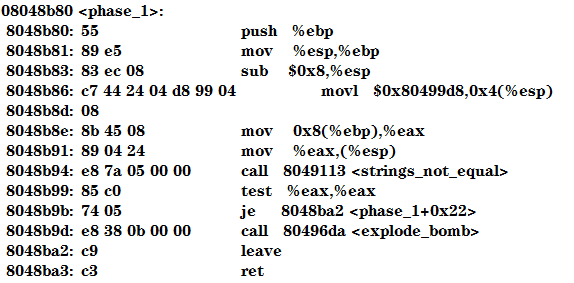
# LAB2

首先查看整个bomb.c的代码，发现整个炸弹组是由6个小炸弹（函数）组成的。整个main函数比较简单，函数间变量几乎没有影响。因此，只需要依次解除6个小炸弹即可。

所以，接下来便开始依次调试各函数。

**调试函数1:**



**汇编源码与说明：**

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。

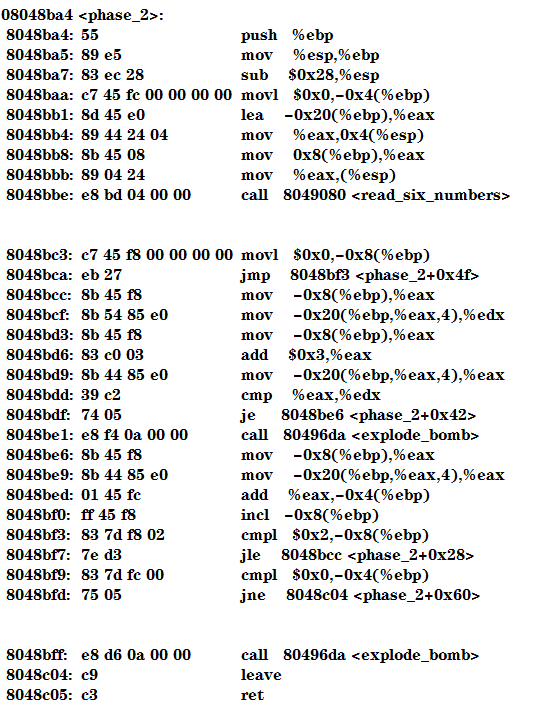
用（gdb） x/s 0x80499d8查看位于该地址处的内容，可以看到出现

**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。

**调试函数2:**

**汇编源码与说明：**



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。

**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

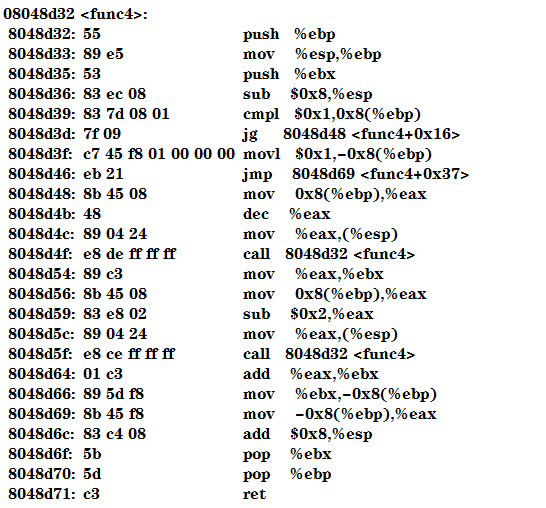
8048c4c: 89 45 dc mov %eax,-0x24(%ebp)

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

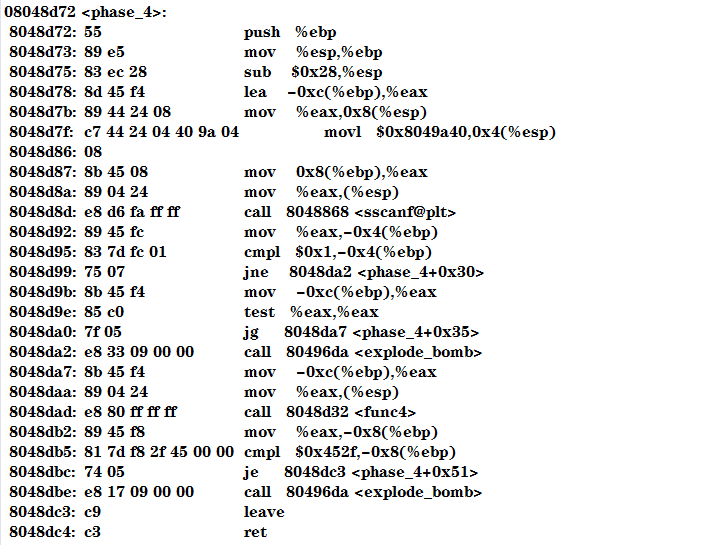
由上三行可知，比较第一个参数在不在0到7之间，然后后面根据第一个输入的参数进行跳转。我们只要破解一种输入就行了。选取第一个输入为0 o 841

**调试函数4:**

**汇编源码与说明：**



如上Fun4函数分析可得是计算斐波那契数列



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