**CSAPP**

**实 验 报 告**

**学生姓名** 吴语港

**学生学号** SA19225404

**实验日期** 2019/10/15

实 验 报 告

**一、实验名称：二进制炸弹**

**二、实验学时： 3**

1. **实验内容和目的：**

1.二进制炸弹包含若干个阶段，每个阶段需要输入特定的字符串，所有输入正确则炸弹被排除，否则…..

任务是找出这些字符串

字符串记录到文件中，可按下列方式验证

./bomb solution.txt

用换行区别不同阶段的字符串

实验完成后上传solution.txt的打包文件

2.从教辅系统上下载压缩包、组队表格

压缩包包含48个bomb代码包

在组队表格中找到自己的组号

根据“组号%48 + 1”领取自己的代码

本组：组号9，领取bomb10

解压后有如下文件

bomb 二进制可执行文件，任务目标文件

bomb.c bomb的源文件，辅助理解bomb代码

3.GDB参考

课本3.12（第二版3.11）节，“现实生活：使用GDB调试器”

更详细一些的资料：

http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Debug.html

官方网站

http://www.gnu.org/software/gdb/

1. **实验原理：**

1. 查看bomb.c可知程序利用phase\_\*函数（\*为1~6） 检查输入字符串是否合法，不合法就引爆炸弹。我们的任务就是逆向出每个phase的检查规则，构造出合法字符串。

当然，bomb.c没有给出phase\_\*的源码

逆向方法

Gdb

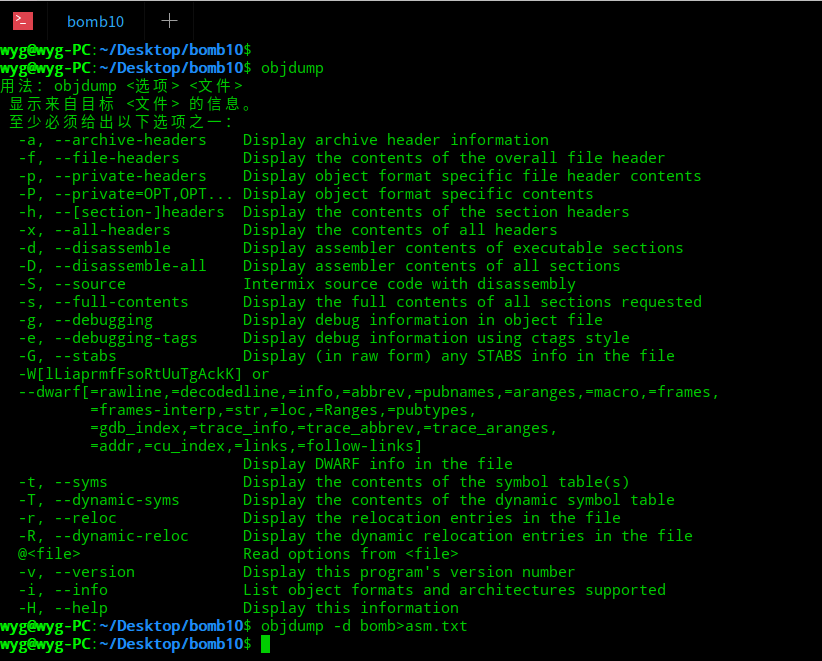
直接反汇编

**五、实验步骤及结果：**

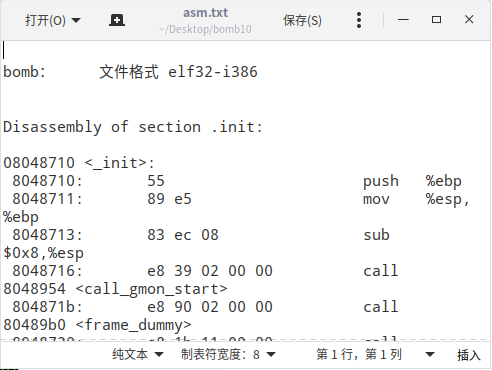
**1.准备工作：**

首先反汇编

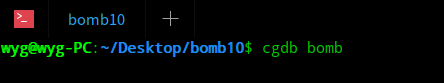
首先反汇编代码，objdump –d bomb > asm.txt，对bomb进行反汇编并将汇编代码输出到asm.txt中。



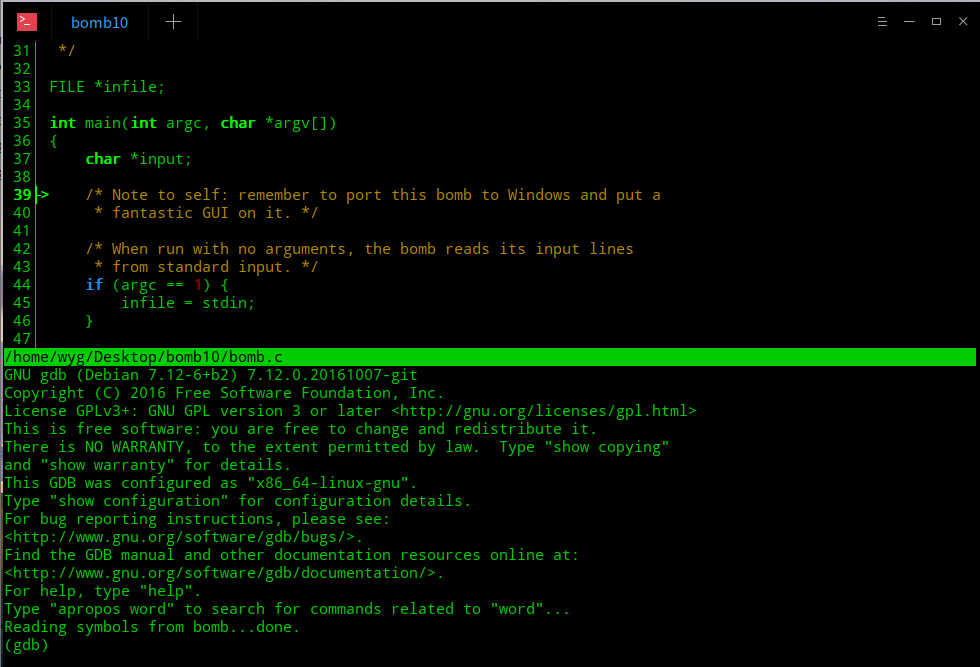
asm.txt文件已经生成,为后面的分析做准备



开始GDB调试



进入调试窗口



**2. Phase1：**

调试函数1

**汇编源码与说明：**

08048b80 <phase\_1>:

8048b80: 55 push %ebp

8048b81: 89 e5 mov %esp,%ebp

8048b83: 83 ec 08 sub $0x8,%esp

8048b86: c7 44 24 04 d8 99 04 movl $0x80499d8,0x4(%esp)

8048b8d: 08

8048b8e: 8b 45 08 mov 0x8(%ebp),%eax

8048b91: 89 04 24 mov %eax,(%esp)

8048b94: e8 7a 05 00 00 call 8049113 <strings\_not\_equal>

8048b99: 85 c0 test %eax,%eax

8048b9b: 74 05 je 8048ba2 <phase\_1+0x22>

8048b9d: e8 38 0b 00 00 call 80496da <explode\_bomb>

8048ba2: c9 leave

8048ba3: c3 ret

第一步先压栈，ebp为栈底指针，esp为栈顶指针

第二步把esp赋值给ebp

第三步把0x18赋值给esp

第四步把内存中地址为$0x80499d8的内容赋值给\*（esp+4）

第五步输入的字符\*（ebp+8）=》eax即为所求，理由如后续。

后续比较两字符串是否相等，如果相等令eax为0 否则为1(这里我是根据前后逻辑关系推测的！严谨的讲可以反汇编这个函数，具体看其中值的具体变化）

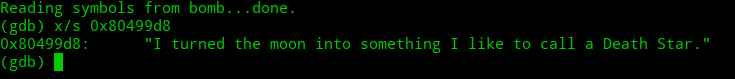
再往后，如果eax=0 则进入phase2 否则爆炸

最后为第二个炸弹做准备。返回。

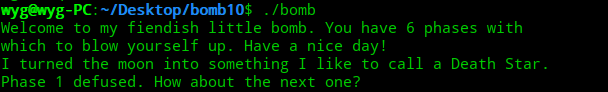
用（gdb） x/s 0x80499d8查看位于该地址处的内容，可以看到出现

**I turned the moon into something I like to call a Death Star.**

读出数据后使用RUN命令测试成功，上面字符串即为所求。



测试：



**3. Phase2：**

调试函数2

08048ba4 <phase\_2>:

8048ba4: 55 push %ebp

8048ba5: 89 e5 mov %esp,%ebp

8048ba7: 83 ec 28 sub $0x28,%esp

8048baa: c7 45 fc 00 00 00 00 movl $0x0,-0x4(%ebp)

8048bb1: 8d 45 e0 lea -0x20(%ebp),%eax

8048bb4: 89 44 24 04 mov %eax,0x4(%esp)

8048bb8: 8b 45 08 mov 0x8(%ebp),%eax

8048bbb: 89 04 24 mov %eax,(%esp)

8048bbe: e8 bd 04 00 00 call 8049080 <read\_six\_numbers>

8048bc3: c7 45 f8 00 00 00 00 movl $0x0,-0x8(%ebp)

8048bca: eb 27 jmp 8048bf3 <phase\_2+0x4f>

8048bcc: 8b 45 f8 mov -0x8(%ebp),%eax

8048bcf: 8b 54 85 e0 mov -0x20(%ebp,%eax,4),%edx

8048bd3: 8b 45 f8 mov -0x8(%ebp),%eax

8048bd6: 83 c0 03 add $0x3,%eax

8048bd9: 8b 44 85 e0 mov -0x20(%ebp,%eax,4),%eax

8048bdd: 39 c2 cmp %eax,%edx

8048bdf: 74 05 je 8048be6 <phase\_2+0x42>

8048be1: e8 f4 0a 00 00 call 80496da <explode\_bomb>

8048be6: 8b 45 f8 mov -0x8(%ebp),%eax

8048be9: 8b 44 85 e0 mov -0x20(%ebp,%eax,4),%eax

8048bed: 01 45 fc add %eax,-0x4(%ebp)

8048bf0: ff 45 f8 incl -0x8(%ebp)

8048bf3: 83 7d f8 02 cmpl $0x2,-0x8(%ebp)

8048bf7: 7e d3 jle 8048bcc <phase\_2+0x28>

8048bf9: 83 7d fc 00 cmpl $0x0,-0x4(%ebp)

8048bfd: 75 05 jne 8048c04 <phase\_2+0x60>

8048bff: e8 d6 0a 00 00 call 80496da <explode\_bomb>

8048c04: c9 leave

8048c05: c3 ret

生成的txt文件中代码如上所示。

由这句话8048bc3: c7 45 f8 00 00 00 00 movl $0x0,-0x8(%ebp)

表明-0x8(%ebp)里的值一开始是0

然后根据8048bf3: 83 7d f8 02 cmpl $0x2,-0x8(%ebp)知道循环三次

再根据以下

8048bcc: 8b 45 f8 mov -0x8(%ebp),%eax

8048bcf: 8b 54 85 e0 mov -0x20(%ebp,%eax,4),%edx

8048bd3: 8b 45 f8 mov -0x8(%ebp),%eax

8048bd6: 83 c0 03 add $0x3,%eax

8048bd9: 8b 44 85 e0 mov -0x20(%ebp,%eax,4),%eax

8048bdd: 39 c2 cmp %eax,%edx

8048bdf: 74 05 je 8048be6 <phase\_2+0x42>

这段可知每隔三个数比较数组元素是否相等，不相等则跳转炸弹爆炸，故输入6个相等的数即可通过。

测试：



**4. Phase3：**

调试函数3

**汇编源码与说明：**

08048c06 <phase\_3>:

8048c06: 55 push %ebp

8048c07: 89 e5 mov %esp,%ebp

8048c09: 83 ec 38 sub $0x38,%esp

8048c0c: c7 45 f8 00 00 00 00 movl $0x0,-0x8(%ebp)

8048c13: 8d 45 f0 lea -0x10(%ebp),%eax

8048c16: 89 44 24 10 mov %eax,0x10(%esp)

8048c1a: 8d 45 ef lea -0x11(%ebp),%eax

8048c1d: 89 44 24 0c mov %eax,0xc(%esp)

8048c21: 8d 45 f4 lea -0xc(%ebp),%eax

8048c24: 89 44 24 08 mov %eax,0x8(%esp)

8048c28: c7 44 24 04 16 9a 04 movl $0x8049a16,0x4(%esp)

8048c2f: 08

8048c30: 8b 45 08 mov 0x8(%ebp),%eax

8048c33: 89 04 24 mov %eax,(%esp)

8048c36: e8 2d fc ff ff call 8048868 <sscanf@plt>

8048c3b: 89 45 f8 mov %eax,-0x8(%ebp)

8048c3e: 83 7d f8 02 cmpl $0x2,-0x8(%ebp)

8048c42: 7f 05 jg 8048c49 <phase\_3+0x43>

8048c44: e8 91 0a 00 00 call 80496da <explode\_bomb>

8048c49: 8b 45 f4 mov -0xc(%ebp),%eax

8048c4c: 89 45 dc mov %eax,-0x24(%ebp)

8048c4f: 83 7d dc 07 cmpl $0x7,-0x24(%ebp)

8048c53: 0f 87 c0 00 00 00 ja 8048d19 <phase\_3+0x113>

8048c59: 8b 55 dc mov -0x24(%ebp),%edx

8048c5c: 8b 04 95 20 9a 04 08 mov 0x8049a20(,%edx,4),%eax

8048c63: ff e0 jmp \*%eax

8048c65: c6 45 ff 6d movb $0x6d,-0x1(%ebp)

8048c69: 8b 45 f0 mov -0x10(%ebp),%eax

8048c6c: 3d 9b 01 00 00 cmp $0x19b,%eax

8048c71: 0f 84 ab 00 00 00 je 8048d22 <phase\_3+0x11c>

8048c77: e8 5e 0a 00 00 call 80496da <explode\_bomb>

8048c7c: e9 a1 00 00 00 jmp 8048d22 <phase\_3+0x11c>

8048c81: c6 45 ff 66 movb $0x66,-0x1(%ebp)

8048c85: 8b 45 f0 mov -0x10(%ebp),%eax

8048c88: 3d 81 00 00 00 cmp $0x81,%eax

8048c8d: 0f 84 8f 00 00 00 je 8048d22 <phase\_3+0x11c>

8048c93: e8 42 0a 00 00 call 80496da <explode\_bomb>

8048c98: e9 85 00 00 00 jmp 8048d22 <phase\_3+0x11c>

8048c9d: c6 45 ff 6f movb $0x6f,-0x1(%ebp)

8048ca1: 8b 45 f0 mov -0x10(%ebp),%eax

8048ca4: 3d 24 01 00 00 cmp $0x124,%eax

8048ca9: 74 77 je 8048d22 <phase\_3+0x11c>

8048cab: e8 2a 0a 00 00 call 80496da <explode\_bomb>

8048cb0: eb 70 jmp 8048d22 <phase\_3+0x11c>

8048cb2: c6 45 ff 70 movb $0x70,-0x1(%ebp)

8048cb6: 8b 45 f0 mov -0x10(%ebp),%eax

8048cb9: 3d 6f 01 00 00 cmp $0x16f,%eax

8048cbe: 74 62 je 8048d22 <phase\_3+0x11c>

8048cc0: e8 15 0a 00 00 call 80496da <explode\_bomb>

8048cc5: eb 5b jmp 8048d22 <phase\_3+0x11c>

8048cc7: c6 45 ff 67 movb $0x67,-0x1(%ebp)

8048ccb: 8b 45 f0 mov -0x10(%ebp),%eax

8048cce: 3d 34 01 00 00 cmp $0x134,%eax

8048cd3: 74 4d je 8048d22 <phase\_3+0x11c>

8048cd5: e8 00 0a 00 00 call 80496da <explode\_bomb>

8048cda: eb 46 jmp 8048d22 <phase\_3+0x11c>

8048cdc: c6 45 ff 70 movb $0x70,-0x1(%ebp)

8048ce0: 8b 45 f0 mov -0x10(%ebp),%eax

8048ce3: 3d 0e 03 00 00 cmp $0x30e,%eax

8048ce8: 74 38 je 8048d22 <phase\_3+0x11c>

8048cea: e8 eb 09 00 00 call 80496da <explode\_bomb>

8048cef: eb 31 jmp 8048d22 <phase\_3+0x11c>

8048cf1: c6 45 ff 79 movb $0x79,-0x1(%ebp)

8048cf5: 8b 45 f0 mov -0x10(%ebp),%eax

8048cf8: 3d 14 02 00 00 cmp $0x214,%eax

8048cfd: 74 23 je 8048d22 <phase\_3+0x11c>

8048cff: e8 d6 09 00 00 call 80496da <explode\_bomb>

8048d04: eb 1c jmp 8048d22 <phase\_3+0x11c>

8048d06: c6 45 ff 77 movb $0x77,-0x1(%ebp)

8048d0a: 8b 45 f0 mov -0x10(%ebp),%eax

8048d0d: 83 f8 6e cmp $0x6e,%eax

8048d10: 74 10 je 8048d22 <phase\_3+0x11c>

8048d12: e8 c3 09 00 00 call 80496da <explode\_bomb>

8048d17: eb 09 jmp 8048d22 <phase\_3+0x11c>

8048d19: c6 45 ff 62 movb $0x62,-0x1(%ebp)

8048d1d: e8 b8 09 00 00 call 80496da <explode\_bomb>

8048d22: 0f b6 45 ef movzbl -0x11(%ebp),%eax

8048d26: 38 45 ff cmp %al,-0x1(%ebp)

8048d29: 74 05 je 8048d30 <phase\_3+0x12a>

8048d2b: e8 aa 09 00 00 call 80496da <explode\_bomb>

8048d30: c9 leave

8048d31: c3 ret

总体汇编代码如上所示，具体分析如下：

本题代码整体太长，一点一点解读既费时间，又可能找不到头绪。所幸我们找到了里面一直出现的cmp je语句，这提示我们一个重要信息，即附近有个switch循环，仔细寻找后如下

8048c13: 8d 45 f0 lea -0x10(%ebp),%eax

8048c16: 89 44 24 10 mov %eax,0x10(%esp)

8048c1a: 8d 45 ef lea -0x11(%ebp),%eax

8048c1d: 89 44 24 0c mov %eax,0xc(%esp)

8048c21: 8d 45 f4 lea -0xc(%ebp),%eax

8048c24: 89 44 24 08 mov %eax,0x8(%esp)

分析上面几个lea和mov语句可知，该操作获取了三个参数，依次是int，char，int

再根据如下

8048c49: 8b 45 f4 mov -0xc(%ebp),%eax

8048c4c: 89 45 dc mov %eax,-0x24(%ebp)

8048c4f: 83 7d dc 07 cmpl $0x7,-0x24(%ebp)

由上三行可知，比较第一个参数在不在0到7之间，然后后面根据第一个输入的参数进行跳转。我们只要破解一种输入就行了。选取第一个输入为0 o 841

5. Phase4：

调试函数4

**汇编源码与说明：**

08048d72 <phase\_4>:

8048d72: 55 push %ebp

8048d73: 89 e5 mov %esp,%ebp

8048d75: 83 ec 28 sub $0x28,%esp

8048d78: 8d 45 f4 lea -0xc(%ebp),%eax

8048d7b: 89 44 24 08 mov %eax,0x8(%esp)

8048d7f: c7 44 24 04 40 9a 04 movl $0x8049a40,0x4(%esp)

8048d86: 08

8048d87: 8b 45 08 mov 0x8(%ebp),%eax

8048d8a: 89 04 24 mov %eax,(%esp)

8048d8d: e8 d6 fa ff ff call 8048868 <sscanf@plt>

8048d92: 89 45 fc mov %eax,-0x4(%ebp)

8048d95: 83 7d fc 01 cmpl $0x1,-0x4(%ebp)

8048d99: 75 07 jne 8048da2 <phase\_4+0x30>

8048d9b: 8b 45 f4 mov -0xc(%ebp),%eax

8048d9e: 85 c0 test %eax,%eax

8048da0: 7f 05 jg 8048da7 <phase\_4+0x35>

8048da2: e8 33 09 00 00 call 80496da <explode\_bomb>

8048da7: 8b 45 f4 mov -0xc(%ebp),%eax

8048daa: 89 04 24 mov %eax,(%esp)

8048dad: e8 80 ff ff ff call 8048d32 <func4>

8048db2: 89 45 f8 mov %eax,-0x8(%ebp)

8048db5: 81 7d f8 2f 45 00 00 cmpl $0x452f,-0x8(%ebp)

8048dbc: 74 05 je 8048dc3 <phase\_4+0x51>

8048dbe: e8 17 09 00 00 call 80496da <explode\_bomb>

8048dc3: c9 leave

8048dc4: c3 ret

由以下代码8048d7f: c7 44 24 04 74 9a 04 movl $0x8049a74,0x4(%esp)可看出此时刻获取了某个参数，打个断点看一下得

p (char\*) 0x8049a74

显示如下

$1 = 0x8049a74 "%d"说明此刻输入一个整数

再由

8048db5: 81 7d f8 2f 45 00 00 cmpl $0x452f,-0x8(%ebp)

8048dbc: 74 05 je 8048dc3 <phase\_4+0x51>

8048dbe: e8 17 09 00 00 call 80496da <explode\_bomb>

8048dc3: c9 leave

得即从斐波那契函数等于0x452f时是第几个数，即为输入。

F(x) = F(x-1)+F(x-2), F(1)=1,F(2)=1,F(3)=2......F(12)=17711即0x452f，所以我们输入的数是12

**6. Phase5：**

调试函数5

08048dc5 <phase\_5>:

8048dc5: 55 push %ebp ///压栈 ebp为栈指针 esp为栈指针

8048dc6: 89 e5 mov %esp,%ebp //把esp赋值给ebp

8048dc8: 83 ec 18 sub $0x18,%esp //0x18赋值给esp

8048dcb: 8b 45 08 mov 0x8(%ebp),%eax //\*（esp+8）=》eax 就是输入的字符串

8048dce: 89 04 24 mov %eax,(%esp) //eax的值赋值给\*esp

8048dd1:e8 13 03 00 00 call 80490e9 <string\_length>

8048dd6:89 45 fc mov %eax,-0x4(%ebp)

8048dd9:83 7d fc 06 cmpl $0x6,-0x4(%ebp)//输入字符串长6

8048ddd:74 05 je 8048de4 <phase\_5+0x1f>

8048ddf: e8 f6 08 00 00 call 80496da <explode\_bomb>

8048de4: c7 45 f8 00 00 00 00 movl $0x0,-0x8(%ebp)

8048deb:eb 20 jmp 8048e0d <phase\_5+0x48>

8048ded:8b 55 f8 mov -0x8(%ebp),%edx

8048df0: 8b 45 f8 mov -0x8(%ebp),%eax

8048df3: 03 45 08 add 0x8(%ebp),%eax

8048df6: 0f b6 00 movzbl (%eax),%eax

8048df9: 0f be c0 movsbl %al,%eax

8048dfc: 83 e0 0f and $0xf,%eax//取后4位

8048dff: 0f b6 80 c0 a5 04 08 movzbl 0x804a5c0(%eax),%eax

8048e06: 88 44 15 f1 mov %al,-0xf(%ebp,%edx,1)

8048e0a: ff 45 f8 incl -0x8(%ebp)

8048e0d: 83 7d f8 05 cmpl $0x5,-0x8(%ebp)

8048e11: 7e da jle 8048ded <phase\_5+0x28>

8048e13: c6 45 f7 00 movb $0x0,-0x9(%ebp)

8048e17: c7 44 24 04 43 9a 04 movl $0x8049a43,0x4(%esp)

8048e1e: 08

8048e1f: 8d 45 f1 lea -0xf(%ebp),%eax

8048e22: 89 04 24 mov %eax,(%esp)

8048e25: e8 e9 02 00 00 call 8049113 <strings\_not\_equal>

8048e2a: 85 c0 test %eax,%eax

8048e2c: 74 05 je 8048e33 <phase\_5+0x6e>//如果eax=0 则进入phase6 否则爆炸

8048e2e: e8 a7 08 00 00 call 80496da <explode\_bomb>

8048e33: c9 leave //为结束函数做准备

8048e34: c3 ret //返回

1.调用string\_length测长度3，若不为6则爆炸

8048dd1:e8 13 03 00 00 call 80490e9 <string\_length>

8048dd6:89 45 fc mov %eax,-0x4(%ebp)

8048dd9:83 7d fc 06 cmpl $0x6,-0x4(%ebp)//输入字符串长6

8048ddd:74 05 je 8048de4 <phase\_5+0x1f>

8048ddf: e8 f6 08 00 00 call 80496da <explode\_bomb>

2.循环6次，取出字符，截取低四位

8048df6: 0f b6 00 movzbl (%eax),%eax

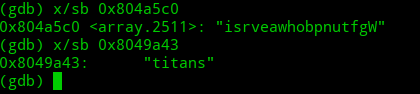
8048df9: 0f be c0 movsbl %al,%eax

8048dfc: 83 e0 0f and $0xf,%eax//取后4位

8048dff: 0f b6 80 c0 a5 04 08 movzbl 0x804a5c0(%eax),%eax

8048e06: 88 44 15 f1 mov %al,-0xf(%ebp,%edx,1)

3.根据输入字符ASCII码低四位与索引比较，不同则爆炸。



我们得到的作为参考用，也就是索引表功能的字符串为isrveawhobpnutfg，而我们的目的字符串为titans,分别为13位，0位，13位，5位，11位和1位，所以只要我们输入的六个字符的相应的低4位的二进制表示为13,0,13,5,11,1

即可。所以这一关我选取的密码是mOm5k1

**7. Phase6：**

调试函数6(注释)

08048e35 <fun6>:

8048e35: 55 push %ebp

8048e36: 89 e5 mov %esp,%ebp

8048e38: 83 ec 10 sub $0x10,%esp # esp-0x10

8048e3b: 8b 45 08 mov 0x8(%ebp),%eax

# eax=arg=local2=0x804a66c

8048e3e: 89 45 f0 mov %eax,-0x10(%ebp) # local4=arg

8048e41: 8b 45 08 mov 0x8(%ebp),%eax

8048e44: 89 45 f0 mov %eax,-0x10(%ebp)

8048e47: 8b 45 08 mov 0x8(%ebp),%eax # eax=arg

# eax=\*(arg+8)=\*(0x804a66c)

8048e4a: 8b 40 08 mov 0x8(%eax),%eax

# local3=eax=\*(0x804a66c)

8048e4d: 89 45 f4 mov %eax,-0xc(%ebp)

# eax=local4=0x804a66c

8048e50: 8b 45 f0 mov -0x10(%ebp),%eax

# \*(0x804a66c)=0

8048e53: c7 40 08 00 00 00 00 movl $0x0,0x8(%eax)

8048e5a: eb 62 jmp 8048ebe <fun6+0x89>

#loop

8048e5c: 8b 45 f0 mov -0x10(%ebp),%eax # eax=local4

# local1=local4

8048e5f: 89 45 fc mov %eax,-0x4(%ebp)

8048e62: 8b 45 f0 mov -0x10(%ebp),%eax

# local2=local4

8048e65: 89 45 f8 mov %eax,-0x8(%ebp)

8048e68: eb 0f jmp 8048e79 <fun6+0x44>

8048e6a: 8b 45 fc mov -0x4(%ebp),%eax

8048e6d: 89 45 f8 mov %eax,-0x8(%ebp)

8048e70: 8b 45 fc mov -0x4(%ebp),%eax

8048e73: 8b 40 08 mov 0x8(%eax),%eax

8048e76: 89 45 fc mov %eax,-0x4(%ebp)

8048e79: 83 7d fc 00 cmpl $0x0,-0x4(%ebp)

# if(local1==0)

8048e7d: 74 0e je 8048e8d <fun6+0x58>

8048e7f: 8b 45 fc mov -0x4(%ebp),%eax

8048e82: 8b 10 mov (%eax),%edx

8048e84: 8b 45 f4 mov -0xc(%ebp),%eax

8048e87: 8b 00 mov (%eax),%eax

8048e89: 39 c2 cmp %eax,%edx

8048e8b: 7f dd jg 8048e6a <fun6+0x35>

8048e8d: 8b 45 f8 mov -0x8(%ebp),%eax

8048e90: 3b 45 fc cmp -0x4(%ebp),%eax

8048e93: 74 0b je 8048ea0 <fun6+0x6b>

8048e95: 8b 55 f8 mov -0x8(%ebp),%edx

8048e98: 8b 45 f4 mov -0xc(%ebp),%eax

8048e9b: 89 42 08 mov %eax,0x8(%edx)

8048e9e: eb 06 jmp 8048ea6 <fun6+0x71>

8048ea0: 8b 45 f4 mov -0xc(%ebp),%eax

8048ea3: 89 45 f0 mov %eax,-0x10(%ebp)

8048ea6: 8b 45 f4 mov -0xc(%ebp),%eax

8048ea9: 8b 40 08 mov 0x8(%eax),%eax

8048eac: 89 45 f8 mov %eax,-0x8(%ebp)

8048eaf: 8b 55 f4 mov -0xc(%ebp),%edx

8048eb2: 8b 45 fc mov -0x4(%ebp),%eax

8048eb5: 89 42 08 mov %eax,0x8(%edx)

8048eb8: 8b 45 f8 mov -0x8(%ebp),%eax

8048ebb: 89 45 f4 mov %eax,-0xc(%ebp)

8048ebe: 83 7d f4 00 cmpl $0x0,-0xc(%ebp) #是否为0

#if(local3!=0) jump 39;

8048ec2: 75 98 jne 8048e5c <fun6+0x27>

8048ec4: 8b 45 f0 mov -0x10(%ebp),%eax

8048ec7: c9 leave

8048ec8: c3 ret

08048ec9 <phase\_6>:

8048ec9: 55 push %ebp

8048eca: 89 e5 mov %esp,%ebp

8048ecc: 83 ec 18 sub $0x18,%esp

8048ecf: c7 45 f8 3c a6 04 08 movl $0x804a63c,-0x8(%ebp)

8048ed6: 8b 45 08 mov 0x8(%ebp),%eax

8048ed9: 89 04 24 mov %eax,(%esp)

8048edc: e8 77 f9 ff ff call 8048858 <atoi@plt>

# atoi (表示 ascii to integer)是把字符串转换成整型数的一个函数

8048ee1: 89 c2 mov %eax,%edx

8048ee3: 8b 45 f8 mov -0x8(%ebp),%eax

8048ee6: 89 10 mov %edx,(%eax)

8048ee8: 8b 45 f8 mov -0x8(%ebp),%eax

8048eeb: 89 04 24 mov %eax,(%esp)

8048eee: e8 42 ff ff ff call 8048e35 <fun6>

8048ef3: 89 45 f8 mov %eax,-0x8(%ebp)

8048ef6: 8b 45 f8 mov -0x8(%ebp),%eax

8048ef9: 89 45 fc mov %eax,-0x4(%ebp)

8048efc: c7 45 f4 01 00 00 00 movl $0x1,-0xc(%ebp)

8048f03: eb 0c jmp 8048f11 <phase\_6+0x48>

8048f05: 8b 45 fc mov -0x4(%ebp),%eax

8048f08: 8b 40 08 mov 0x8(%eax),%eax

8048f0b: 89 45 fc mov %eax,-0x4(%ebp)

8048f0e: ff 45 f4 incl -0xc(%ebp)

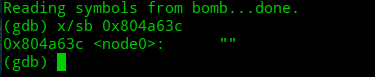
8048f11: 83 7d f4 08 cmpl $0x8,-0xc(%ebp) #是否为8

8048f15: 7e ee jle 8048f05 <phase\_6+0x3c>

8048f17: 8b 45 fc mov -0x4(%ebp),%eax

8048f1a: 8b 10 mov (%eax),%edx

8048f1c: a1 3c a6 04 08 mov 0x804a63c,%eax



#在这设一个断点

8048f21: 39 c2 cmp %eax,%edx

8048f23: 74 05 je 8048f2a <phase\_6+0x61>

8048f25: e8 b0 07 00 00 call 80496da <explode\_bomb>

8048f2a: c9 leave

8048f2b: c3 ret

答案：345

解题：看到phase\_6的第88行是和输入的数做比较，只要在这里设置断点直接查看edx中的值即可



0x159=345

**8.实验总结：**

一开始接触这个实验的时候真的是像刘姥姥进了大观园，什么都不懂或者说是什么都不敢动。查阅一些料后并学习了一些gdb 的使用方法后， 开始着手研究反编译代码，并一步一步她开始拆除炸弹。本次实验后， 对涌过gdb 调试工具来调试程序有了一些了解。整个二进制炸弹的程序分别用到了直接她址寻址、循环语句、跳转、递归函数、字符串switch码转换和链表等知识，涉及内容多覆盖面广，破解炸弹的过程是一次重要的巩固和提高的过程。