湖南大學

HUNAN UNIVERSITY

课程实验报告

课　程 名 称：　深入理解计算机系统

实验项目名称：　bomb 实验

专　业 班 级：　计科 1908

姓　　名：　周永浩

学　　号：　201908010808

指 导 教　师：　赵欢、黄丽达

完 成 时 间：　2021 年　5　月　12　日

信息科学与工程学院

实验题目：bomb 实验

实验目的：

熟悉汇编程序以及相关的调试方法，使用 gdb 工具反汇编出代码，并进行代码分析

实验环境：虚拟机、bomb 程序包、实验课程相关资料

实验内容及操作步骤：

1：将 bomb 程序包解压后放入到虚拟机中

2：打开命令程序框，输入 objdump -d bomb>1.txt，将 bomb 函数反汇编代码放入到 1.txt 文件中，对 1.txt 文件反汇编代码进行分析

Phase1：

```
08048b50 <phase_1>:
 8048b50:   83 ec 1c                sub    $0x1c,%esp        #开辟一段空间
 8048b53:   c7 44 24 04 ac a1 04    movl   $0x804a1ac,0x4(%esp) #将0x804a1ac放入到esp加4的空
间内
 8048b5a:   08
 8048b5b:   8b 44 24 20             mov    0x20(%esp),%eax   #0x20(%esp)存放值给eax
 8048b5f:   89 04 24                mov    %eax,(%esp)   #把eax值存放到0x20(%esp)
 8048b62:   e8 7d 04 00 00          call   8048fe4 <strings_not_equal>   #调用函数
 8048b67:   85 c0                   test   %eax,%eax     #函数返回值相与改变操作码
 8048b69:   74 05                   je     8048b70 <phase_1+0x20>   #相与结果为0跳转到
8048b70
 8048b6b:   e8 86 05 00 00          call   80490f6 <explode_bomb>    #相与结果不为0引起爆炸
 8048b70:   83 c4 1c                add    $0x1c,%esp
 8048b73:   c3                      ret
```

0x8048b67：将 string_not_equal 函数返回值 eax 与自身相与，改变操作码；

如果函数返回结果为 0，跳转到 explode_bomb 函数，爆炸；否则不产生爆炸

分析 string_not_equal 函数，即匹配两个字符串是否相等，函数的两个参数为两个字符串的首地址，在 0x8048b53 指令中存在立即数，使用 gdb 查看 0x804a1ac 中字符串，即为在函数中进行对比的字符串，对比字符串结果为 0，不会引起爆炸。

```
(gdb) x/s 0x804a1ac
0x804a1ac:      "When I get angry, Mr. Bigglesworth gets upset."
```

在函数<explode_bomb>函数之前设置断点，找到函数的入口地址 0x80490f6，在这个位置设置断点，输入上步骤得到的字符即可通关

```
(gdb) x/s 0x804a1ac
0x804a1ac:      "When I get angry, Mr. Bigglesworth gets upset."
(gdb) break *0x80490f6
Breakpoint 1 at 0x80490f6
(gdb) r
Starting program: /home/zhou828/Desktop/bomb16/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
When I get angry, Mr. Bigglesworth gets upset.
Phase 1 defused. How about the next one?
```

Phase2：

```
08048b74 <phase_2>:
 8048b74:   56                      push   %esi
 8048b75:   53                      push   %ebx
 8048b76:   83 ec 34                sub    $0x34,%esp    #开辟内存空间
 8048b79:   8d 44 24 18             lea    0x18(%esp),%eax  #将0x18(%esp)中存放值放到eax中
 8048b7d:   89 44 24 04             mov    %eax,0x4(%esp)   #将eax中值存放到0x4(%esp)中，函数
参数准备
 8048b81:   8b 44 24 40             mov    0x40(%esp),%eax  #将0x40(%esp)中存放值放到eax中
 8048b85:   89 04 24                mov    %eax,(%esp)  #将eax中值存放到(%esp)中,函数参数准备
 8048b88:   e8 9e 06 00 00          call   804922b <read_six_numbers>   #调用
<read_six_numbers>函数
 8048b8d:   83 7c 24 18 00          cmpl   $0x0,0x18(%esp)  #0x18(%esp)-0改变标志位
 8048b92:   75 07                   jne    8048b9b <phase_2+0x27>   #0x18(%esp)!=0,bomb
 8048b94:   83 7c 24 1c 01          cmpl   $0x1,0x1c(%esp)  #0x1c(%esp)-1改变标志位
 8048b99:   74 05                   je     8048ba0 <phase_2+0x2c>   #0x1c(%esp)!=1,bomb
 8048b9b:   e8 56 05 00 00          call   80490f6 <explode_bomb>
 8048ba0:   8d 5c 24 20             lea    0x20(%esp),%ebx  #ebx存放a[2]地址
 8048ba4:   8d 74 24 30             lea    0x30(%esp),%esi  #esi存放a[6]地址
 8048ba8:   8b 43 f8                mov    -0x8(%ebx),%eax  #eax=a[i-2]
 8048bab:   03 43 fc                add    -0x4(%ebx),%eax  #eax=a[i-1]+a[i-2]
 8048bae:   39 03                   cmp    %eax,(%ebx)  #比较(%ebx)与eax
 8048bb0:   74 05                   je     8048bb7 <phase_2+0x43>   #(%ebx)!=eax,bomb
 8048bb2:   e8 3f 05 00 00          call   80490f6 <explode_bomb>
 8048bb7:   83 c3 04                add    $0x4,%ebx    #ebx+4，找到a[i+1]
 8048bba:   39 f3                   cmp    %esi,%ebx    #ebx-esi
 8048bbc:   75 ea                   jne    8048ba8 <phase_2+0x34>   #ebx!=esi,持续循环
 8048bbe:   83 c4 34                add    $0x34,%esp   #释放空间
 8048bc1:   5b                      pop    %ebx     #弹出ebx
 8048bc2:   5e                      pop    %esi     #弹出esi
 8048bc3:   c3                      ret
```

调用 read_six_number 函数可知，程序要求输入六个数，其中第一个和第二个数为 0 和 1，不符合就会 bomb

对这六个数进行 a[i]==a[i-1]+a[i-2]的比较，不符合就会 bomb

进行 ebx 进行循环判断，直到 a[5]比较完成

相当于输入 a[0]=0，a[1]=1 的斐波那契数，直到输入 6 个数字包括 a[0]和 a[1]

```
0 1 1 2 3 5
That's number 2.  Keep going!
```

Phase3：

```
08048bc4 <phase_3>:
 8048bc4:   83 ec 2c                sub    $0x2c,%esp
 8048bc7:   8d 44 24 1c             lea    0x1c(%esp),%eax
 8048bcb:   89 44 24 0c             mov    %eax,0xc(%esp)   #0x1c(%esp)->0xc(%esp)

 8048bcf:   8d 44 24 18             lea    0x18(%esp),%eax
 8048bd3:   89 44 24 08             mov    %eax,0x8(%esp)   #0x18(%esp)->0x8(%esp)

 8048bd7:   c7 44 24 04 9f a3 04    movl   $0x804a39f,0x4(%esp) #输入整数地址->0x4(%esp)
 8048bde:   08
 8048bdf:   8b 44 24 30             mov    0x30(%esp),%eax
 8048be3:   89 04 24                mov    %eax,(%esp)      #0x30(%esp)中的值->(%esp)

 8048be6:   e8 85 fc ff ff          call   8048870 <__isoc99_sscanf@plt> #调用函数

 8048beb:   83 f8 01                cmp    $0x1,%eax    #函数返回值与1比较
 8048bee:   7f 05                   jg     8048bf5 <phase_3+0x31>
 8048bf0:   e8 01 05 00 00          call   80490f6 <explode_bomb>   #函数返回值<=1,bomb

 8048bf5:   83 7c 24 18 07          cmpl   $0x7,0x18(%esp)
   #cmp指令*0x18(%esp)>7,bomb(第一个整数的限制条件)
 8048bfa:   77 3c                   ja     8048c38 <phase_3+0x74>#函数指向bomb

 8048bfc:   8b 44 24 18             mov    0x18(%esp),%eax  #*0x18(%esp)->eax
 8048c00:   ff 24 85 0c a2 04 08    jmp    *0x804a20c(,%eax,4)#无条件跳转指令，跳转表

 8048c07:   b8 97 03 00 00          mov    $0x397,%eax  #eax=0x397
 8048c0c:   eb 3b                   jmp    8048c49 <phase_3+0x85>   #跳转0x8048c49
 8048c0e:   b8 b0 00 00 00          mov    $0xb0,%eax
```

```
8048c44:   b8 e4 02 00 00          mov    $0x2e4,%eax
8048c49:   3b 44 24 1c             cmp    0x1c(%esp),%eax  #eax!=*0x1c(%esp),bomb
8048c4d:   74 05                   je     8048c54 <phase_3+0x90>
8048c4f:   e8 a2 04 00 00          call   80490f6 <explode_bomb>
8048c54:   83 c4 2c                add    $0x2c,%esp   #*(%esp)+0x2c
8048c57:   c3                      ret
```

调用<__isoc99_sscanf@plt>函数，之前传了四个参数，其中 0x1c(%esp)、0x18(%esp)为输入的两个整数的地址；0x30(%esp)为为存放输入字符串的首地址；0x804a39f 存放字符串首地址

```
(gdb) x/s 0x804a39f
0x804a39f:      "%d %d"
```

，这证明要输入两个整数。

```
8048bf5:   83 7c 24 18 07          cmpl   $0x7,0x18(%esp)
   #cmp指令*0x18(%esp)>7,bomb(第一个整数的限制条件)
8048bfa:   77 3c                   ja     8048c38 <phase_3+0x74>#函数指向bomb
```

第一个整数输入的限制条件，限制第一个整数要小于等于 7

```
8048c00:   ff 24 85 0c a2 04 08    jmp    *0x804a20c(,%eax,4)#无条件跳转指令，跳转表
```

这句指令是 switch 选择分支语句，进行跳转表地址选择

对跳转进行跳转测试，输入 1，跳转到指令为 0x8048c44

```
(gdb) p/x *(0x804a20c+4*1)
$1 = 0x8048c44
```

在 0x8048c44 指令中，根据汇编得到比较的数为 740（十进制），输入 1 740 就能解开关卡

```
1 740
Halfway there!
```

Phase_4:

首先根据 fun4 函数的反汇编得到 c 代码如下:

```
int fun4(int x,int y,int z)
{
    int m1=(z-y)>>31;
    int m=(m1+z-y)>>1;
    int n=m+y;
    if(n<=x)
    {
        if(n==x)return 0;
        else
            return fun4(x,n+1,z)*2+1;
    }
    else
        return 2*fun4(x,y,n-1)
}
```

```
int main()
{
    int i=0;
    for(i=0;i<=14;i++)
    {
        if(2==fun4(i,0,14))
        cout<<i<<endl;
    }
    return 0;
}
```

```
4 2
So you got that one.  Try this one.
```

在第一个方框内：*0x1c(%esp)为输入的第二个值，*0x18(%esp)为输入的第一个值

在第二个方框内：*0x18(%esp)<=e&&*0x18(%esp)>=0，不会爆炸，限定第一个值的范围为[0,e]

在第三个方框内：调用 fun4 函数前参数准备，并将返回值传给 eax

在第四个方框内：限定了 fun4 函数返回值为 2，并且*0x1c(%esp)==2，如果不符合就会 bomb

找第一个键入值 x：根据两个参数和返回值必须等于 2，0<=x<=14 这三个条件，找第一个键入值 x

参数：int fun4 (int x , int y , int z)=int fun4 (int x , 0 , 14)

结合上面 fun4c++代码，在 main（）函数中编写程序，得到键入值 x=4 或者 5

输入 5 2 或者 4 2 即可即可通过

```
08048cc5 <phase_4>:
 8048cc5:    83 ec 2c                sub    $0x2c,%esp    #分配栈空间

 8048cc8:    8d 44 24 1c             lea    0x1c(%esp),%eax
 8048ccc:    89 44 24 0c             mov    %eax,0xc(%esp)    #0x1c(%esp)->0xc(%esp)（输入的
第二个值，用于判断）

 8048cd0:    8d 44 24 18             lea    0x18(%esp),%eax
 8048cd4:    89 44 24 08             mov    %eax,0x8(%esp)    #0x18(%esp)->0x8(%esp)（输入的
第一个值）

 8048cd8:    c7 44 24 04 9f a3 04    movl   $0x804a39f,0x4(%esp) #"%d %d"
 8048cdf:    08

 8048ce0:    8b 44 24 30             mov    0x30(%esp),%eax
 8048ce4:    89 04 24                mov    %eax,(%esp)    #0x30(%esp)中的值->(%esp)

 8048ce7:    e8 84 fb ff ff          call   8048870 <__isoc99_sscanf@plt> #调用函数

 8048cec:    83 f8 02                cmp    $0x2,%eax    #eax函数返回值与0x2进行比较
 8048cef:    75 0d                   jne    8048cfe <phase_4+0x39>
#eax!=0x2,跳转到0x8048cfe指令,bomb

#eax==0x2
 8048cf1:    8b 44 24 18             mov    0x18(%esp),%eax  #*0x18(%esp)->eax
 8048cf5:    85 c0                   test   %eax,%eax    #eax&&eax,检测符号位
 8048cf7:    78 05                   js     8048cfe <phase_4+0x39>    #*0x18(%esp)<0,bomb

#*0x18(%esp)>=0&&*0x18(%esp)<=e
 8048cf9:    83 f8 0e                cmp    $0xe,%eax    #*0x18(%esp)>e,bomb

#eax<=e,跳转到0x8048d03指令
 8048cfc:    7e 05                   jle    8048d03 <phase_4+0x3e>

#8048cfe指令：调用bomb函数，bomb
 8048cfe:    e8 f3 03 00 00          call   80490f6 <explode_bomb>


 8048d03:    c7 44 24 08 0e 00 00    movl   $0xe,0x8(%esp)    #e->0x8(%esp)
 8048d0a:    00
 8048d0b:    c7 44 24 04 00 00 00    movl   $0x0,0x4(%esp)    #0->0x4(%esp)
 8048d12:    00
 8048d13:    8b 44 24 18             mov    0x18(%esp),%eax
 8048d17:    89 04 24                mov    %eax,(%esp)       #*0x18(%esp)->(%esp)

 8048d1a:    e8 39 ff ff ff          call   8048c58 <func4>   #调用函数fun4,返回参数到eax

 8048d1f:    83 f8 02                cmp    $0x2,%eax
 8048d22:    75 07                   jne    8048d2b <phase_4+0x66>#eax!=2,bomb

 8048d24:    83 7c 24 1c 02          cmpl   $0x2,0x1c(%esp)
 8048d29:    74 05                   je     8048d30 <phase_4+0x6b>#eax==2&&*0x1c(%esp)==2,
跳转到0x8048d30指令(栈空间释放)

 8048d2b:    e8 c6 03 00 00          call   80490f6 <explode_bomb>#eax!=2或者*0x1c(%esp)!=2
 8048d30:    83 c4 2c                add    $0x2c,%esp    #栈空间释放

8048d33:     c3                      ret
```

Phase_5:

```
08048d34 <phase_5>:
 8048d34:   53                      push   %ebx      #保存ebx值
 8048d35:   83 ec 28                sub    $0x28,%esp    #分配栈空间
 8048d38:   8b 5c 24 30             mov    0x30(%esp),%ebx
 8048d3c:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
 8048d42:   89 44 24 1c             mov    %eax,0x1c(%esp)  #使用金丝雀值，判断栈堆是否溢出
 8048d46:   31 c0                   xor    %eax,%eax        #异或

 8048d48:   89 1c 24                mov    %ebx,(%esp)      #(%esp)是字符串首地址
 8048d4b:   e8 7b 02 00 00          call   8048fcb <string_length>#调用字符串长度函数,长度l
 8048d50:   83 f8 06                cmp    $0x6,%eax        #函数返回值eax与6比较
 8048d53:   74 05                   je     8048d5a <phase_5+0x26>    #l==6,跳转0x8048d5a
 8048d55:   e8 9c 03 00 00          call   80490f6 <explode_bomb>    #l!=6,bomb

 8048d5a:   b8 00 00 00 00          mov    $0x0,%eax    #eax值=0

 8048d5f:   0f be 14 03             movsbl (%ebx,%eax,1),%edx
 8048d63:   83 e2 0f                and    $0xf,%edx    #取每个字符ASCII码的低四位
 8048d66:   0f b6 92 2c a2 04 08    movzbl 0x804a22c(%edx),%edx
#其中0x804a22c是字符串首地址，加上偏移量，取出字符串中的字符给edx
 8048d6d:   88 54 04 15             mov    %dl,0x15(%esp,%eax,1)
 8048d71:   83 c0 01                add    $0x1,%eax    #eax中值+1
 8048d74:   83 f8 06                cmp    $0x6,%eax    #eax中值与6进行比较
 8048d77:   75 e6                   jne    8048d5f <phase_5+0x2b>    #经过6次循环

 8048d79:   c6 44 24 1b 00          movb   $0x0,0x1b(%esp)
 8048d7e:   c7 44 24 04 02 a2 04    movl   $0x804a202,0x4(%esp)#保存有待比较的字符串
 8048d85:   08
 8048d86:   8d 44 24 15             lea    0x15(%esp),%eax
 8048d8a:   89 04 24                mov    %eax,(%esp)  #<strings_not_equal>函数参数准备
 8048d8d:   e8 52 02 00 00          call   8048fe4 <strings_not_equal>

 8048d92:   85 c0                   test   %eax,%eax    #函数返回值eax，相与进行比较
 8048d94:   74 05                   je     8048d9b <phase_5+0x67>#函数返回值等于0
 8048d96:   e8 5b 03 00 00          call   80490f6 <explode_bomb>#函数返回值！=0,bomb

 8048d9b:   8b 44 24 1c             mov    0x1c(%esp),%eax
 8048d9f:   65 33 05 14 00 00 00    xor    %gs:0x14,%eax #释放金丝雀值
 8048da6:   74 05                   je     8048dad <phase_5+0x79>
 8048da8:   e8 23 fa ff ff          call   80487d0 <__stack_chk_fail@plt> #栈攻击检测
 8048dad:   83 c4 28                add    $0x28,%esp#释放栈空间
 8048db0:   5b                      pop    %ebx #释放ebx值
 8048db1:   c3                      ret
```

```
                            je     8048d5a <phase_5+0x26>    #l==6,跳转0x8048d5a
```

在调用函数<string_length>和`call   80490f6 <explode_bomb>   #l!=6,bomb`,可知要输入一个长度为 6 的字符串，进行分析后可知输入值的最低四位要作为偏移量取出字符串中字符`movzbl 0x804a22c(%edx),%edx`

使用 gdb x/s 查看 0x804a22c 字符串

```
(gdb) x/s 0x804a22c
0x804a22c <array.2956>:  "maduiersnfotvbylSo you think you can stop the bomb wit
h ctrl-c, do you?"
```

由程序可知，要进行比对六个字符串，待比较字符串的地址为 0x804a202,使用 gdb x/s 查看 0x804a202 中字符串

```
(gdb) x/s 0x804a202
0x804a202:      "oilers"
```

o 在字符串中位置为 A，偏移量为 A，键入第一个值的 ASCII 码最后四位为 A，可以输入 j、z、J、Z
i 在字符串中位置为 4，偏移量为 4，键入第一个值的 ASCII 码最后四位为 4，可以输入 t、T、d、D
l 在字符串中位置为 F，偏移量为 F，键入第一个值的 ASCII 码最后四位为 F，可以输入 O、o
e 在字符串中位置为 5，偏移量为 5，键入第一个值的 ASCII 码最后四位为 5，可以输入 E、e、U、u
r 在字符串中位置为 6，偏移量为 6，键入第一个值的 ASCII 码最后四位为 6，可以输入 F、f、V、v
s 在字符串中位置为 7，偏移量为 7，键入第一个值的 ASCII 码最后四位为 7，可以输入 G、g、W、w、
输入字符串 jtoefg 即可通关

```
jtoefg
Good work!  On to the next...
```

Phase_6:

1：首先输入 6 个整数，并且让这 6 整数不相等，且都小于等于 6

```
08048db2 <phase_6>:
 8048db2:   56                      push   %esi
 8048db3:   53                      push   %ebx
 8048db4:   83 ec 44                sub    $0x44,%esp

 8048db7:   8d 44 24 10             lea    0x10(%esp),%eax
 8048dbb:   89 44 24 04             mov    %eax,0x4(%esp)
 8048dbf:   8b 44 24 50             mov    0x50(%esp),%eax
 8048dc3:   89 04 24                mov    %eax,(%esp)
 8048dc6:   e8 60 04 00 00          call   804922b <read_six_numbers>#输入六个数字

 8048dcb:   be 00 00 00 00          mov    $0x0,%esi

 8048dd0:   8b 44 b4 10             mov    0x10(%esp,%esi,4),%eax#保存*0x10(%esp)=x
 8048dd4:   83 e8 01                sub    $0x1,%eax     #x-1
 8048dd7:   83 f8 05                cmp    $0x5,%eax     #(x-1)和5比较
 8048dda:   76 05                   jbe    8048de1 <phase_6+0x2f>    #x<=6,跳转，否则bomb
 8048ddc:   e8 15 03 00 00          call   80490f6 <explode_bomb>    #限制键入值个数

 8048de1:   83 c6 01                add    $0x1,%esi     #esi中值+1
 8048de4:   83 fe 06                cmp    $0x6,%esi     #esi中值与6比较
 8048de7:   74 33                   je     8048e1c <phase_6+0x6a>    #相等跳转到0x8048e1c
 8048de9:   89 f3                   mov    %esi,%ebx     #ebx=esi

 8048deb:   8b 44 9c 10             mov    0x10(%esp,%ebx,4),%eax    #a[ebx]
 8048def:   39 44 b4 0c             cmp    %eax,0xc(%esp,%esi,4)     #a[ebx] a[esi-1]比较
 8048df3:   75 05                   jne    8048dfa <phase_6+0x48>    #不相等跳转(限制元素值)
 8048df5:   e8 fc 02 00 00          call   80490f6 <explode_bomb>    #相等bomb
```

```
 8048deb:   8b 44 9c 10             mov    0x10(%esp,%ebx,4),%eax    #a[ebx]
 8048def:   39 44 b4 0c             cmp    %eax,0xc(%esp,%esi,4)     #a[ebx] a[esi-1]比较
 8048df3:   75 05                   jne    8048dfa <phase_6+0x48>    #不相等跳转(限制元素值)
 8048df5:   e8 fc 02 00 00          call   80490f6 <explode_bomb>    #相等bomb

 8048dfa:   83 c3 01                add    $0x1,%ebx     #ebx中值+1
 8048dfd:   83 fb 05                cmp    $0x5,%ebx     #ebx中值与5比较
 8048e00:   7e e9                   jle    8048deb <phase_6+0x39>#ebx<=5跳转到0x8048deb

 8048e02:   eb cc                   jmp    8048dd0 <phase_6+0x1e>#无条件跳转到0x8048dd0
  #以上汇编代码主要实现，让输入的六个数不相等，且都小于等于6
```

2：查看内存中值，得到一个链表的数据

```
 8048e2c:   ba 3c c1 04 08          mov    $0x804c13c,%edx  #将0x804c13c地址传入edx
```

```
(gdb) p/x *0x804c13c@32
$2 = {0x17b, 0x1, 0x804c148, 0x374, 0x2, 0x804c154, 0x29b, 0x3, 0x804c160,
  0x3e3, 0x4, 0x804c16c, 0xa0, 0x5, 0x804c178, 0x194, 0x6, 0x0, 0x10, 0x0,
  0x0, 0x0, 0x0, 0x0, 0x0, 0x804a3b5, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}
```

1	2	3	4	5	6
0x17b	0x374	0x29b	0x3e3	0xa0	0x194

3：实现降序排列，对链表中的权值进行降序排列

```
8048e5f:   c7 40 08 00 00 00 00    movl   $0x0,0x8(%eax)
8048e66:   be 05 00 00 00          mov    $0x5,%esi
8048e6b:   8b 43 08                mov    0x8(%ebx),%eax    #下一个节点首地址传给eax
8048e6e:   8b 10                   mov    (%eax),%edx   #下一个节点权值传给edx
8048e70:   39 13                   cmp    %edx,(%ebx)   #这个节点权值和下一个节点权值进行比较
8048e72:   7d 05                   jge    8048e79 <phase_6+0xc7>#当前节点权值>=下一个节点权值
跳转(权值降序排列)
8048e74:   e8 7d 02 00 00          call   80490f6 <explode_bomb>
```

降序排列后：

4	2	3	6	1	5
0x3e3	0x374	0x29b	0x194	0x17b	0xa0

4：输入 4 2 3 6 1 5，验证结果正确性

```
4 2 3 6 1 5
Congratulations! You've defused the bomb!
```

最终 6 个实验结果：

```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
When I get angry, Mr. Bigglesworth gets upset.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
1 740
Halfway there!
4 2
So you got that one.  Try this one.
jtoefg
Good work!  On to the next...
4 2 3 6 1 5
Congratulations! You've defused the bomb!
```

隐藏关卡：

网上查找得知隐藏关卡在<phase_defused>函数中

```
cmpl   $0x6,0x804c3cc    #与6进行比较，等于6才能进入
```

接着发现几个立即数，猜测是字符串首地址，用 gdb 查看内容

```
(gdb) x/s 0x804a3a5
0x804a3a5:       "%d %d %s"
(gdb) x/s 0x804c4d0
0x804c4d0 <input_strings+240>:   ""
(gdb) x/4xw 0x804c4d0
0x804c4d0 <input_strings+240>:  0x00000000      0x00000000      0x00000000      0x00000000
(gdb) x/s 0x804a274
0x804a274:       "Curses, you've found the secret phase!"
(gdb) x/s 0x804a29c
0x804a29c:       "But finding it and solving it are quite different..."
(gdb) x/s 0x804a2d4
0x804a2d4:       "Congratulations! You've defused the bomb!"
(gdb) x/s 0x804a3ae
0x804a3ae:       "DrEvil"
```

其中"%d %d %s"是输入两个整数和一个字符串，"DrEvil"可能作为输入的字符串，由于第四关中也调用了函数

```
call   8048870 <__isoc99_sscanf@plt>#第四关也调用了这个函数
```

在第四关中输入两个整数后再输入字符串"DrEvil"即可进入隐藏关卡，在 6 关通完时会显现出隐藏关卡

在<secret_phase>中函数输入值作为 fun7 参数的第二个参数（以十进制输入），并且要求返回值为 4

```
8048f0e:   c7 04 24 88 c0 04 08     movl   $0x804c088,(%esp)#参数准备
8048f15:   e8 6d ff ff ff           call   8048e87 <fun7>#调用函数<fun7>
8048f1a:   83 f8 04                 cmp    $0x4,%eax     #返回为4，否则爆炸
8048f1d:   74 05                    je     8048f24 <secret_phase+0x4c>#fun7函数返回值要等于4
```

在<fun7>中，fun7 函数为递归函数

```
 8048e8b:   8b 54 24 20              mov    0x20(%esp),%edx#第一个参数A，即0x804c088
 8048e8f:   8b 4c 24 24              mov    0x24(%esp),%ecx#第二个参数B，输入的数
#*A>B，将(A+4)作为地址参数放入递归
8048e9d:   89 4c 24 04              mov    %ecx,0x4(%esp)
8048ea1:   8b 42 04                 mov    0x4(%edx),%eax
8048ea4:   89 04 24                 mov    %eax,(%esp)#将(A+4)作为地址传入递归
8048ea7:   e8 db ff ff ff           call   8048e87 <fun7>
8048eac:   01 c0                    add    %eax,%eax#函数返回值加倍
8048eae:   eb 23                    jmp    8048ed3 <fun7+0x4c>
#*A<B(*A!=B,否则结束递归)，将(A+8)作为地址进入递归
8048eb0:   b8 00 00 00 00           mov    $0x0,%eax#eax=0
8048eb5:   39 cb                    cmp    %ecx,%ebx#*A和B比较
8048eb7:   74 1a                    je     8048ed3 <fun7+0x4c>#相等结束递归，否则跳转
8048eb9:   89 4c 24 04              mov    %ecx,0x4(%esp)

8048ebd:   8b 42 08                 mov    0x8(%edx),%eax
8048ec0:   89 04 24                 mov    %eax,(%esp)
8048ec3:   e8 bf ff ff ff           call   8048e87 <fun7>
8048ec8:   8d 44 00 01              lea    0x1(%eax,%eax,1),%eax#将函数返回值x2再加1
```

通过寻找

```
(gdb) p/x *(0x804c088+4)
$7 = 0x804c094
(gdb) p/x *(0x804c094+4)
$8 = 0x804c0c4
(gdb) p/x *(0x804c0c4+8)
$9 = 0x804c10c
(gdb) x/4wx 0x804c10c
0x804c10c <n42>:        0x00000007      0x00000000      0x00000000      0x00000014
```

可得最终输入为 7，即可通关

通关答案为：

When I get angry, Mr. Bigglesworth gets upset.

0 1 1 2 3 5

1 7 4 0

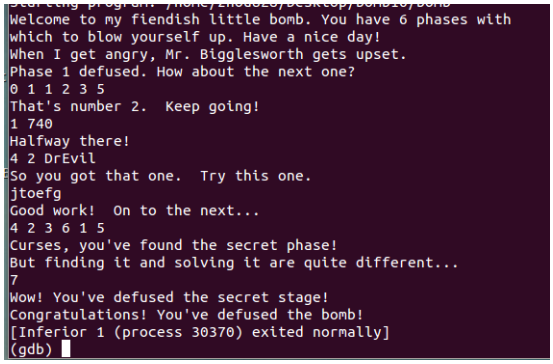
4 2 DrEvil

Jtoefg

4 2 3 6 1 5

7

通关结果如图：



收获与体会：通过爆炸实验，增强了我对汇编语言的理解，增强了自己对于调试的熟练度，对 Linux 系统的一些操作命令有一些了解和掌握。

实验成绩	

实验报告撰写说明

1. 实验题目和目的

请从实验指导资料中获取。

2．实验步骤和内容

包括：

（1）本次实验的要求；

（2）源程序清单或者主要伪代码；

（3）预期结果；

（4）上机执行或调试结果：包括原始数据、相应的运行结果和必要的说明（截图）；

3．实验体会

调试中遇到的问题及解决办法；若最终未完成调试，要试着分析原因；调试程序的心得与体会；对课程及实验的建议等。