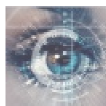
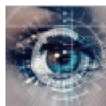
深入理解计算机系统——bomblab（炸弹实验）

原创

置顶 Mr.Slin 于 2020-03-03 21:34:58 发布 3449 收藏 47
分类专栏： 深入理解计算机系统 实验
版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/forest_one/article/details/104641575
版权



深入理解计算机系统 同时被 2 个专栏收录
3 篇文章 2 订阅
订阅专栏



实验
9 篇文章 1 订阅
订阅专栏

通过此次实验，提高阅读和理解汇编代码的能力，学习使用gdb调试工具。
word版报告下载：download.csdn.net/download/forest_one/12210296

一、实验要求与准备

1.1实验内容

<1>本次实验为熟悉汇编程序及其调试方法的实验。

<2>实验内容包含2个文件bomb（可执行文件）和bomb.c（c源文件）。

<3>使用gdb工具反汇编出汇编代码，结合c语言文件找到每个关卡的入口函数。

<4>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通关。

<5>本实验一共有7个关卡，包括6个普通关卡和1个隐藏关卡。

1.2 peda的安装与使用

在本实验中，我们使用了gdb-peda插件，gdb-peda具有更加友好的用户页面，使得调试更加有效，并且gdb-peda能够实时跟踪查看寄存器、反汇编语句以及栈帧之中的部分内容，并且在进行函数跳转时，提供了可能进行传递的参数，使得gdb调试更加可视化。

```
----------------------------------registers-----------------------------------]
AX: 0x27 ("'")
BX: 0xbffff1d4 --> 0xbffff396 ("/home/ubuntu/Desktop/bomb/bomb")
CX: 0xb7fd7000 ("Curses, you've found the secret phase!\n?\ny!\n 6 phases with\
")
DX: 0xb7fc1898 --> 0x0
SI: 0x0
DI: 0x0
BP: 0xbffff138 --> 0x0
SP: 0xbffff090 --> 0x804a2c0 ("But finding it and solving it are quite differen
...")
IP: 0x80492d2 (<phase_defused+119>:    call   0x8048800 <puts@plt>)
FLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)

   0x80492bf <phase_defused+100>:     mov    DWORD PTR [esp],0x804a298
   0x80492c6 <phase_defused+107>:     call   0x8048800 <puts@plt>
   0x80492cb <phase_defused+112>:     mov    DWORD PTR [esp],0x804a2c0
=> 0x80492d2 <phase_defused+119>:     call   0x8048800 <puts@plt>
   0x80492d7 <phase_defused+124>:     call   0x8048ebd <secret_phase>
   0x80492dc <phase_defused+129>:     mov    DWORD PTR [esp],0x804a2f8
   0x80492e3 <phase_defused+136>:     call   0x8048800 <puts@plt>
   0x80492e8 <phase_defused+141>:     mov    eax,DWORD PTR [esp+0x7c]

arg[0]: 0x804a2c0 ("But finding it and solving it are quite different...")

0000| 0xbffff090 --> 0x804a2c0 ("But finding it and solving it are quite differe
nt...")
0004| 0xbffff094 --> 0x804a3d2 ("DrEvil")
0008| 0xbffff098 --> 0xbffff0b4 --> 0x2
0012| 0xbffff09c --> 0xbffff0b8 --> 0x4
0016| 0xbffff0a0 --> 0xbffff0bc ("DrEvil")
0020| 0xbffff0a4 --> 0x804a3b7 ("%d %d %d %d %d %d")
0024| 0xbffff0a8 --> 0xbffff0e0 --> 0x1
0028| 0xbffff0ac --> 0xbffff0e4 --> 0x6
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x080492d2 in phase_defused ()
```

寄存器

汇编代码

可能传递的参数

stack

https://blog.csdn.net/forest_one

Peda的安装

```
1  git clone https://github.com/longld/peda.git ~/peda
2  echo "source ~/peda/peda.py" >> ~/.gdbinit
```

下面不想排版了，直接上之前写的报告的截图吧！！！！（写的有点繁琐）

我会给出word版（附带目录）的下载链接，可能大家看起来会方便一点！

二、实验任务

2.1 pahas_1

使用 gdb 调试

2.1.1 objdump 反汇编

反汇编，得到汇编代码如下：

https://blog.csdn.net/forest_one

```
08048b50 <phase_1>:
 8048b50:	83 ec 1c             	sub    $0x1c,%esp
 8048b53:	c7 44 24 04 a4 a1 04 	movl   $0x804a1a4,0x4(%esp)
 8048b5a:	08
 8048b5b:	8b 44 24 20          	mov    0x20(%esp),%eax
 8048b5f:	89 04 24             	mov    %eax,(%esp)
 8048b62:	e8 5d 04 00 00       	call   8048fc4 <strings_not_equal>
```

```
8048b67:    85 c0                   test   %eax,%eax
8048b69:    74 05                   je     8048b70 <phase_1+0x20>
8048b6b:    e8 66 05 00 00          call   80490d6 <explode_bomb>
8048b70:    83 c4 1c                add    $0x1c,%esp
8048b73:    c3                      ret
```

2.1.2 汇编分析

```
Dump of assembler code for function phase_1:
=> 0x08048b50 <+0>:   sub    $0x1c,%esp
   0x08048b53 <+3>:   movl   $0x804a1a4,0x4(%esp)   // 某一地址入栈
   0x08048b5b <+11>:  mov    0x20(%esp),%eax      // 输入字符串地址到 eax
   0x08048b5f <+15>:  mov    %eax,(%esp)          // 地址入栈
   0x08048b62 <+18>:  call   0x8048fc4 <strings_not_equal> //调用函数
   0x08048b67 <+23>:  test   %eax,%eax //函数返回值进行逻辑与运算
   0x08048b69 <+25>:  je     0x8048b70 <phase_1+32> //%eax=0 跳转
   0x08048b6b <+27>:  call   0x80490d6 <explode_bomb>
    //若上述跳转语句没有执行则爆炸
   0x08048b70 <+32>:  add    $0x1c,%esp
   0x08048b73 <+35>:  ret //phase_1 结束
```

https://blog.csdn.net/forest_one

2.1.3 具体调试

首先，输入字符串：abcd。进入调试我们看到调用的函数为<strings_not_equal>，即比较两个字符串是否相等。可以想到该函数需要的两个参数为个两个字符串的首地址。在调用函数<strings_not_equal>之前，mov 指令可能是进行了传参：

```
8048b53:    c7 44 24 04 a4 a1 04    movl   $0x804a1a4,0x4(%esp)
8048b5a:    08
8048b5b:    8b 44 24 20             mov    0x20(%esp),%eax
8048b5f:    89 04 24                mov    %eax,(%esp)
```

按照这个思路我们先查看一下地址 0x804a1a4：

```
(gdb) x/s 0x804a1a4
0x804a1a4:      "I am not part of the problem. I am a Republican."
```

由上图可知，我们知道了地址 0x804a1a4 存储的为一个字符串，并且不为我们输入的字符串，

我们猜测该字符串为进行比较的字符串之一，即正确结果。

接下来查看寄存器 eax：

```
(gdb) i r
eax            0x804c3e0        134530016
ecx            0x5      5
edx            0x1      1
ebx            0xbffff1d4       -1073745452
esp            0xbffff100       0xbffff100
ebp            0xbffff138       0xbffff138
esi            0x0      0
edi            0x0      0
```

```
eip            0x8048b5f        0x8048b5f <phase_1+15>
eflags         0x286    [ PF SF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
```

我们发现寄存器 eax 中存储的是一个地址，同样，我们查看地址 0x804c3e0 处的字符串：

```
(gdb) x/s 0x804c3e0
0x804c3e0 <input_strings>:      "abcd"
```

https://blog.csdn.net/forest_one

上述地址处存储的字符串为我们输入的字符串，故上述猜测是正确的，我们需要输入的是一个字符串：I am not part of the problem. I am a Republican.

为了了解比较的过程，我们进入函数<strings_not_equal>，分析汇编如下：

```
Dump of assembler code for function strings_not_equal:
=> 0x08048fc4 <+0>: sub     $0x10,%esp
   0x08048fc7 <+3>: mov     %ebx,0x4(%esp)                         接收传参
   0x08048fcb <+7>: mov     %esi,0x8(%esp)
   0x08048fcf <+11>:    mov     %edi,0xc(%esp)
   0x08048fd3 <+15>:    mov     0x14(%esp),%ebx
   0x08048fd7 <+19>:    mov     0x18(%esp),%esi
   0x08048fdb <+23>:    mov     %ebx,(%esp)                        string_length
   0x08048fde <+26>:    call    0x8048fab <string_length>          传参
   0x08048fe3 <+31>:    mov     %eax,%edi
   0x08048fe5 <+33>:    mov     %esi,(%esp)
   0x08048fe8 <+36>:    call    0x8048fab <string_length>
   0x08048fed <+41>:    mov     $0x1,%edx
   0x08048ff2 <+46>:    cmp     %eax,%edi                 //比较字符串长度
   0x08048ff4 <+48>:    jne     0x8049029 <strings_not_equal+101>
   0x08048ff6 <+50>:    movzbl  (%ebx),%eax
   0x08048ff9 <+53>:    mov     $0x0,%dl
   0x08048ffb <+55>:    test    %al,%al                   //判断是否为空字符
   0x08048ffd <+57>:    je      0x8049029 <strings_not_equal+101>
   0x08048fff <+59>:    mov     $0x1,%dl
   0x08049001 <+61>:    cmp     (%esi),%al                //比较第一位
```

https://blog.csdn.net/forest_one

```
   0x08049003 <+63>:    jne     0x8049029 <strings_not_equal+101>
   0x08049005 <+65>:    mov     $0x0,%eax
   0x0804900a <+70>:    jmp     0x8049014 <strings_not_equal+80>
   0x0804900c <+72>:    add     $0x1,%eax
   0x0804900f <+75>:    cmp     (%esi,%eax,1),%dl         // 判断下一位是否相同
   0x08049012 <+78>:    jne     0x8049024 <strings_not_equal+96>
   0x08049014 <+80>:    movzbl  0x1(%ebx,%eax,1),%edx     //输入的下一字符
   0x08049019 <+85>:    test    %dl,%dl                   //是否字符串结束
   0x0804901b <+87>:    jne     0x804900c <strings_not_equal+72>
   0x0804901d <+89>:    mov     $0x0,%edx
   0x08049022 <+94>:    jmp     0x8049029 <strings_not_equal+101>
```

```
0x08049022 <+94>:    jmp    0x8049029 <strings_not_equal+101>
0x08049024 <+96>:    mov    $0x1,%edx
0x08049029 <+101>:   mov    %edx,%eax
0x0804902b <+103>:   mov    0x4(%esp),%ebx
0x0804902f <+107>:   mov    0x8(%esp),%esi
0x08049033 <+111>:   mov    0xc(%esp),%edi
0x08049037 <+115>:   add    $0x10,%esp
0x0804903a <+118>:   ret
```

由上我们可以看出，字符串比较是通过先判断字符串长度、后循环逐位进行的。

我们输入字符串：I am not part of the problem. I am a Republican.进行验证：

```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am not part of the problem. I am a Republican.
Phase 1 defused. How about the next one?
```

可以看到 phase_1 过关成功，并给出了下一关的输入提示。

https://blog.csdn.net/forest_one

2.1.4 解除 bomb 密码

I am not part of the problem. I am a Republican.

2.2 pahas_2

使用 gdb 调试

2.2.1 objdump 反汇编

反汇编，得到汇编代码如下：

https://blog.csdn.net/forest_one

```
08048b74 <phase_2>:
 8048b74:   56                  push   %esi
 8048b75:   53                  push   %ebx
 8048b76:   83 ec 34            sub    $0x34,%esp
 8048b79:   8d 44 24 18         lea    0x18(%esp),%eax
 8048b7d:   89 44 24 04         mov    %eax,0x4(%esp)
 8048b81:   8b 44 24 40         mov    0x40(%esp),%eax
 8048b85:   89 04 24            mov    %eax,(%esp)
 8048b88:   e8 7e 06 00 00      call   804920b <read_six_numbers>
 8048b8d:   83 7c 24 18 00      cmpl   $0x0,0x18(%esp)
 8048b92:   75 07               jne    8048b9b <phase_2+0x27>
 8048b94:   83 7c 24 1c 01      cmpl   $0x1,0x1c(%esp)
 8048b99:   74 05               je     8048ba0 <phase_2+0x2c>
 8048b9b:   e8 36 05 00 00      call   80490d6 <explode_bomb>
 8048ba0:   8d 5c 24 20         lea    0x20(%esp),%ebx
 8048ba4:   8d 74 24 30         lea    0x30(%esp),%esi
 8048ba8:   8b 43 f8            mov    -0x8(%ebx),%eax
 8048bab:   03 43 fc            add    -0x4(%ebx),%eax
 8048bae:   39 03               cmp    %eax,(%ebx)
 8048bb0:   74 05               je     8048bb7 <phase_2+0x43>
 8048bb2:   e8 1f 05 00 00      call   80490d6 <explode_bomb>
```

```
8048bb7:    83 c3 04                add     $0x4,%ebx
8048bba:    39 f3                   cmp     %esi,%ebx
8048bbc:    75 ea                   jne     8048ba8 <phase_2+0x34>
8048bbe:    83 c4 34                add     $0x34,%esp
8048bc1:    5b                      pop     %ebx
8048bc2:    5e                      pop     %esi
8048bc3:    c3                      ret
```

https://blog.csdn.net/forest_one

2.2.2 汇编分析

```
Dump of assembler code for function phase_2:
=> 0x08048b74 <+0>:    push   %esi
   0x08048b75 <+1>:    push   %ebx
   0x08048b76 <+2>:    sub    $0x34,%esp
   0x08048b79 <+5>:    lea    0x18(%esp),%eax
   0x08048b7d <+9>:    mov    %eax,0x4(%esp)
   0x08048b81 <+13>:   mov    0x40(%esp),%eax
   0x08048b85 <+17>:   mov    %eax,(%esp)    //0x804c430 地址入栈(输入字符串地址)
   0x08048b88 <+20>:   call   0x804920b <read_six_numbers> //提取字符串中数字
   0x08048b8d <+25>:   cmpl   $0x0,0x18(%esp)          //判断是否为 0
   0x08048b92 <+30>:   jne    0x8048b9b <phase_2+39>  //不相等 bomb
   0x08048b94 <+32>:   cmpl   $0x1,0x1c(%esp)        //第二个数字和 0x1 是否相等
   0x08048b99 <+37>:   je     0x8048ba0 <phase_2+44>  //相等跳转
   0x08048b9b <+39>:   call   0x80490d6 <explode_bomb>  //不相等 bomb
   0x08048ba0 <+44>:   lea    0x20(%esp),%ebx    // 将 0x20(%esp)存入 ebx
   0x08048ba4 <+48>:   lea    0x30(%esp),%esi    // 将 0x30(%esp)存入 esi


   0x08048ba8 <+52>:   mov    -0x8(%ebx),%eax   //往前 2 个数字
   0x08048bab <+55>:   add    -0x4(%ebx),%eax   //往前 1 个数字和往前两个数字相加
   0x08048bae <+58>:   cmp    %eax,(%ebx)      //判断当前数字是否与前两个数字相等
   0x08048bb0 <+60>:   je     0x8048bb7 <phase_2+67>
   0x08048bb2 <+62>:   call   0x80490d6 <explode_bomb>
   0x08048bb7 <+67>:   add    $0x4,%ebx
   0x08048bba <+70>:   cmp    %esi,%ebx   //判断是否完成 6 个数字判断
   0x08048bbc <+72>:   jne    0x8048ba8 <phase_2+52>//不相等跳转循环
   0x08048bbe <+74>:   add    $0x34,%esp
   //phase_2 结束，接下来进行调用者栈帧的恢复操作
   0x08048bc1 <+77>:   pop    %ebx
   0x08048bc2 <+78>:   pop    %esi
   0x08048bc3 <+79>:   ret
```

https://blog.csdn.net/forest_one

2.2.3 具体调试

我们先看函数<read_six_numbers>即读入 6 个数字。再看其上面的几个传参指令：一个是把地址 0x40(%ebp)传了进去；另一个是传入了地址 0x18(%ebp)储存的值。按照第一关的思路，我们可以想到 0x40(%ebp)储存的值为我们输入字符串的首地址。这里对于对于输入做一个说明：我们通过看 bomb.c 文件可以知道在每一关执行之前，都要执行一个 read_line()函数，也就是你的输入是在此时被读入的。而在具体的每一关里面的输入(比如这里的 <read_six_numbers>)只是在你刚刚刚

输入的一行数据里面进行操作，而你输入的数据(当作字符串处理)储存的首地址储存在地址0x40(%ebp)，后续几关的读入也是同样的原理。

既然 0x40(%ebp)是用来进行<read_six_numbers>操作的，那么地址 0x18(%ebp)用来干嘛的呢？我们从输入的一行里面进行读入数据后，输入的数据需要储存起来的。更具后续对地址0x18(%ebp)储存的值进行判断的操作，可以大概想到地址 0x18(%ebp)是用来保存读入的数据的。而我们需要保存的是六个数字，所以这个地址只是六个数据的第一个其中一个，其他的数据应该储存在该地址加上一个偏移量的地址中。下面我们来进行验证：

查看 0x40(%ebp)地址：

```
(gdb) x/s 0x804c430
0x804c430 <input_strings+80>:   "1 2 3 4 5 6"
```

查看 0x18(%ebp)地址：

```
(gdb) x/6wx 0xbffff0f8
0xbffff0f8:     0x00000001      0x00000002      0x00000003      0x00000004
0xbffff108:     0x00000005      0x00000006
```

由上我们可以看出，<read_six_numbers>函数从输入的字符串中读取 6 个字符并转化成 6 个整数保存在了内存当中，用以接下来的比较。

接下来我们继续向下看：首先是分别将地址 0x18(%ebp)和地址 0x1c(%ebp)的值分别和 0 与 1 比较，不想等就爆炸。这里说明我们输入的前两个数应该为 0 和 1。接下来进入了一个循环，我们先看循环的终止条件：cmp %esi，%ebx。即循环执行四次后终止。

对于每次循环的操作都是将当前（%ebx）的低两个地址储存的数据之和与地址（%ebx）储存的数据进行比较不相等则爆炸。

总结一下通过密码就是：你需要输入 6 个数，第一个数为 0,第二个数为 1，其余的数就是其前面两个数之和。就是要输入斐波拉契数列的前六项。如下：https://blog.csdn.net/forest_one

```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am not part of the problem. I am a Republican.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
```

第二关过关成功。

```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am not part of the problem. I am a Republican.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
```

2.2.4 解除 bomb 密码

0 1 1 2 3 5

//有 bug　前 6 位正确即可

https://blog.csdn.net/forest_one

2.3 pahas_3

使用 gdb-peda 调试

2.3.1 objdump 反汇编

反汇编，得到汇编代码如下：

```
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)
 8048bcf:	8d 44 24 18          	lea    0x18(%esp),%eax
 8048bd3:	89 44 24 08          	mov    %eax,0x8(%esp)
 8048bd7:	c7 44 24 04 c3 a3 04 	movl   $0x804a3c3,0x4(%esp)
 8048bde:	08 
 8048bdf:	8b 44 24 30          	mov    0x30(%esp),%eax
 8048be3:	89 04 24             	mov    %eax,(%esp)
 8048be6:	e8 85 fc ff ff       	call   8048870 <__isoc99_sscanf@plt>
 8048beb:	83 f8 01             	cmp    $0x1,%eax
 8048bee:	7f 05                	jg     8048bf5 <phase_3+0x31>
 8048bf0:	e8 e1 04 00 00       	call   80490d6 <explode_bomb>
 8048bf5:	83 7c 24 18 07       	cmpl   $0x7,0x18(%esp)
 8048bfa:	77 3c                	ja     8048c38 <phase_3+0x74>
 8048bfc:	8b 44 24 18          	mov    0x18(%esp),%eax
 8048c00:	ff 24 85 00 a2 04 08 	jmp    *0x804a200(,%eax,4)
 8048c07:	b8 24 02 00 00       	mov    $0x224,%eax
```

https://blog.csdn.net/forest_one

```
 8048c0c:	eb 3b                	jmp    8048c49 <phase_3+0x85>
 8048c0e:	b8 82 03 00 00       	mov    $0x382,%eax
 8048c13:	eb 34                	jmp    8048c49 <phase_3+0x85>
 8048c15:	b8 32 03 00 00       	mov    $0x332,%eax
 8048c1a:	eb 2d                	jmp    8048c49 <phase_3+0x85>
 8048c1c:	b8 0c 01 00 00       	mov    $0x10c,%eax
 8048c21:	eb 26                	jmp    8048c49 <phase_3+0x85>
 8048c23:	b8 c3 03 00 00       	mov    $0x3c3,%eax
 8048c28:	eb 1f                	jmp    8048c49 <phase_3+0x85>
 8048c2a:	b8 87 01 00 00       	mov    $0x187,%eax
 8048c2f:	eb 18                	jmp    8048c49 <phase_3+0x85>
 8048c31:	b8 1c 02 00 00       	mov    $0x21c,%eax
 8048c36:	eb 11                	jmp    8048c49 <phase_3+0x85>
 8048c38:	e8 99 04 00 00       	call   80490d6 <explode_bomb>
 8048c3d:	b8 00 00 00 00       	mov    $0x0,%eax
 8048c42:	eb 05                	jmp    8048c49 <phase_3+0x85>
 8048c44:	b8 41 03 00 00       	mov    $0x341,%eax
 8048c49:	3b 44 24 1c          	cmp    0x1c(%esp),%eax
 8048c4d:	74 05                	je     8048c54 <phase_3+0x90>
 8048c4f:	e8 82 04 00 00       	call   80490d6 <explode_bomb>
 8048c54:	83 c4 2c             	add    $0x2c,%esp
 8048c57:	c3                   	ret    
```

https://blog.csdn.net/forest_one

2.3.2 汇编分析

部分代码：

```
=> 0x08048bc4 <+0>:    sub    esp, 0x2c   //分配空间
   0x08048bc7 <+3>:    lea    eax, [esp+0x1c]  //地址给 eax
   0x08048bcb <+7>:    mov    DWORD PTR [esp+0xc], eax  //eax 放入栈帧中
   0x08048bcf <+11>:   lea    eax, [esp+0x18]
```

```
0x08048bd3 <+15>:  mov    DWORD PTR [esp+0x8],eax
0x08048bd7 <+19>:  mov    DWORD PTR [esp+0x4],0x804a3c3 //放入一个地址到栈帧
0x08048bdf <+27>:  mov    eax,DWORD PTR [esp+0x30]
0x08048be3 <+31>:  mov    DWORD PTR [esp],eax
0x08048be6 <+34>:  call   0x8048870 <__isoc99_sscanf@plt> //更高要求输入函数
0x08048beb <+39>:  cmp    eax,0x1 //比较函数返回值与 1
0x08048bee <+42>:  jg     0x8048bf5 <phase_3+49>//大于跳转
0x08048bf0 <+44>:  call   0x80490d6 <explode_bomb>//否则 bomb
0x08048bf5 <+49>:  cmp    DWORD PTR [esp+0x18],0x7 //与 7 比较
0x08048bfa <+54>:  ja     0x8048c38 <phase_3+116> //否则 bomb
0x08048bfc <+56>:  mov    eax,DWORD PTR [esp+0x18] //取栈帧中的数
0x08048c00 <+60>:  jmp    DWORD PTR [eax*4+0x804a200] //根据数字*4 进行跳转
```

2.3.3 具体调试

我们随意输入一个字符串，开始进行调试，在调试过程中，我们注意到在调用 sscanf 输入函数时，进行传参时，我们发现直接写入了一个地址到栈帧中，通过 gdb-peda 查看栈帧如下：

https://blog.csdn.net/forest_one

```
mov    DWORD PTR [esp+0x4],0x804a3c3
```

```
0004| 0xbffff0f4 --> 0x804a3c3 ("%d %d")
```

由上我们可以看出，这个地址写的是一个格式规范，“%d %d”表示两个整数，经过查找资料可以知道“%d %d”写在内存的可读区域中，之后的题目的 sscanf 均为带有规范的输入函数，由此我们可以知道，本题需要输入的为两个整数。

继续进行，比较输入的第一个数字和 7

```
cmp    DWORD PTR [esp+0x18],0x7
```

```
ESP: 0xbffff0f0    0024| 0xbffff108 --> 0x2
```

大于 7 则爆炸，并且由于是无符号比较，所以输入还要>=0, 由此可以得出第一个输入数字要大于等于 0 小于等于 7。

之后根据第一个输入的数字进行跳转：

```
jmp    DWORD PTR [eax*4+0x804a200]
```

我们可以由此看出，这是一个 swich 跳转语句，根据输入的第一个数字跳转到跳转表进行跳转。以输入第一个数字为 2 为例：

```
[------------------------------------code-------------------------------------]
   0x8048bf5 <phase_3+49>:      cmp    DWORD PTR [esp+0x18],0x7
   0x8048bfa <phase_3+54>:      ja     0x8048c38 <phase_3+116>
   0x8048bfc <phase_3+56>:      mov    eax,DWORD PTR [esp+0x18]
=> 0x8048c00 <phase_3+60>:      jmp    DWORD PTR [eax*4+0x804a200]
 | 0x8048c07 <phase_3+67>:      mov    eax,0x224
 | 0x8048c0c <phase_3+72>:      jmp    0x8048c49 <phase_3+133>
 | 0x8048c0e <phase_3+74>:      mov    eax,0x382
 | 0x8048c13 <phase_3+79>:      jmp    0x8048c49 <phase_3+133>
 |->   0x8048c0e <phase_3+74>:  mov    eax,0x382
       0x8048c13 <phase_3+79>:  jmp    0x8048c49 <phase_3+133>
       0x8048c15 <phase_3+81>:  mov    eax,0x332
       0x8048c1a <phase_3+86>:  jmp    0x8048c49 <phase_3+133>
                                                              JUMP is taken
```

根据输入的第一个数字 2 进行跳转

https://blog.csdn.net/forest_one

```
0x8048c0e <phase_3+74>:      mov    eax,0x382
```

```
=> 0x8048c13 <phase_3+79>:    jmp    0x8048c49 <phase_3+133>
 | 0x8048c15 <phase_3+81>:    mov    eax,0x332
 | 0x8048c1a <phase_3+86>:    jmp    0x8048c49 <phase_3+133>
 | 0x8048c1c <phase_3+88>:    mov    eax,0x10c
 | 0x8048c21 <phase_3+93>:    jmp    0x8048c49 <phase_3+133>
 |-> 0x8048c49 <phase_3+133>: cmp    eax,DWORD PTR [esp+0x1c]
     0x8048c4d <phase_3+137>: je     0x8048c54 <phase_3+144>
     0x8048c4f <phase_3+139>: call   0x80490d6 <explode_bomb>
```

将 0x323 放入 eax 并返回，将 eax（0x323）中与第二个数字比较，如果不相等 bomb，否则第三关成功！所以，我们推测出答案有多种组合，其中一个为 2 0x323。综合分析从反汇编代码可以看出其他答案。

https://blog.csdn.net/forest_one

```
8048c07:  b8 24 02 00 00     mov    $0x224,%eax
8048c0c:  eb 3b              jmp    8048c49 <phase_3+0x85>
8048c0e:  b8 82 03 00 00     mov    $0x382,%eax
8048c13:  eb 34              jmp    8048c49 <phase_3+0x85>
8048c15:  b8 32 03 00 00     mov    $0x332,%eax
8048c1a:  eb 2d              jmp    8048c49 <phase_3+0x85>
8048c1c:  b8 0c 01 00 00     mov    $0x10c,%eax
8048c21:  eb 26              jmp    8048c49 <phase_3+0x85>
8048c23:  b8 c3 03 00 00     mov    $0x3c3,%eax
8048c28:  eb 1f              jmp    8048c49 <phase_3+0x85>
8048c2a:  b8 87 01 00 00     mov    $0x187,%eax
8048c2f:  eb 18              jmp    8048c49 <phase_3+0x85>
8048c31:  b8 1c 02 00 00     mov    $0x21c,%eax
8048c36:  eb 11              jmp    8048c49 <phase_3+0x85>
```

验证结果：

```
2 898
Halfway there!
```

第三关通过。

2.3.4 解除 bomb 密码

2 898（不唯一）

https://blog.csdn.net/forest_one

2.4 pahas_4

使用 gdb-peda 调试

2.4.1 objdump 反汇编

反汇编，得到汇编代码如下：

```
08048cc5 <phase_4>:
 8048cc5:  83 ec 2c              sub    $0x2c,%esp
 8048cc8:  8d 44 24 1c           lea    0x1c(%esp),%eax
 8048ccc:  89 44 24 0c           mov    %eax,0xc(%esp)
 8048cd0:  8d 44 24 18           lea    0x18(%esp),%eax
 8048cd4:  89 44 24 08           mov    %eax,0x8(%esp)
 8048cd8:  c7 44 24 04 c3 a3 04  movl   $0x804a3c3,0x4(%esp)
 8048cdf:  08
 8048ce0:  8b 44 24 30           mov    0x30(%esp),%eax
```

```
8048ce4:   89 04 24                mov    %eax,(%esp)
8048ce7:   e8 84 fb ff ff          call   8048870 <__isoc99_sscanf@plt>
8048cec:   83 f8 02                cmp    $0x2,%eax
8048cef:   75 0d                   jne    8048cfe <phase_4+0x39>
8048cf1:   8b 44 24 18             mov    0x18(%esp),%eax
8048cf5:   85 c0                   test   %eax,%eax
8048cf7:   78 05                   js     8048cfe <phase_4+0x39>
8048cf9:   83 f8 0e                cmp    $0xe,%eax
8048cfc:   7e 05                   jle    8048d03 <phase_4+0x3e>
8048cfe:   e8 d3 03 00 00          call   80490d6 <explode_bomb>
```

```
8048d03:   c7 44 24 08 0e 00 00    movl   $0xe,0x8(%esp)
8048d0a:   00
8048d0b:   c7 44 24 04 00 00 00    movl   $0x0,0x4(%esp)
8048d12:   00
8048d13:   8b 44 24 18             mov    0x18(%esp),%eax
8048d17:   89 04 24                mov    %eax,(%esp)
8048d1a:   e8 39 ff ff ff          call   8048c58 <func4>
8048d1f:   83 f8 04                cmp    $0x4,%eax
8048d22:   75 07                   jne    8048d2b <phase_4+0x66>
8048d24:   83 7c 24 1c 04          cmpl   $0x4,0x1c(%esp)
8048d29:   74 05                   je     8048d30 <phase_4+0x6b>
8048d2b:   e8 a6 03 00 00          call   80490d6 <explode_bomb>
8048d30:   83 c4 2c                add    $0x2c,%esp
8048d33:   c3                      ret
```

https://blog.csdn.net/forest_one

2.4.2 汇编分析

```
Dump of assembler code for function phase_4:
=> 0x08048cc5 <+0>:    sub    esp,0x2c
   0x08048cc8 <+3>:    lea    eax,[esp+0x1c]
   0x08048ccc <+7>:    mov    DWORD PTR [esp+0xc],eax        }
   0x08048cd0 <+11>:   lea    eax,[esp+0x18]                 }
   0x08048cd4 <+15>:   mov    DWORD PTR [esp+0x8],eax        }
   0x08048cd8 <+19>:   mov    DWORD PTR [esp+0x4],0x804a3c3  }  格式输入
   0x08048ce0 <+27>:   mov    eax,DWORD PTR [esp+0x30]       }
   0x08048ce4 <+31>:   mov    DWORD PTR [esp],eax            }
```

https://blog.csdn.net/forest_one

```
   0x08048ce7 <+34>:   call   0x8048870 <__isoc99_sscanf@plt>
   0x08048cec <+39>:   cmp    eax,0x2
   0x08048cef <+42>:   jne    0x8048cfe <phase_4+57>
   0x08048cf1 <+44>:   mov    eax,DWORD PTR [esp+0x18]  //输入的第一个数字
   0x08048cf5 <+48>:   test   eax,eax                   //判断是否为负数
   0x08048cf7 <+50>:   js     0x8048cfe <phase_4+57>//为负数则 bomb
   0x08048cf9 <+52>:   cmp    eax,0xe                   //比较第一个数字和 14
   0x08048cfc <+55>:   jle    0x8048d03 <phase_4+62>
 //小于等于 14 跳转+62，否则 bomb
   0x08048cfe <+57>:   call   0x80490d6 <explode_bomb>
   0x08048d03 <+62>:   mov    DWORD PTR [esp+0x8],0xe   //传参 14（第三个参数）
```

```
0x08048d0b <+70>:  mov    DWORD PTR [esp+0x4],0x0  //传参 0 （第二个参数）
0x08048d13 <+78>:  mov    eax,DWORD PTR [esp+0x18]
0x08048d17 <+82>:  mov    DWORD PTR [esp],eax
   //传参第一个输入数（第一个参数）
0x08048d1a <+85>:  call   0x8048c58 <func4>   //进入递归
0x08048d1f <+90>:  cmp    eax,0x4             //需要返回 4
0x08048d22 <+93>:  jne    0x8048d2b <phase_4+102> //不相等 bomb
0x08048d24 <+95>:  cmp    DWORD PTR [esp+0x1c],0x4 //比较
0x08048d29 <+100>: je     0x8048d30 <phase_4+107>
0x08048d2b <+102>: call   0x80490d6 <explode_bomb>
0x08048d30 <+107>: add    esp,0x2c
0x08048d33 <+110>: ret
```

https://blog.csdn.net/forest_one

2.4.3 具体调试

我们任意输入字符串开始进行调试，根据格式输入：

```
[------------------------------------code------------------------------------]
   0x8048cd0 <phase_4+11>:    lea    eax,[esp+0x18]
   0x8048cd4 <phase_4+15>:    mov    DWORD PTR [esp+0x8],eax
   0x8048cd8 <phase_4+19>:    mov    DWORD PTR [esp+0x4],0x804a3c3
=> 0x8048ce0 <phase_4+27>:    mov    eax,DWORD PTR [esp+0x30]
   0x8048ce4 <phase_4+31>:    mov    DWORD PTR [esp],eax
   0x8048ce7 <phase_4+34>:    call   0x8048870 <__isoc99_sscanf@plt>
   0x8048cec <phase_4+39>:    cmp    eax,0x2
   0x8048cef <phase_4+42>:    jne    0x8048cfe <phase_4+57>
```

```
0x804a3c3 ("%d %d")
```

由此可以得出，我们我们需要输入的为两个整数，且第一个数字为大于等于 0 小于 14

```
0x8048cf5 <phase_4+48>:    test   eax,eax
0x8048cf7 <phase_4+50>:    js     0x8048cfe <phase_4+57>
0x8048cf9 <phase_4+52>:    cmp    eax,0xe
0x8048cfc <phase_4+55>:    jle    0x8048d03 <phase_4+62>
```

接下来，我们通过汇编分析可以得知，在调用 func4 之前进行了三次传参操作，可以知道 func4 函数有三个参数，且进行调用时，按照汇编传参规范，依次传入第三个参数为 0xe，第二个参数为

0x0，第三个参数为输入第一个数字。

```
0x8048d03 <phase_4+62>:    mov    DWORD PTR [esp+0x8],0xe
0x8048d0b <phase_4+70>:    mov    DWORD PTR [esp+0x4],0x0
0x8048d13 <phase_4+78>:    mov    eax,DWORD PTR [esp+0x18]
0x8048d17 <phase_4+82>:    mov    DWORD PTR [esp],eax
0x8048d1a <phase_4+85>:    call   0x8048c58 <func4>
```

接下来，我们对调用的函数进行分析，由分析易知，函数 func4 为递归函数，不断调用自身，结合 phase_4 中的内容，最后需要返回的数字为 4。

```
Dump of assembler code for function func4:
=> 0x08048c58 <+0>:   sub    esp,0x1c
   0x08048c5b <+3>:   mov    DWORD PTR [esp+0x14],ebx
   0x08048c5f <+7>:   mov    DWORD PTR [esp+0x18],esi
   0x08048c63 <+11>:  mov    edx,DWORD PTR [esp+0x20]
```

```
//接收传入第一个输入参数
0x08048c67 <+15>:  mov    eax,DWORD PTR [esp+0x24] //接收传参 0



0x08048c6b <+19>:  mov    ebx,DWORD PTR [esp+0x28] //14
0x08048c6f <+23>:  mov    ecx,ebx
0x08048c71 <+25>:  sub    ecx,eax
0x08048c73 <+27>:  mov    esi,ecx    //esi=ecx=ecx-eax=ebx-eax
0x08048c75 <+29>:  shr    esi,0x1f    // 取 esi 符号位
0x08048c78 <+32>:  add    ecx,esi    //加符号位
0x08048c7a <+34>:  sar    ecx,1      //除以 2
0x08048c7c <+36>:  add    ecx,eax    //ecx=ecx+eax
0x08048c7e <+38>:  cmp    ecx,edx   //比较 ecx 和第一个输入数
0x08048c80 <+40>:  jle    0x8048c99 <func4+65>
//小于等于跳转，否则继续即大于
0x08048c82 <+42>:  sub    ecx,0x1                 //ecx=ecx-1
0x08048c85 <+45>:  mov    DWORD PTR [esp+0x8],ecx    //传参第三个参数
0x08048c89 <+49>:  mov    DWORD PTR [esp+0x4],eax    //传参第二个参数
0x08048c8d <+53>:  mov    DWORD PTR [esp],edx        //传参第一个参数
0x08048c90 <+56>:  call   0x8048c58 <func4>     //进入第一个条件递归
0x08048c95 <+61>:  add    eax,eax                 //递归结果*2
0x08048c97 <+63>:  jmp    0x8048cb9 <func4+97>// return
0x08048c99 <+65>:  mov    eax,0x0          //eax=0
0x08048c9e <+70>:  cmp    ecx,edx          //比较第一个输入和 ecx
0x08048ca0 <+72>:  jge    0x8048cb9 <func4+97>
 //ecx 大于等于第一个输入则跳转，否则继续即小于
0x08048ca2 <+74>:  mov    DWORD PTR [esp+0x8],ebx    //传参第三个参数
0x08048ca6 <+78>:  add    ecx,0x1                 //ecx=ecx+1
0x08048ca9 <+81>:  mov    DWORD PTR [esp+0x4],ecx    //传参第二个参数
0x08048cad <+85>:  mov    DWORD PTR [esp],edx        //传参第一个参数
0x08048cb0 <+88>:  call   0x8048c58 <func4>       //进入第二个条件递归



0x08048cb5 <+93>:  lea    eax,[eax+eax*1+0x1]      //递归结果*2+1
0x08048cb9 <+97>:  mov    ebx,DWORD PTR [esp+0x14]
0x08048cbd <+101>: mov    esi,DWORD PTR [esp+0x18]
0x08048cc1 <+105>: add    esp,0x1c
0x08048cc4 <+108>: ret
```

http://blog.csdn.net/forest_one

https://blog.csdn.net/forest_one

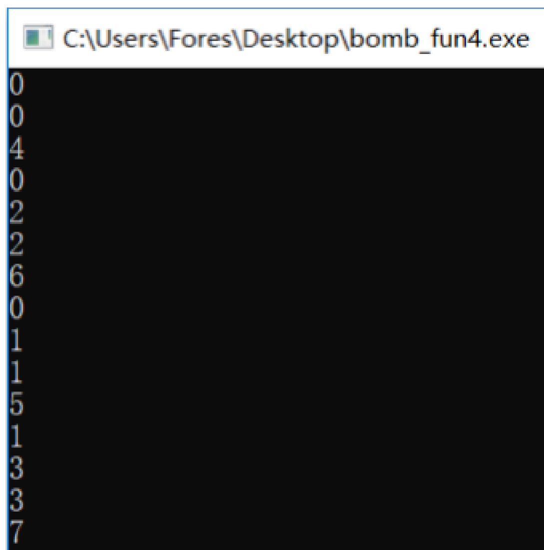
根据汇编代码我们可以写出对应的 C++代码如下图;

```cpp
#include<iostream>
using namespace std;
int f(int c1,int c2,int c3)
{
    int p=c3-c2;
    int r; //eax
    int a; //ecx
    if(p<0) a=((c3-c2)+1)/2+c2;
    else  a=(c3-c2)/2+c2;
    if(a>c1)
    return 2*f(c1,c2,a-1);
    r=0;
    if(a<c1)
    return 2*f(c1,a+1,c3)+1;
    return r;
}
int main()
{
   for(int i=0;i<=14;i++)
   cout<<f(i,0,14)<<endl;
```

```
}
```

https://blog.csdn.net/forest_one

我们根据前面的汇编分析，知道函数需要返回的数为 4，我们运行代码可以得到以下的结果：



从以上结果可以看出，当输入为 2 时，函数可以返回 4，故我们可以得出输入的两个数字当中第一个为 2，而之后输入的第二个数字与 4 进行比较，不相等则 bomb。

```
0x8048d24 <phase_4+95>:     cmp    DWORD PTR [esp+0x1c],0x4
0x8048d29 <phase_4+100>:    je     0x8048d30 <phase_4+107>
```

由以上分析可以得出，我们需要输入的数字为 2 4.第一个数字为递归函数提供第一个参数，第二个参数为直接比较数字。

验证结果：

https://blog.csdn.net/forest_one

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

第四关通过。

2.4.4 解除 bomb 密码

2 4

2.5 pahas_5

使用 gdb-pade 调试

2.5.1 objdump 反汇编

反汇编，得到汇编代码如下：

```
08048d34 <phase_5>:
 8048d34:   53                  push   %ebx
 8048d35:   83 ec 18            sub    $0x18,%esp
 8048d38:   8b 5c 24 20         mov    0x20(%esp),%ebx
 8048d3c:   89 1c 24            mov    %ebx,(%esp)
```

```
8048d3f:    e8 67 02 00 00          call   8048fab <string_length>
8048d44:    83 f8 06                cmp    $0x6,%eax
8048d47:    74 05                   je     8048d4e <phase_5+0x1a>
8048d49:    e8 88 03 00 00          call   80490d6 <explode_bomb>
```

https://blog.csdn.net/forest_one

```
8048d44:    83 f8 06                cmp    $0x6,%eax
8048d47:    74 05                   je     8048d4e <phase_5+0x1a>
8048d49:    e8 88 03 00 00          call   80490d6 <explode_bomb>
8048d4e:    ba 00 00 00 00          mov    $0x0,%edx
8048d53:    b8 00 00 00 00          mov    $0x0,%eax
8048d58:    0f be 0c 03             movsbl (%ebx,%eax,1),%ecx
8048d5c:    83 e1 0f                and    $0xf,%ecx
8048d5f:    03 14 8d 20 a2 04 08    add    0x804a220(,%ecx,4),%edx
8048d66:    83 c0 01                add    $0x1,%eax
8048d69:    83 f8 06                cmp    $0x6,%eax
8048d6c:    75 ea                   jne    8048d58 <phase_5+0x24>
8048d6e:    83 fa 36                cmp    $0x36,%edx
8048d71:    74 05                   je     8048d78 <phase_5+0x44>
8048d73:    e8 5e 03 00 00          call   80490d6 <explode_bomb>
8048d78:    83 c4 18                add    $0x18,%esp
8048d7b:    5b                      pop    %ebx
8048d7c:    c3                      ret
```

2.5.2 汇编分析

```
Dump of assembler code for function phase_5:
=> 0x08048d34 <+0>:    push   ebx
```

https://blog.csdn.net/forest_one

```
   0x08048d35 <+1>:    sub    esp,0x18
   0x08048d38 <+4>:    mov    ebx,DWORD PTR [esp+0x20]  //输入字符串地址入 ebx
   0x08048d3c <+8>:    mov    DWORD PTR [esp],ebx        //输入字符串地址入栈
   0x08048d3f <+11>:   call   0x8048fab <string_length>
   0x08048d44 <+16>:   cmp    eax,0x6                   //eax 需要=6 否则 bomb
   0x08048d47 <+19>:   je     0x8048d4e <phase_5+26>  //相等不 bomb
   0x08048d49 <+21>:   call   0x80490d6 <explode_bomb>
   0x08048d4e <+26>:   mov    edx,0x0
   0x08048d53 <+31>:   mov    eax,0x0
   0x08048d58 <+36>:   movsx  ecx,BYTE PTR [ebx+eax*1]   //取某一地址处的内容
   0x08048d5c <+40>:   and    ecx,0xf       //取字符串某一字符的低 4 位
   0x08048d5f <+43>:   add    edx,DWORD PTR [ecx*4+0x804a220]
    //edx+地址存储的内容
   0x08048d66 <+50>:   add    eax,0x1       //eax+1
   0x08048d69 <+53>:   cmp    eax,0x6       //是否=6 不等于则继续循环
   0x08048d6c <+56>:   jne    0x8048d58 <phase_5+36>
   0x08048d6e <+58>:   cmp    edx,0x36   //edx 不等于 0x36，则 bomb
   0x08048d71 <+61>:   je     0x8048d78 <phase_5+68>
   0x08048d73 <+63>:   call   0x80490d6 <explode_bomb>
   0x08048d78 <+68>:   add    esp,0x18
   0x08048d7b <+71>:   pop    ebx
   0x08048d7c <+72>:   ret
```

https://blog.csdn.net/forest_one

2.5.3 具体调试

我们通过分析汇编，可以知道闯关成功的条件时，对 edx 根据 ecx*4+0x804a220 不断进行累加，通过 6 次累加，使得 edx=0x36，我们首先来看一下从地址 0x804a220 开始 16 个地址内的内容。为什么是 16 个？因为在汇编程序中，ecx 为对输入的字符取低 4 位，最大所能表示的是 15，故能够进行相加操作的数存储在从地址 0x804a220 开始 16 个地址内。

```
gdb-peda$ x/16wx 0x804a220
0x804a220 <array.2956>: 0x00000002      0x0000000a      0x00000006      0x000000
01
0x804a230 <array.2956+16>:      0x0000000c      0x00000010      0x00000009      0
x00000003
0x804a240 <array.2956+32>:      0x00000004      0x00000007      0x0000000e      0
x00000005
0x804a250 <array.2956+48>:      0x0000000b      0x00000008      0x0000000f      0
x0000000d
```

可见，可能在 edx 进行累加的数字为以上的 16 个数字，我们需要凑出 0x36，通过观察我们可以看出 5*0xa+0x4，而 0xa 是第二个数字，同时 0x4 是第八个数字，由此我们知道，我们需要的 6 个字符为 aaaaah，因为 a 的 assci 码的十六进制的低四位为 1，而 h 的 assci 码的十六进制的低四位为 8，所以满足以上条件。

验证结果：

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

https://blog.csdn.net/forest_one

第五关通过。

2.5.4 解除 bomb 密码

ahhhhh

2.6 pahas_6

使用 gdb-peda 调试

2.6.1 objdump 反汇编

反汇编，得到汇编代码如下：

https://blog.csdn.net/forest_one

```
08048d7d <phase_6>:
 8048d7d:   56                      push   %esi
 8048d7e:   53                      push   %ebx
 8048d7f:   83 ec 44                sub    $0x44,%esp
 8048d82:   8d 44 24 10             lea    0x10(%esp),%eax
 8048d86:   89 44 24 04             mov    %eax,0x4(%esp)
 8048d8a:   8b 44 24 50             mov    0x50(%esp),%eax
 8048d8e:   89 04 24                mov    %eax,(%esp)
 8048d91:   e8 75 04 00 00          call   804920b <read_six_numbers>
 8048d96:   be 00 00 00 00          mov    $0x0,%esi
 8048d9b:   8b 44 b4 10             mov    0x10(%esp,%esi,4),%eax
 8048d9f:   83 e8 01                sub    $0x1,%eax
```

```
8048da2:   83 f8 05             cmp    $0x5,%eax
8048da5:   76 05                jbe    8048dac <phase_6+0x2f>
8048da7:   e8 2a 03 00 00       call   80490d6 <explode_bomb>
8048dac:   83 c6 01             add    $0x1,%esi
8048daf:   83 fe 06             cmp    $0x6,%esi
8048db2:   74 1b                je     8048dcf <phase_6+0x52>
8048db4:   89 f3                mov    %esi,%ebx
8048db6:   8b 44 9c 10          mov    0x10(%esp,%ebx,4),%eax
8048dba:   39 44 b4 0c          cmp    %eax,0xc(%esp,%esi,4)
8048dbe:   75 05                jne    8048dc5 <phase_6+0x48>
8048dc0:   e8 11 03 00 00       call   80490d6 <explode_bomb>
8048dc5:   83 c3 01             add    $0x1,%ebx
8048dc8:   83 fb 05             cmp    $0x5,%ebx
8048dcb:   7e e9                jle    8048db6 <phase_6+0x39>
8048dcd:   eb cc                jmp    8048d9b <phase_6+0x1e>
8048dcf:   8d 44 24 10          lea    0x10(%esp),%eax
8048dd3:   8d 5c 24 28          lea    0x28(%esp),%ebx
8048dd7:   b9 07 00 00 00       mov    $0x7,%ecx
8048ddc:   89 ca                mov    %ecx,%edx
8048dde:   2b 10                sub    (%eax),%edx
8048de0:   89 10                mov    %edx,(%eax)
8048de2:   83 c0 04             add    $0x4,%eax
```
https://blog.csdn.net/forest_one

```
8048de7:   75 f3                jne    8048ddc <phase_6+0x5f>
8048de9:   bb 00 00 00 00       mov    $0x0,%ebx
8048dee:   eb 16                jmp    8048e06 <phase_6+0x89>
8048df0:   8b 52 08             mov    0x8(%edx),%edx
8048df3:   83 c0 01             add    $0x1,%eax
8048df6:   39 c8                cmp    %ecx,%eax
8048df8:   75 f6                jne    8048df0 <phase_6+0x73>
8048dfa:   89 54 b4 28          mov    %edx,0x28(%esp,%esi,4)
8048dfe:   83 c3 01             add    $0x1,%ebx
8048e01:   83 fb 06             cmp    $0x6,%ebx
```
https://blog.csdn.net/forest_one

```
8048e06:   89 de                mov    %ebx,%esi
8048e08:   8b 4c 9c 10          mov    0x10(%esp,%ebx,4),%ecx
8048e0c:   b8 01 00 00 00       mov    $0x1,%eax
8048e11:   ba 3c c1 04 08       mov    $0x804c13c,%edx
8048e16:   83 f9 01             cmp    $0x1,%ecx
8048e19:   7f d5                jg     8048df0 <phase_6+0x73>
8048e1b:   eb dd                jmp    8048dfa <phase_6+0x7d>
8048e1d:   8b 5c 24 28          mov    0x28(%esp),%ebx
8048e21:   8b 44 24 2c          mov    0x2c(%esp),%eax
8048e25:   89 43 08             mov    %eax,0x8(%ebx)
8048e28:   8b 54 24 30          mov    0x30(%esp),%edx
8048e2c:   89 50 08             mov    %edx,0x8(%eax)
8048e2f:   8b 44 24 34          mov    0x34(%esp),%eax
8048e33:   89 42 08             mov    %eax,0x8(%edx)
8048e36:   8b 54 24 38          mov    0x38(%esp),%edx
8048e3a:   89 50 08             mov    %edx,0x8(%eax)
8048e3d:   8b 44 24 3c          mov    0x3c(%esp),%eax
8048e41:   89 42 08             mov    %eax,0x8(%edx)
```

```
8048e44:    c7 40 08 00 00 00 00     movl   $0x0,0x8(%eax)
8048e4b:    be 05 00 00 00           mov    $0x5,%esi
8048e50:    8b 43 08                 mov    0x8(%ebx),%eax
8048e53:    8b 10                    mov    (%eax),%edx
8048e55:    39 13                    cmp    %edx,(%ebx)
8048e57:    7d 05                    jge    8048e5e <phase_6+0xe1>
8048e59:    e8 78 02 00 00           call   80490d6 <explode_bomb>
8048e5e:    8b 5b 08                 mov    0x8(%ebx),%ebx
8048e61:    83 ee 01                 sub    $0x1,%esi
8048e64:    75 ea                    jne    8048e50 <phase_6+0xd3>
8048e66:    83 c4 44                 add    $0x44,%esp
8048e69:    5b                       pop    %ebx
8048e6a:    5e                       pop    %esi
8048e6b:    c3                       ret
```

2.6.2 汇编分析

```
Dump of assembler code for function phase_6:
   0x08048d7d <+0>:    push   esi
```

https://blog.csdn.net/forest_one

```
   0x08048d7e <+1>:    push   ebx
   0x08048d7f <+2>:    sub    esp,0x44
   0x08048d82 <+5>:    lea    eax,[esp+0x10]
   0x08048d86 <+9>:    mov    DWORD PTR [esp+0x4],eax
   0x08048d8a <+13>:   mov    eax,DWORD PTR [esp+0x50]
   0x08048d8e <+17>:   mov    DWORD PTR [esp],eax      //输入首地址（传参）
   0x08048d91 <+20>:   call   0x804920b <read_six_numbers>
```

```
[----------------------------------code----------------------------------]
   0x8049239 <read_six_numbers+46>:     mov    DWORD PTR [esp+0x4],0x804a3b7
   0x8049241 <read_six_numbers+54>:     mov    eax,DWORD PTR [esp+0x30]
   0x8049245 <read_six_numbers+58>:     mov    DWORD PTR [esp],eax
=> 0x8049248 <read_six_numbers+61>:     call   0x8048870 <__isoc99_sscanf@plt>
   0x804924d <read_six_numbers+66>:     cmp    eax,0x5
   0x8049250 <read_six_numbers+69>:     jg     0x8049257 <read_six_numbers+76>
   0x8049252 <read_six_numbers+71>:     call   0x80490d6 <explode_bomb>
   0x8049257 <read_six_numbers+76>:     add    esp,0x2c
Guessed arguments:
arg[0]: 0x804c570 ("2 3 5 4 6 1")
arg[1]: 0x804a3b7 ("%d %d %d %d %d %d")
```

```
//输入 6 个整数
=> 0x08048d96 <+25>:   mov    esi,0x0
   0x08048d9b <+30>:   mov    eax,DWORD PTR [esp+esi*4+0x10]
//取输入的第 i 个整数
   0x08048d9f <+34>:   sub    eax,0x1
   0x08048da2 <+37>:   cmp    eax,0x5
   0x08048da5 <+40>:   jbe    0x8048dac <phase_6+47>
 //小于等于跳转，所以 6 个数小于 6
   0x08048da7 <+42>:   call   0x80490d6 <explode_bomb>
   0x08048dac <+47>:   add    esi,0x1
   0x08048daf <+50>:   cmp    esi,0x6              //判断是否循环判断完了 6 个数
   0x08048db2 <+53>:   je     0x8048dcf <phase_6+82>
   0x08048db4 <+55>:   mov    ebx,esi
   0x08048db6 <+57>:   mov    eax,DWORD PTR [esp+ebx*4+0x10]
//输入后五个数依次给 eax
```

https://blog.csdn.net/forest_one

```
   0x08048dba <+61>:  cmp    DWORD PTR [esp+esi*4+0xc],eax
```

```
gdb-peda$ x/1wx 0xbffff0e0
0xbffff0e0:     0x00000006
```

```
//判断第二个数是否为 6
//判断是否有重复数字
   0x08048dbe <+65>:  jne    0x8048dc5 <phase_6+72>
   0x08048dc0 <+67>:  call   0x80490d6 <explode_bomb>
 // DWORD PTR [esp+esi*4+0xc]=eax bomb  （不能重复）
   0x08048dc5 <+72>:  add    ebx,0x1       //记录后五个数字的验证
   0x08048dc8 <+75>:  cmp    ebx,0x5
   0x08048dcb <+78>:  jle    0x8048db6 <phase_6+57>
   0x08048dcd <+80>:  jmp    0x8048d9b <phase_6+30>

   0x08048dcf <+82>:  lea    eax,[esp+0x10]     //输入第一个数字首地址
```

https://blog.csdn.net/forest_one

```
   0x08048dd3 <+86>:  lea    ebx,[esp+0x28]
   0x08048dd7 <+90>:  mov    ecx,0x7
   0x08048ddc <+95>:  mov    edx,ecx
   0x08048dde <+97>:  sub    edx,DWORD PTR [eax]
   0x08048de0 <+99>:  mov    DWORD PTR [eax],edx
   0x08048de2 <+101>: add    eax,0x4
   0x08048de5 <+104>: cmp    eax,ebx
   0x08048de7 <+106>: jne    0x8048ddc <phase_6+95>//数字=7-原数字
   0x08048de9 <+108>: mov    ebx,0x0
   0x08048dee <+113>: jmp    0x8048e06 <phase_6+137>//跳转

   0x08048df0 <+115>: mov    edx,DWORD PTR [edx+0x8]
//从起始地址+上更改数字*8
   0x08048df3 <+118>: add    eax,0x1
   0x08048df6 <+121>: cmp    eax,ecx
   0x08048df8 <+123>: jne    0x8048df0 <phase_6+115>
   0x08048dfa <+125>: mov    DWORD PTR [esp+esi*4+0x28],edx
//从第一个数字对应地址开始放到[esp+esi*4+0x28]
   0x08048dfe <+129>: add    ebx,0x1
   0x08048e01 <+132>: cmp    ebx,0x6
   0x08048e04 <+135>: je     0x8048e1d <phase_6+160>//完成 6 个数
   0x08048e06 <+137>: mov    esi,ebx
   0x08048e08 <+139>: mov    ecx,DWORD PTR [esp+ebx*4+0x10]
//更改后数字给 ecx
   0x08048e0c <+143>: mov    eax,0x1
   0x08048e11 <+148>: mov    edx,0x804c13c
   0x08048e16 <+153>: cmp    ecx,0x1
   0x08048e19 <+156>: jg     0x8048df0 <phase_6+115> //大于跳转
   0x08048e1b <+158>: jmp    0x8048dfa <phase_6+125>
```

https://blog.csdn.net/forest_one

```
   0x08048e1d <+160>: mov    ebx,DWORD PTR [esp+0x28]
  //保存对应地址第一个给 ebx
```

```
0x08048e21 <+164>: mov    eax,DWORD PTR [esp+0x2c]
0x08048e25 <+168>: mov    DWORD PTR [ebx+0x8],eax
0x08048e28 <+171>: mov    edx,DWORD PTR [esp+0x30]
0x08048e2c <+175>: mov    DWORD PTR [eax+0x8],edx
0x08048e2f <+178>: mov    eax,DWORD PTR [esp+0x34]
0x08048e33 <+182>: mov    DWORD PTR [edx+0x8],eax
0x08048e36 <+185>: mov    edx,DWORD PTR [esp+0x38]
0x08048e3a <+189>: mov    DWORD PTR [eax+0x8],edx
0x08048e3d <+192>: mov    eax,DWORD PTR [esp+0x3c]
0x08048e41 <+196>: mov    DWORD PTR [edx+0x8],eax
0x08048e44 <+199>: mov    DWORD PTR [eax+0x8],0x0
0x08048e4b <+206>: mov    esi,0x5
```

https://blog.csdn.net/forest_one

```
 //构造链表
0x08048e50 <+211>: mov    eax,DWORD PTR [ebx+0x8]
0x08048e53 <+214>: mov    edx,DWORD PTR [eax]
0x08048e55 <+216>: cmp    DWORD PTR [ebx],edx  //链表中签一个数大于后一个数
0x08048e57 <+218>: jge    0x8048e5e <phase_6+225>
0x08048e59 <+220>: call   0x80490d6 <explode_bomb>
0x08048e5e <+225>: mov    ebx,DWORD PTR [ebx+0x8]
0x08048e61 <+228>: sub    esi,0x1
0x08048e64 <+231>: jne    0x8048e50 <phase_6+211>
0x08048e66 <+233>: add    esp,0x44
0x08048e69 <+236>: pop    ebx
0x08048e6a <+237>: pop    esi
0x08048e6b <+238>: ret
```

https://blog.csdn.net/forest_one

2.6.3 具体调试

由汇编分析可以得知，需要输入的 6 个数字均不能大于 6 且小于等于 0，并且要求不能有重复数字，即为 1-6 的一个排列。我们以 1 2 3 4 5 6 为例输入：

```
0x804c570 ("1 2 3 4 5 6")
```

在经历过对输入数据的检测之后，我们对输入的 6 个数字进行了操作，即数字=7-原数字，并将操作后的数字存储到从[esp+0x10]开始的 24 字节内存中：

```
0x08048dcf <+82>:    lea    eax,[esp+0x10]
0x08048dd3 <+86>:    lea    ebx,[esp+0x28]
0x08048dd7 <+90>:    mov    ecx,0x7
0x08048ddc <+95>:    mov    edx,ecx
0x08048dde <+97>:    sub    edx,DWORD PTR [eax]
0x08048de0 <+99>:    mov    DWORD PTR [eax],edx
0x08048de2 <+101>:   add    eax,0x4
0x08048de5 <+104>:   cmp    eax,ebx
0x08048de7 <+106>:   jne    0x8048ddc <phase_6+95>
```

```
gdb-peda$ x/6wx 0xbffff0e0
0xbffff0e0:   0x00000006   0x00000005   0x00000004   0x00000003
0xbffff0f0:   0x00000002   0x00000001
```

之后将对应的数字存储到对应的为止，以第一个数字构造为例，进行变换之后第一个数字变为了 6，通过循环不断查找，将内存中的六个数字地址按照输入数字更改后顺序存入 0xbffff0f8 之后的 24 个字节内存中。

```
0x8048df6 <phase_6+121>:    cmp    eax,ecx
0x8048df8 <phase_6+123>:    jne    0x8048df0 <phase_6+115>
0x8048dfa <phase_6+125>:    mov    DWORD PTR [esp+esi*4+0x28],edx
```

https://blog.csdn.net/forest_one

完成操作之后利用上述数据进行构造链表

```
gdb-peda$ x/18wx 0x804c13c
0x804c13c <node1>:      0x000003da      0x00000001      0x00000000      0x000000
bf
0x804c14c <node2+4>:    0x00000002      0x0804c13c      0x00000270      0x000000
03
0x804c15c <node3+8>:    0x0804c148      0x00000323      0x00000004      0x0804c1
54
0x804c16c <node5>:      0x00000185      0x00000005      0x0804c160      0x000003
b8
0x804c17c <node6+4>:    0x00000006      0x0804c16c
```

0x804c130	0x3da
	1
	0x00000000
0x804c148	0x000000bf
	2
	0x804c130
0x804c154	0x270
	3

https://blog.csdn.net/forest_one

	0x804c148
0x804c160	0x323
	4
	0x804c154
0x804c116c	0x185
	5
	0x804c160
0x804c178	0x3b8
	6
	0x804c116c

由于在构造时，用 7-输入数字得到了新的数字数列，所以 1 2 3 4 5 6-》6 5 4 3 2 1 所以构造出的链表是从下往上的。

```
8048e50:   8b 43 08          mov    0x8(%ebx),%eax
8048e53:   8b 10             mov    (%eax),%edx
8048e55:   39 13             cmp    %edx,(%ebx)
8048e57:   7d 05             jge    8048e5e <phase_6+0xe1>
8048e59:   e8 78 02 00 00    call   80490d6 <explode_bomb>
8048e5e:   8b 5b 08          mov    0x8(%ebx),%ebx
8048e61:   83 ee 01          sub    $0x1,%esi
8048e64:   75 ea             jne    8048e50 <phase_6+0xd3>
```

在构造的链表中不断循环，需要前面的数字大于等于后面的数字，所以链表中数字为从大到小排列，又因为原数字进行了变换，所以一开始的序列使得链表从小到大排列，所以最后的结果为 6 1

3 4 2 5。

https://blog.csdn.net/forest_one

验证结果：

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

第六关通过。

2.6.4 解除 bomb 密码

0 1 1 2 3 5

2.7 secret_pahas

使用 gdb 调试

2.7.1 objdump 反汇编

反汇编，得到汇编代码如下：

https://blog.csdn.net/forest_one

```
0804925b <phase_defused>:
 804925b:	81 ec 8c 00 00 00    	sub    $0x8c,%esp
 8049261:	65 a1 14 00 00 00    	mov    %gs:0x14,%eax
 8049267:	89 44 24 7c          	mov    %eax,0x7c(%esp)
 804926b:	31 c0                	xor    %eax,%eax
 804926d:	83 3d cc c3 04 08 06 	cmpl   $0x6,0x804c3cc
 8049274:	75 72                	jne    80492e8 <phase_defused+0x8d>
 8049276:	8d 44 24 2c          	lea    0x2c(%esp),%eax
 804927a:	89 44 24 10          	mov    %eax,0x10(%esp)
 804927e:	8d 44 24 28          	lea    0x28(%esp),%eax
 8049282:	89 44 24 0c          	mov    %eax,0xc(%esp)
 8049286:	8d 44 24 24          	lea    0x24(%esp),%eax
 804928a:	89 44 24 08          	mov    %eax,0x8(%esp)
 804928e:	c7 44 24 04 c9 a3 04 	movl   $0x804a3c9,0x4(%esp)
 8049295:	08 
 8049296:	c7 04 24 d0 c4 04 08 	movl   $0x804c4d0,(%esp)
 804929d:	e8 ce f5 ff ff       	call   8048870 <__isoc99_sscanf@plt>
 80492a2:	83 f8 03             	cmp    $0x3,%eax
 80492a5:	75 35                	jne    80492dc <phase_defused+0x81>
 80492a7:	c7 44 24 04 d2 a3 04 	movl   $0x804a3d2,0x4(%esp)
 80492ae:	08 
 80492af:	8d 44 24 2c          	lea    0x2c(%esp),%eax
 80492b3:	89 04 24             	mov    %eax,(%esp)
 80492b6:	e8 09 fd ff ff       	call   8048fc4 <strings_not_equal>
 80492bb:	85 c0                	test   %eax,%eax
 80492bd:	75 1d                	jne    80492dc <phase_defused+0x81>
```

https://blog.csdn.net/forest_one

```
 80492bf:	c7 04 24 98 a2 04 08 	movl   $0x804a298,(%esp)
```

```
 80492c6:    e8 35 f5 ff ff          call   8048800 <puts@plt>
 80492cb:    c7 04 24 c0 a2 04 08    movl   $0x804a2c0,(%esp)
 80492d2:    e8 29 f5 ff ff          call   8048800 <puts@plt>
 80492d7:    e8 e1 fb ff ff          call   8048ebd <secret_phase>
 80492dc:    c7 04 24 f8 a2 04 08    movl   $0x804a2f8,(%esp)
 80492e3:    e8 18 f5 ff ff          call   8048800 <puts@plt>
 80492e8:    8b 44 24 7c             mov    0x7c(%esp),%eax
 80492ec:    65 33 05 14 00 00 00    xor    %gs:0x14,%eax
 80492f3:    74 05                   je     80492fa <phase_defused+0x9f>
 80492f5:    e8 d6 f4 ff ff          call   80487d0 <__stack_chk_fail@plt>
 80492fa:    81 c4 8c 00 00 00       add    $0x8c,%esp
 8049300:    c3                      ret
```

https://blog.csdn.net/forest_one

2.7.2 汇编分析

```
Dump of assembler code for function phase_defused:
   0x0804925b <+0>:    sub    esp,0x8c
   0x08049261 <+6>:    mov    eax,gs:0x14
   0x08049267 <+12>:   mov    DWORD PTR [esp+0x7c],eax
   0x0804926b <+16>:   xor    eax,eax
   0x0804926d <+18>:   cmp    DWORD PTR ds:0x804c3cc,0x6
   0x08049274 <+25>:   jne    0x80492e8 <phase_defused+141>
   0x08049276 <+27>:   lea    eax,[esp+0x2c]
   0x0804927a <+31>:   mov    DWORD PTR [esp+0x10],eax
   0x0804927e <+35>:   lea    eax,[esp+0x28]
   0x08049282 <+39>:   mov    DWORD PTR [esp+0xc],eax
   0x08049286 <+43>:   lea    eax,[esp+0x24]
   0x0804928a <+47>:   mov    DWORD PTR [esp+0x8],eax
   0x0804928e <+51>:   mov    DWORD PTR [esp+0x4],0x804a3c9
   0x08049296 <+59>:   mov    DWORD PTR [esp],0x804c4d0
```

```
[-------------------------------------code-------------------------------------]
   0x804928a <phase_defused+47>:        mov    DWORD PTR [esp+0x8],eax
   0x804928e <phase_defused+51>:        mov    DWORD PTR [esp+0x4],0x804a3c9
   0x8049296 <phase_defused+59>:        mov    DWORD PTR [esp],0x804c4d0
=> 0x804929d <phase_defused+66>:        call   0x8048870 <__isoc99_sscanf@plt>
   0x80492a2 <phase_defused+71>:        cmp    eax,0x3
   0x80492a5 <phase_defused+74>:        jne    0x80492dc <phase_defused+129>
   0x80492a7 <phase_defused+76>:        mov    DWORD PTR [esp+0x4],0x804a3d2
   0x80492af <phase_defused+84>:        lea    eax,[esp+0x2c]
Guessed arguments:
arg[0]: 0x804c4d0 --> 0x342032 ('2 4')
arg[1]: 0x804a3c9 ("%d %d %s")
arg[2]: 0xbffff0b4 --> 0xbffff0ec --> 0x3
arg[3]: 0xbffff0b8 --> 0xbffff0f0 --> 0x5
arg[4]: 0xbffff0bc --> 0xbffff0f4 --> 0x2
```

```
=> 0x0804929d <+66>:   call   0x8048870 <__isoc99_sscanf@plt>
   0x080492a2 <+71>:   cmp    eax,0x3
   0x080492a5 <+74>:   jne    0x80492dc <phase_defused+129>
```

https://blog.csdn.net/forest_one

```
   0x080492a7 <+76>:   mov    DWORD PTR [esp+0x4],0x804a3d2
```

```
gdb-peda$ x/s 0x804a3d2
0x804a3d2:      "DrEvil"
gdb-peda$
```

```
0x080492af <+84>:  lea    eax,[esp+0x2c]
0x080492b3 <+88>:  mov    DWORD PTR [esp],eax
0x080492b6 <+91>:  call   0x8048fc4 <strings_not_equal>
0x080492bb <+96>:  test   eax,eax
0x080492bd <+98>:  jne    0x80492dc <phase_defused+129>
0x080492bf <+100>: mov    DWORD PTR [esp],0x804a298
0x080492c6 <+107>: call   0x8048800 <puts@plt>
0x080492cb <+112>: mov    DWORD PTR [esp],0x804a2c0
0x080492d2 <+119>: call   0x8048800 <puts@plt>
0x080492d7 <+124>: call   0x8048ebd <secret_phase>
```
https://blog.csdn.net/forest_one

2.7.3 具体调试

由以上汇编分析可知，进入隐藏炸弹满足的条件是在第四关的密码“2 4”后面加上一个字符串“DrEvil”即可进入隐藏关卡。我们对隐藏关 secret_phase 进行如下分析：

```
=> 0x08048ebd <+0>:   push   ebx
   0x08048ebe <+1>:   sub    esp,0x18
   0x08048ec1 <+4>:   call   0x80490fd <read_line>
   0x08048ec6 <+9>:   mov    DWORD PTR [esp+0x8],0xa
   0x08048ece <+17>:  mov    DWORD PTR [esp+0x4],0x0
   0x08048ed6 <+25>:  mov    DWORD PTR [esp],eax
   0x08048ed9 <+28>:  call   0x80488e0 <strtol@plt>
   0x08048ede <+33>:  mov    ebx,eax
   0x08048ee0 <+35>:  lea    eax,[eax-0x1]     //输入数字减一
   0x08048ee3 <+38>:  cmp    eax,0x3e8     //比较
   0x08048ee8 <+43>:  jbe    0x8048eef <secret_phase+50>
             //小于等于跳转，否则 bomb
   0x08048eea <+45>:  call   0x80490d6 <explode_bomb>
   0x08048eef <+50>:  mov    DWORD PTR [esp+0x4],ebx     //传参输入的数
   0x08048ef3 <+54>:  mov    DWORD PTR [esp],0x804c088   //传参地址
   0x08048efa <+61>:  call   0x8048e6c <fun7>
   0x08048eff <+66>:  cmp    eax,0x7
   0x08048f02 <+69>:  je     0x8048f09 <secret_phase+76>
   0x08048f04 <+71>:  call   0x80490d6 <explode_bomb>
   0x08048f09 <+76>:  mov    DWORD PTR [esp],0x804a1d8
   0x08048f10 <+83>:  call   0x8048800 <puts@plt>
   0x08048f15 <+88>:  call   0x804925b <phase_defused>
   0x08048f1a <+93>:  add    esp,0x18
   0x08048f1d <+96>:  pop    ebx
   0x08048f1e <+97>:  ret
```

我们注意到需要输入一个数字，这个数字减一与 0x3e8 进行比较，如果大于则 bomb，所以输入的数字小于等于 0x3e9。进入函数 fun7，返回值要为 7，否则则会 bomb。我们进入 fun7 进行分析：

```
Dump of assembler code for function fun7:
=> 0x08048e6c <+0>:   push   ebx
   0x08048e6d <+1>:   sub    esp,0x18
   0x08048e70 <+4>:   mov    edx,DWORD PTR [esp+0x20]
   0x08048e74 <+8>:   mov    eax,DWORD PTR [esp+0x24]
```

```
0x08048e74 <+8>:   mov    ecx, DWORD PTR [esp+0x24]
0x08048e78 <+12>:  test   edx, edx      //检测是否为 0
0x08048e7a <+14>:  je     0x8048eb3 <fun7+71>//为 0 返回负一
0x08048e7c <+16>:  mov    ebx, DWORD PTR [edx]
0x08048e7e <+18>:  cmp    ebx, ecx              //比较给定地址数和输入
0x08048e80 <+20>:  jle    0x8048e95 <fun7+41> //小于等于跳转 41，大于继续
0x08048e82 <+22>:  mov    DWORD PTR [esp+0x4], ecx
0x08048e86 <+26>:  mov    eax, DWORD PTR [edx+0x4]    //传参地址+8
0x08048e89 <+29>:  mov    DWORD PTR [esp], eax        //传参
0x08048e8c <+32>:  call   0x8048e6c <fun7>
0x08048e91 <+37>:  add    eax, eax                     //return 2*fun7
0x08048e93 <+39>:  jmp    0x8048eb8 <fun7+76>
0x08048e95 <+41>:  mov    eax, 0x0                    //赋值 0
0x08048e9a <+46>:  cmp    ebx, ecx              //比较给定地址数和输入
0x08048e9c <+48>:  je     0x8048eb8 <fun7+76>  //相等结束
0x08048e9e <+50>:  mov    DWORD PTR [esp+0x4], ecx  //传参 输入的数
0x08048ea2 <+54>:  mov    eax, DWORD PTR [edx+0x8]
0x08048ea5 <+57>:  mov    DWORD PTR [esp], eax        //传参地址+8
0x08048ea8 <+60>:  call   0x8048e6c <fun7>
0x08048ead <+65>:  lea    eax, [eax+eax*1+0x1]        //return 2*fun4+1
0x08048eb1 <+69>:  jmp    0x8048eb8 <fun7+76>
0x08048eb3 <+71>:  mov    eax, 0xffffffff  //返回-1
0x08048eb8 <+76>:  add    esp, 0x18
0x08048ebb <+79>:  pop    ebx
0x08048ebc <+80>:  ret
```

由以上分析可知 fun7 同样为一个递归函数，且递归函数具有两个参数，我们同样根据汇编代码可以写出 C 代码；

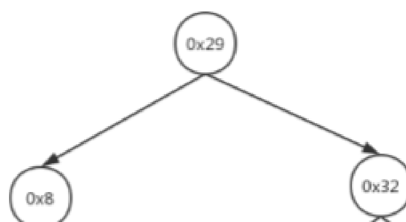
https://blog.csdn.net/forest_one

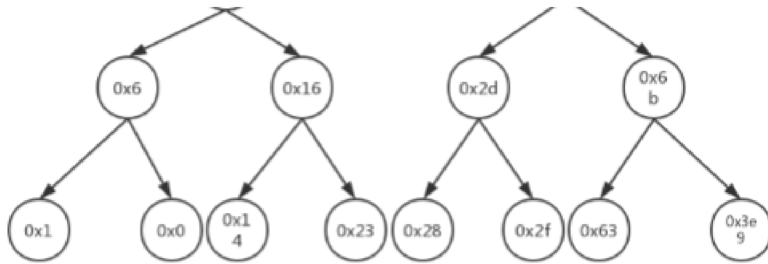
[*] bomb_fun4.cpp

```
1  int fun7(_DWORD *a1, int a2)
2  {
3    int result; // eax
4    if ( !a1 )
5      return -1;
6    if ( *a1 > a2 )
7      return 2 * fun7(*(a1+1), a2);
8    result = 0;
9    if ( *a1 != a2 )
10     result = 2 * fun7(*(a1+2), a2) + 1;
11   return result;
12 }
```

https://blog.csdn.net/forest_one

根据以上内容进行分析，我们可以得知是一个二叉树结构，对内存中的数据查看可以构造以下树：





函数返回值应为 7，根据递归式子 7——》3——》1——》0，所以每次进行递归的条件都是输入的数值大于当前结点的数值，即一直向右子树进行查找，进行四次查找之后，返回 0，依次完成递归，由此可以得出我们需要输入的数字为 0x3e9，其实这个答案在开始时已经给了我们一些提示，因为当输入之后，输入的数字先进行了-1 与 0x3e8 进行比较，即 0x3e9 为符合要求的最大数字。

验证结果：

```
Curses, you've found the secret phase!
But finding it and solving it are quite different...
1001
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```

隐藏关卡通过。

https://blog.csdn.net/forest_one