi),%eax

8048e75: 3b 02 cmp (%edx),%eax

8048e77: 7d 05 jge .lab5

**//** 前面一个数必须比后一个数大

8048e79: e8 7e 06 00 00 call 80494fc <explode\_bomb>

.lab5

**//** 检查 计数器**%edi**是否大于**4**

8048e7e: 8b 76 08 mov 0x8(%esi),%esi

8048e81: 47 inc %edi

8048e82: 83 ff 04 cmp $0x4,%edi

8048e85: 7e e9 jle .lab6

**//** 以上 结束重建

.lab8

**//** 还原现场

8048e87: 8d 65 a8 lea -0x58(%ebp),%esp

8048e8a: 5b pop %ebx

8048e8b: 5e pop %esi

8048e8c: 5f pop %edi

8048e8d: 89 ec mov %ebp,%esp

8048e8f: 5d pop %ebp

8048e90: c3 ret

8048e91: 8d 76 00 lea 0x0(%esi),%esi

原汇编代码实在太乱，有许多个jmp，所以我把一部分jmp地地址换成了标签(.labx)，这样看起来会清楚很多。

这个phase做的事情是按照我们输入的6个数对一个链表进行重排，使得最后的链表必须从大到小排序才能拆弹。

重排的方式是，用户输入数n，然后取出原链表第n项，放在新链表最后一个结点的next项。形成新链表。

因此正确的用户输入是，按原链表数值大小输入其编号，这样得到的新链表一定是顺序的。

而地址0x804b26c保存着原链表第一个节点的地址。

这个节点的结构为:

struct st

{

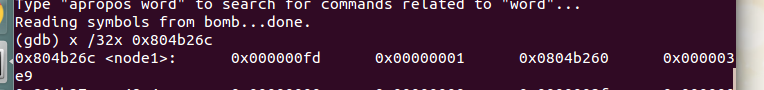
int data1;//排序依据

int data2;//没啥卵用

st\* next;

}

通过gdb观察0x804b26c（首结点）内存的值，结果是



data1 == 0xfd=253, data2 == 0x1, next == 0x0804b260。

接着观察0x0804b260（结点2）：



data1==0x2d5=725 data2==0x2 next== 0x804b254

依此类推得

第3个结点: data1=0x12d=301 next=0x0804b248

第4个结点: data1=0x3e5=997 next=0x0804b23c

第5个结点: data1=0xd4=212 next=0x0804b230

第6个结点: data1=0x1b0=432

也就是说要重新排的话 从大到小编号为，4 2 6 3 1 5

**答案**

所有关卡的答案为：

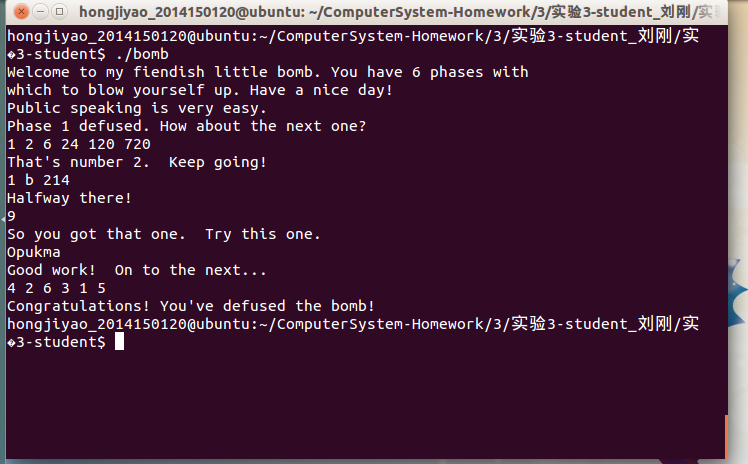
Public speaking is very easy.

1 2 6 24 120 720

0 q 777 或者 1 b 214等

Opukma 等等

4 2 6 3 1 5



**五、实验总结与体会**

**汇编真好玩。**

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  指导教师签字： 李炎然    2016年 4月21日 |
| 备注： |