二进制拆弹实验报告

161220049 黄奕诚

为更加清晰直观地陈述解题思路，我将每个阶段的汇编代码复制下来，写好注释，并在最后做必要的解析。

第一阶段：

```
0804898f <phase_1>:
 804898f:    55                 push   %ebp
 8048990:    89 e5                 mov    %esp,%ebp
 8048992:    83 ec 08              sub    $0x8,%esp              //栈顶减小 8 字节，腾出空间
 8048995:    83 ec 08              sub    $0x8,%esp              //栈顶减小 8 字节，腾出空间
 8048998:    68 60 90 04 08        push   $0x8049060             //将 0x8049060 中的数压入栈中
 804899d:    ff 75 08              pushl  0x8(%ebp)              //将输入的字符串压入栈中
 80489a0:    e8 9f ff ff ff        call   8048944 <strings_not_equal>  //调用函数比较字符串
 80489a5:    83 c4 10              add    $0x10,%esp             //栈顶减小 16 字节，缩小空间
 80489a8:    85 c0                 test   %eax,%eax              //即 函数返回值
 80489aa:    74 05                 je     80489b1 <phase_1+0x22>    //若返回值==0，结束
 80489ac:    e8 1f ff ff ff        call   80488d0 <explode_bomb>    //若==1，爆炸
 80489b1:    90                 nop
 80489b2:    c9                 leave
 80489b3:    c3                 ret
```

因此输入的字符串只要与 0x8049060 地址储存的值相同即可，通过 GDB 断点调试，可以得到这个字符串为 “vdoharoezqecr”，第一个问题告破。

第二阶段：

```
080489b4 <phase_2>:
 80489b4:    55                 push   %ebp
 80489b5:    89 e5                 mov    %esp,%ebp
 80489b7:    83 ec 48              sub    $0x48,%esp
 80489ba:    8b 45 08              mov    0x8(%ebp),%eax         //将输入字符串加载到%eax
 80489bd:    89 45 c4              mov    %eax,-0x3c(%ebp)       //将输入字符串保存到-60(%ebp)
 80489c0:    65 a1 14 00 00 00     mov    %gs:0x14,%eax          //金丝雀值
 80489c6:    89 45 f4              mov    %eax,-0xc(%ebp)
 80489c9:    31 c0                 xor    %eax,%eax
 80489cb:    c7 45 d8 00 00 00 00  movl   $0x0,-0x28(%ebp)       //将 0 保存到-40(%ebp)
 80489d2:    83 ec 08              sub    $0x8,%esp
 80489d5:    8d 45 dc              lea    -0x24(%ebp),%eax       //将-36(%ebp)赋给%eax
 80489d8:    50                 push   %eax                      //将-36(%ebp)的值压入栈
 80489d9:    ff 75 c4              pushl  -0x3c(%ebp)            //将-60(%ebp)的值压入栈
 80489dc:	e8 0f ff ff ff       	call   80488f0 <read_six_numbers>  //判断输入是否大于 6 个
 80489e1:	83 c4 10             	add    $0x10,%esp
 80489e4:	c7 45 d4 00 00 00 00 	movl   $0x0,-0x2c(%ebp)      //将 0 保存到-44(%ebp)
 80489eb:	eb 2e                	jmp    8048a1b <phase_2+0x67>      //跳转到 8048a1b
 80489ed:	8b 45 d4             	mov    -0x2c(%ebp),%eax      //将-44(%ebp)的值加载到%eax
 80489f0:	8b 44 85 dc          	mov    -0x24(%ebp,%eax,4),%eax//-36(%ebp)+4%eax 赋
给%eax
 80489f4:	8d 50 4e             	lea    0x4e(%eax),%edx        //%eax+78 赋给%edx
 80489f7:	8b 45 d4             	mov    -0x2c(%ebp),%eax      //将-44(%ebp)的值加载到%eax
 80489fa:	83 c0 03             	add    $0x3,%eax               //%eax 的值加 3
 80489fd:	8b 44 85 dc          	mov    -0x24(%ebp,%eax,4),%eax    //-36(%ebp)+4%eax 赋给
%eax
 8048a01:	83 c0 2d             	add    $0x2d,%eax              //%eax 的值加 45
 8048a04:	39 c2                	cmp    %eax,%edx               //作差比较(%edx)-(%eax)
 8048a06:	74 05                	je     8048a0d <phase_2+0x59>       //若相等，继续
 8048a08:	e8 c3 fe ff ff       	call   80488d0 <explode_bomb>      //若不等，爆炸
 8048a0d:	8b 45 d4             	mov    -0x2c(%ebp),%eax      //将-44(%ebp)的值加载到%eax
 8048a10:	8b 44 85 dc          	mov    -0x24(%ebp,%eax,4),%eax    //-36(%ebp)+4%eax 赋给
%eax
 8048a14:	01 45 d8             	add    %eax,-0x28(%ebp)      //-40(%ebp)加%eax 的值
 8048a17:	83 45 d4 01          	addl   $0x1,-0x2c(%ebp)      //-44(%ebp)加 1
 8048a1b:	83 7d d4 02          	cmpl   $0x2,-0x2c(%ebp)      //作差：-44(%ebp)-2
 8048a1f:	7e cc                	jle    80489ed <phase_2+0x39> //若结果小于等于 0，跳回
 8048a21:	83 7d d8 00          	cmpl   $0x0,-0x28(%ebp)      //作差：-40(%ebp)-2
 8048a25:	75 05                	jne    8048a2c <phase_2+0x78> //若结果不等于 0，结束
 8048a27:	e8 a4 fe ff ff       	call   80488d0 <explode_bomb> //若结果等于 0，爆炸
 8048a2c:	90                   	nop
 8048a2d:	8b 45 f4             	mov    -0xc(%ebp),%eax
 8048a30:	65 33 05 14 00 00 00 	xor    %gs:0x14,%eax
 8048a37:	74 05                	je     8048a3e <phase_2+0x8a>
 8048a39:	e8 a2 fa ff ff       	call   80484e0 <__stack_chk_fail@plt>
 8048a3e:	c9                   	leave
 8048a3f:	c3                   	ret
```

核心：分析循环中各个值的变化：首先-44(%ebp)保存的值由 0 递增到 2，当跳出循环后要保证-40(%ebp)保存的值不等于 2；并且，在循环过程中要保证 A[-44(%ebp)]+78==A[-44(%ebp)+3]+45，随后-40（%ebp)+=A[-44(%ebp)]，因此只要保证 A[3]-A[0]=A[4]-A[1]=A[5]-A[2]=33 即可，我输入了 1 2 3 34 35 36，成功。

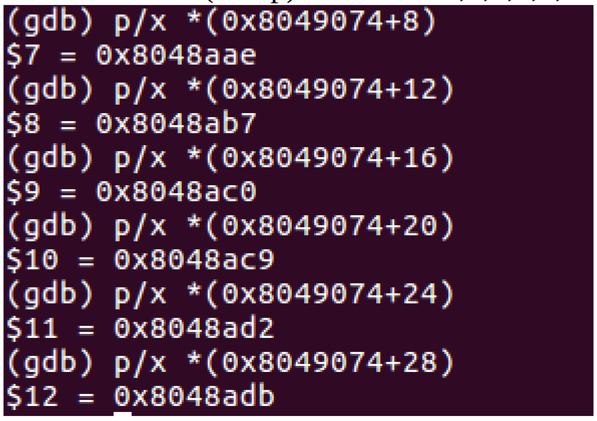
第三阶段：

关键：

```
 8048a9a:	ff e0                	jmp    *%eax   //跳至%eax 的地址
 8048a9c:	c7 45 ec 0e 00 00 00 	movl   $0xe,-0x14(%ebp)      //将 14 保存到-20(%ebp)
 8048aa3:	eb 44                	jmp    8048ae9 <phase_3+0xa9>
 8048aa5:	c7 45 ec 48 00 00 00 	movl   $0x48,-0x14(%ebp)    //将 72 保存到-20(%ebp)
 8048aac:	eb 3b                	jmp    8048ae9 <phase_3+0xa9>
```

```
8048aae:     c7 45 ec 55 00 00 00  movl   $0x55,-0x14(%ebp)    //将 85 保存到-20(%ebp)
8048ab5:     eb 32                 jmp    8048ae9 <phase_3+0xa9>
8048ab7:     c7 45 ec 0c 00 00 00  movl   $0xc,-0x14(%ebp)     //将 12 保存到-20(%ebp)
8048abe:     eb 29                 jmp    8048ae9 <phase_3+0xa9>
8048ac0:     c7 45 ec 63 00 00 00  movl   $0x63,-0x14(%ebp)    //将 99 保存到-20(%ebp)
8048ac7:     eb 20                 jmp    8048ae9 <phase_3+0xa9>
8048ac9:     c7 45 ec 29 00 00 00  movl   $0x29,-0x14(%ebp)    //将 41 保存到-20(%ebp)
8048ad0:     eb 17                 jmp    8048ae9 <phase_3+0xa9>
8048ad2:     c7 45 ec 21 00 00 00  movl   $0x21,-0x14(%ebp)    //将 33 保存到-20(%ebp)
8048ad9:     eb 0e                 jmp    8048ae9 <phase_3+0xa9>
8048adb:     c7 45 ec 3f 00 00 00  movl   $0x3f,-0x14(%ebp)    //将 63 保存到-20(%ebp)
8048ae2:     eb 05                 jmp    8048ae9 <phase_3+0xa9>
8048ae4:     e8 e7 fd ff ff        call   80488d0 <explode_bomb> //爆炸
8048ae9:     8b 45 e8              mov    -0x18(%ebp),%eax     //将-24(%ebp)的值加载到%eax
8048aec:     39 45 ec              cmp    %eax,-0x14(%ebp)     //比较-20(%ebp)与%eax 的值
8048aef:     74 05                 je     8048af6 <phase_3+0xb6>        //若相等，成功
8048af1:     e8 da fd ff ff        call   80488d0 <explode_bomb>        //若不等，爆炸
8048af6:     90                    nop
8048af7:     8b 45 f4              mov    -0xc(%ebp),%eax
8048afa:     65 33 05 14 00 00 00  xor    %gs:0x14,%eax
8048b01:     74 05                 je     8048b08 <phase_3+0xc8>
8048b03:     e8 d8 f9 ff ff        call   80484e0 <__stack_chk_fail@plt>
8048b08:     c9                    leave
8048b09:     c3                    ret
```

核心：由 switch 条件的那几行不难发现，-28(%ebp)的值是有限定的，只能跳转到非 default 的那几行，否则就会爆炸。根据地址的运算，可得出-28(%ebp)的值只能为 2,3,4,5,6,7.通过 gdb 调试：


```
(gdb) p/x *(0x8049074+8)
$7 = 0x8048aae
(gdb) p/x *(0x8049074+12)
$8 = 0x8048ab7
(gdb) p/x *(0x8049074+16)
$9 = 0x8048ac0
(gdb) p/x *(0x8049074+20)
$10 = 0x8048ac9
(gdb) p/x *(0x8049074+24)
$11 = 0x8048ad2
(gdb) p/x *(0x8049074+28)
$12 = 0x8048adb
```

可以确定-20(%ebp)可以得到的值为 85、 12、 99、 41、 33、 63
因此-24(%ebp)的值只能是这么几个，这也是输入的第二个元素。输入的第一个元素则为对应的序号
因此可输入的答案有 6 组，分别是：
2 85
3 12
4 99
5 41
6 33
7 63

第四阶段：

```
08048b3a <phase_4>:
 8048b3a:       55                      push   %ebp
 8048b3b:       89 e5                   mov    %esp,%ebp
 8048b3d:       83 ec 28                sub    $0x28,%esp
 8048b40:       8b 45 08                mov    0x8(%ebp),%eax  //将输入值加载到%eax
 8048b43:       89 45 e4                mov    %eax,-0x1c(%ebp)  //将输入值保存到-28(%ebp)
 8048b46:       65 a1 14 00 00 00       mov    %gs:0x14,%eax
 8048b4c:       89 45 f4                mov    %eax,-0xc(%ebp)
 8048b4f:       31 c0                   xor    %eax,%eax
 8048b51:       68 a0 b2 04 08          push   $0x804b2a0  //将地址 0x804b2a0 保存的数压入
 8048b56:       8d 45 e8                lea    -0x18(%ebp),%eax  //将-24(%ebp)赋给%eax
 8048b59:       50                      push   %eax  //压入%eax
 8048b5a:       68 94 90 04 08          push   $0x8049094  //将地址 0x8049094 保存的数压入
 8048b5f:       ff 75 e4                pushl  -0x1c(%ebp)  //将输入值压入
 8048b62:       e8 d9 f9 ff ff          call   8048540 <__isoc99_sscanf@plt>
 8048b67:       83 c4 10                add    $0x10,%esp
 8048b6a:       89 45 ec                mov    %eax,-0x14(%ebp)
 8048b6d:       83 7d ec 00             cmpl   $0x0,-0x14(%ebp) //判断输入的个数是否大于 0
 8048b71:       7e 07                   jle    8048b7a <phase_4+0x40>
 8048b73:       8b 45 e8                mov    -0x18(%ebp),%eax  //将-24(%ebp)的值加载到%eax
 8048b76:       85 c0                   test   %eax,%eax  //检验-24(%ebp)的值
 8048b78:       7f 05                   jg     8048b7f <phase_4+0x45>  //若大于 0，则继续
 8048b7a:       e8 51 fd ff ff          call   80488d0 <explode_bomb>
 8048b7f:       8b 45 e8                mov    -0x18(%ebp),%eax  //将-24(%ebp)的值加载到%eax
 8048b82:       83 ec 0c                sub    $0xc,%esp
 8048b85:       50                      push   %eax  //将-24(%ebp)的值压入参数，执行 func4
 8048b86:       e8 7f ff ff ff          call   8048b0a <func4>
 8048b8b:       83 c4 10                add    $0x10,%esp
 8048b8e:       89 45 f0                mov    %eax,-0x10(%ebp)     //将返回值赋给-16(%ebp)
 8048b91:       83 7d f0 7d             cmpl   $0x7d,-0x10(%ebp)     //比较返回值与 125
 8048b95:       74 05                   je     8048b9c <phase_4+0x62>  //若等于 125，成功，否则爆
 8048b97:       e8 34 fd ff ff          call   80488d0 <explode_bomb>
 8048b9c:       90                      nop
 8048b9d:       8b 45 f4                mov    -0xc(%ebp),%eax
 8048ba0:       65 33 05 14 00 00 00    xor    %gs:0x14,%eax
 8048ba7:       74 05                   je     8048bae <phase_4+0x74>
 8048ba9:       e8 32 f9 ff ff          call   80484e0 <__stack_chk_fail@plt>
 8048bae:       c9                      leave
 8048baf:       c3                      ret

08048b0a <func4>:
 8048b0a:       55                      push   %ebp
 8048b0b:       89 e5                   mov    %esp,%ebp
 8048b0d:       83 ec 08                sub    $0x8,%esp
 8048b10:       83 7d 08 00             cmpl   $0x0,0x8(%ebp)         //将输入值与 0 比较
 8048b14:       7f 07                   jg     8048b1d <func4+0x13>  //若大于 0，跳至 1d
 8048b16:       b8 01 00 00 00          mov    $0x1,%eax  //若小于等于 0，返回 1
 8048b1b:       eb 1b                   jmp    8048b38 <func4+0x2e>  //直接结束
 8048b1d:       8b 45 08                mov    0x8(%ebp),%eax  //将输入值加载到%eax
 8048b20:       83 e8 01                sub    $0x1,%eax  //%eax 的值减 1
 8048b23:       83 ec 0c                sub    $0xc,%esp
 8048b26:       50                      push   %eax
 8048b27:       e8 de ff ff ff          call   8048b0a <func4>  //递归
 8048b2c:       83 c4 10                add    $0x10,%esp
 8048b2f:       89 c2                   mov    %eax,%edx
 8048b31:       89 d0                   mov    %edx,%eax
 8048b33:       c1 e0 02                shl    $0x2,%eax  //%eax 左移 2 位
 8048b36:       01 d0                   add    %edx,%eax  //%eax+=(4*%eax)
 8048b38:       c9                      leave
 8048b39:       c3                      ret
```

核心：递归，逆推。从返回值为 125 逆向推导：

```
func4(int x)
{
        if (x<=0)
                return 1;
        return 5*func4(x-1);
}
```

125=func4(x)=5func4(x-1)=25func4(x-2) 而 func4(1)=5，因此 x=3

所以输入值为 3

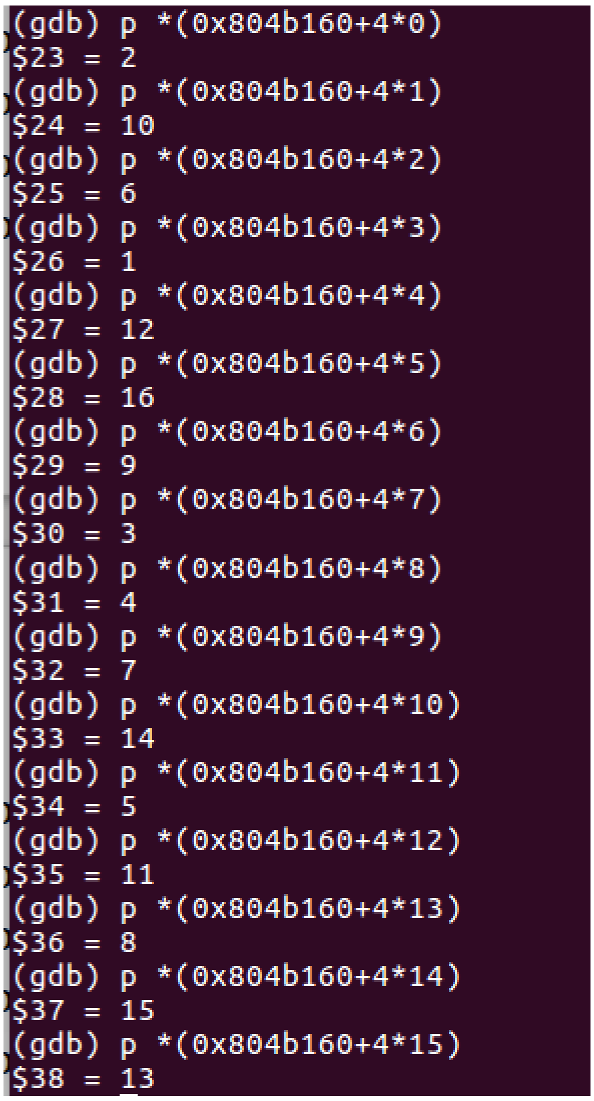
第五阶段

```
08048bb0 <phase_5>:
 8048bb0:       55                      push   %ebp
 8048bb1:       89 e5                   mov    %esp,%ebp
 8048bb3:       83 ec 18                sub    $0x18,%esp
 8048bb6:       83 ec 0c                sub    $0xc,%esp
 8048bb9:       ff 75 08                pushl  0x8(%ebp)  //将输入压入栈，执行函数 string_length
 8048bbc:       e8 ac fd ff ff          call   804896d <string_length>
 8048bc1:       83 c4 10                add    $0x10,%esp
 8048bc4:       89 45 f4                mov    %eax,-0xc(%ebp)  //将返回值存入-12(%ebp)
 8048bc7:       83 7d f4 06             cmpl   $0x6,-0xc(%ebp)  //将返回值与 6 比较
 8048bcb:       74 05                   je     8048bd2 <phase_5+0x22>  //若相等，则继续，否则爆炸
 8048bcd:       e8 fe fc ff ff          call   80488d0 <explode_bomb>
 8048bd2:       c7 45 f0 00 00 00 00    movl   $0x0,-0x10(%ebp)  //将 0 保存到-16(%ebp)
 8048bd9:       c7 45 ec 00 00 00 00    movl   $0x0,-0x14(%ebp)  //将 0 保存到-20(%ebp)
```

```
8048be0:	eb 1f                	jmp    8048c01 <phase_5+0x51> //直接跳转至 c01
8048be2:	8b 55 ec             	mov    -0x14(%ebp),%edx  //将-20(%ebp)的值加载到%edx
8048be5:	8b 45 08             	mov    0x8(%ebp),%eax  //将返回值加载到%eax
8048be8:	01 d0                	add    %edx,%eax  //%eax 加上-20(%ebp)的值
8048bea:	0f b6 00             	movzbl (%eax),%eax  //将此时%eax 指向的值赋给%eax
8048bed:	0f be c0             	movsbl %al,%eax
8048bf0:	83 e0 0f             	and    $0xf,%eax  //取%eax 的最后四位
8048bf3:	8b 04 85 60 b1 04 08 	mov    0x804b160(,%eax,4),%eax  //变址
8048bfa:	01 45 f0             	add    %eax,-0x10(%ebp)  //-16(%ebp)加上%eax 的值
8048bfd:	83 45 ec 01          	addl   $0x1,-0x14(%ebp)  //-20(%ebp)加上 1
8048c01:	83 7d ec 05          	cmpl   $0x5,-0x14(%ebp)  //比较-20(%ebp)的值与 5
8048c05:	7e db                	jle    8048be2 <phase_5+0x32>  //若小于等于，则返回循环
8048c07:	83 7d f0 26          	cmpl   $0x26,-0x10(%ebp)  //比较-16(%ebp)的值和 38
8048c0b:	74 05                	je     8048c12 <phase_5+0x62>  //若相等，则成功
8048c0d:	e8 be fc ff ff       	call   80488d0 <explode_bomb>
8048c12:	90                   	nop
8048c13:	c9                   	leave
8048c14:	c3                   	ret
```

核心：输入的字符串长度必须为 6，其次要经过 6 次循环（-20(%ebp)的值从 0 到 5），%eax 的位置由%eax 每次循环加一。注意到加粗一行%eax 取了最后四位，即从 0000-1111，即 0-15，进行变址。通过 GDB 来寻址，穷举所有情况：

又由 6 个数相加得 38 知，3,4,5,6,7,13 符合题意，转换到输入，应该是 7,8,11,2,9,15
查询 ASCII 码，发现 ghkbio 符合题意



```
(gdb) p *(0x804b160+4*0)
$23 = 2
(gdb) p *(0x804b160+4*1)
$24 = 10
(gdb) p *(0x804b160+4*2)
$25 = 6
(gdb) p *(0x804b160+4*3)
$26 = 1
(gdb) p *(0x804b160+4*4)
$27 = 12
(gdb) p *(0x804b160+4*5)
$28 = 16
(gdb) p *(0x804b160+4*6)
$29 = 9
(gdb) p *(0x804b160+4*7)
$30 = 3
(gdb) p *(0x804b160+4*8)
$31 = 4
(gdb) p *(0x804b160+4*9)
$32 = 7
(gdb) p *(0x804b160+4*10)
$33 = 14
(gdb) p *(0x804b160+4*11)
$34 = 5
(gdb) p *(0x804b160+4*12)
$35 = 11
(gdb) p *(0x804b160+4*13)
$36 = 8
(gdb) p *(0x804b160+4*14)
$37 = 15
(gdb) p *(0x804b160+4*15)
$38 = 13
```

第六阶段

第一片段：

```
8048c15:    55                    push   %ebp
8048c16:    89 e5                 mov    %esp,%ebp
8048c18:    83 ec 68              sub    $0x68,%esp
8048c1b:    8b 45 08              mov    0x8(%ebp),%eax  //将输入加载到%eax
8048c1e:    89 45 a4              mov    %eax,-0x5c(%ebp)  //将输入保存到-92(%ebp)
8048c21:    65 a1 14 00 00 00     mov    %gs:0x14,%eax
8048c27:    89 45 f4              mov    %eax,-0xc(%ebp)
8048c2a:    31 c0                 xor    %eax,%eax
8048c2c:    c7 45 c0 a0 b0 04 08  movl   $0x804b0a0,-0x40(%ebp) //将数保存到-64(%ebp)
8048c33:    83 ec 08              sub    $0x8,%esp
8048c36:    8d 45 c4              lea    -0x3c(%ebp),%eax  //将-60(%ebp)的值加载到%eax
8048c39:    50                    push   %eax  //将-60(%ebp)压入函数
8048c3a:    ff 75 a4              pushl  -0x5c(%ebp)  //将输入压入函数
8048c3d:    e8 ae fc ff ff        call   80488f0 <read_six_numbers> //读取六个数字
8048c42:    83 c4 10              add    $0x10,%esp
8048c45:    c7 45 b8 00 00 00 00  movl   $0x0,-0x48(%ebp) //将 0 赋给-72(%ebp)
8048c4c:    eb 4c                 jmp    8048c9a <phase_6+0x85>  //直接跳至 9a
8048c4e:    8b 45 b8              mov    -0x48(%ebp),%eax  //将-72(%ebp)的值加载到%eax
8048c51:    8b 44 85 c4           mov    -0x3c(%ebp,%eax,4),%eax  //%eax 往后移-72(%ebp)位
8048c55:    85 c0                 test   %eax,%eax  //检验%eax
8048c57:    7e 0c                 jle    8048c65 <phase_6+0x50>  //若小于等于 0，爆炸；要>0
8048c59:    8b 45 b8              mov    -0x48(%ebp),%eax  //将-72(%ebp)的值加载到%eax
8048c5c:    8b 44 85 c4           mov    -0x3c(%ebp,%eax,4),%eax  //%eax 往后移-72(%ebp)位
8048c60:    83 f8 06              cmp    $0x6,%eax  //比较%eax 和 6
8048c63:    7e 05                 jle    8048c6a <phase_6+0x55>  //若大于 6，爆炸；要<=0
8048c65:    e8 66 fc ff ff        call   80488d0 <explode_bomb>
8048c6a:    8b 45 b8              mov    -0x48(%ebp),%eax  //将-72(%ebp)的值加载到%eax
8048c6d:    83 c0 01              add    $0x1,%eax  //%eax 加 1
8048c70:    89 45 bc              mov    %eax,-0x44(%ebp)  //将%eax 的值保存到-68(%ebp)
8048c73:    eb 1b                 jmp    8048c90 <phase_6+0x7b>  //直接跳至 90
8048c75:    8b 45 b8              mov    -0x48(%ebp),%eax  //将-72(%ebp)的值加载到%eax
8048c78:    8b 54 85 c4           mov    -0x3c(%ebp,%eax,4),%edx  //赋值%edx
8048c7c:    8b 45 bc              mov    -0x44(%ebp),%eax  //将-68(%ebp)的值加载到%eax
8048c7f:    8b 44 85 c4           mov    -0x3c(%ebp,%eax,4),%eax  //%eax 往后移-72(%ebp)位
8048c83:    39 c2                 cmp    %eax,%edx  //比较%edx 和%eax
8048c85:    75 05                 jne    8048c8c <phase_6+0x77>  //若不等于 0，继续
8048c87:    e8 44 fc ff ff        call   80488d0 <explode_bomb>
8048c8c:    83 45 bc 01           addl   $0x1,-0x44(%ebp)  //-68(%ebp)的值加 1
8048c90:    83 7d bc 05           cmpl   $0x5,-0x44(%ebp)  //比较-68(%ebp)的值与 5
```

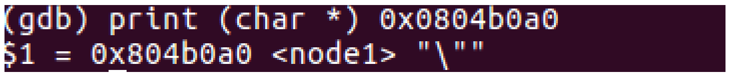
```
8048c94:    7e df          jle    8048c75 <phase_6+0x60>  //若小于等于 5，返回内循环
8048c96:    83 45 b8 01    addl   $0x1,-0x48(%ebp)  //-72(%ebp)的值加 1
8048c9a:    83 7d b8 05    cmpl   $0x5,-0x48(%ebp)  //比较-72(%ebp)与 5
8048c9e:    7e ae          jle    8048c4e <phase_6+0x39>  //若小于等于，则返回外循环
```

可以翻译为 C 语言：

```
phase_6(int * A)
{
        int i,j,k,*B;
        for (i=0;i<=5;i++)
        {
                if (A[i] <= 0 || A[i] >6)
                        explode_bomb();
                else
                {
                        for (j=i+1 ;j<=5;j++)
                        {
                                if ( A[i] == A[j] )
                                        explode_bomb();
                        }
                }
        }
        /* 由这个片段可以看出，每个元素要在[1,6]之间，且互不相等，故为 1,2,3,4,5,6 的一个排列*/
```

第二片段中，

随后我观察了 0x0804b0a0 地址的值：



```
(gdb) print (char *) 0x0804b0a0
$1 = 0x804b0a0 <node1> "\""
```

发现了 node1，说明这是一个特殊数据结构，又从后面的 mov 0x8(%eax),%eax 可以看出是个链表的结构，这是 next 操作。

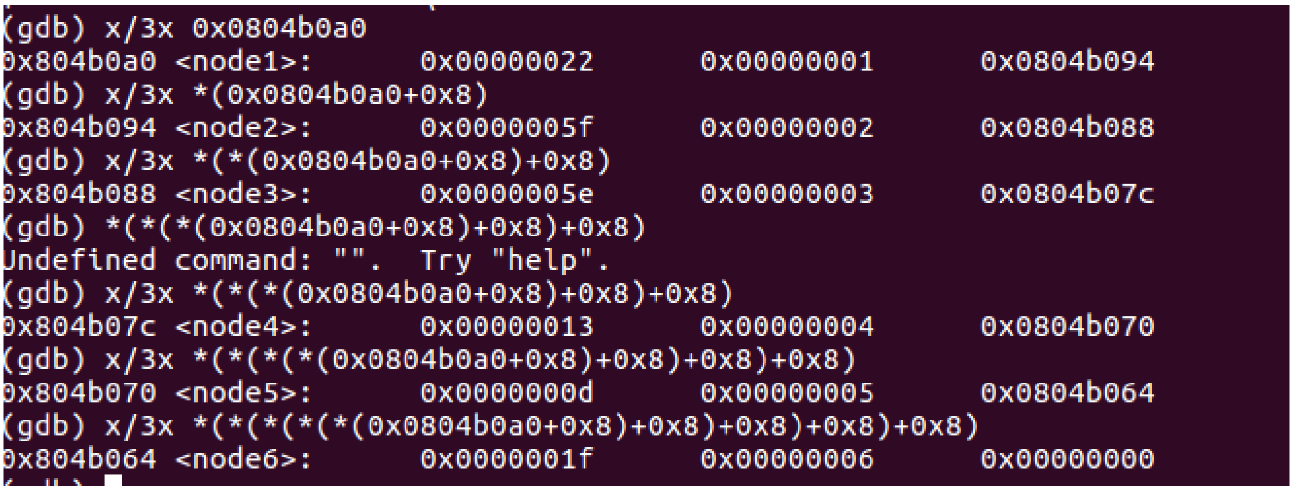
C（伪）代码为

```
for (i=0;i<=5;i++){
        for (j=0;A[i]>j;j++){
                node=node.next;
        }
        B[i]=node;
}
```

这个代码的含义是：让链表中第 A[i]个位置的结点排到 B 数组中第 i 个位置。

随后第三片段，

```
node k=b[0];
for (i=0;i<=5;i++){
        k.next=B[i];
        k=k.next;
}
```



```
(gdb) x/3x 0x0804b0a0
0x804b0a0 <node1>:      0x00000022      0x00000001      0x0804b094
(gdb) x/3x *(0x0804b0a0+0x8)
0x804b094 <node2>:      0x0000005f      0x00000002      0x0804b088
(gdb) x/3x *(*(0x0804b0a0+0x8)+0x8)
0x804b088 <node3>:      0x0000005e      0x00000003      0x0804b07c
(gdb) *(*(*(0x0804b0a0+0x8)+0x8)+0x8)
Undefined command: "".  Try "help".
(gdb) x/3x *(*(*(0x0804b0a0+0x8)+0x8)+0x8)
0x804b07c <node4>:      0x00000013      0x00000004      0x0804b070
(gdb) x/3x *(*(*(*(0x0804b0a0+0x8)+0x8)+0x8)+0x8)
0x804b070 <node5>:      0x0000000d      0x00000005      0x0804b064
(gdb) x/3x *(*(*(*(*(0x0804b0a0+0x8)+0x8)+0x8)+0x8)+0x8)
0x804b064 <node6>:      0x0000001f      0x00000006      0x00000000
```

第四片段说明降序排列，
每个结点的第一个值分别为：22, 5f, 5e, 13, d, 1f
从大到小排列为 5f, 5e, 22, 1f, 13, d
因此 A 数组的值应为 2, 3, 1, 6, 4, 5
故输入 2, 3, 1, 6, 4, 5

秘密阶段：

```
08048df1 <secret_phase>:
 8048df1:   55                      push   %ebp
 8048df2:   89 e5                   mov    %esp,%ebp
 8048df4:   83 ec 18                sub    $0x18,%esp
 8048df7:   83 ec 0c                sub    $0xc,%esp
 8048dfa:   68 c1 90 04 08          push   $0x80490c1
 8048dff:   e8 fc f6 ff ff          call   8048500 <puts@plt>
 8048e04:   83 c4 10                add    $0x10,%esp
 8048e07:   e8 35 fa ff ff          call   8048841 <read_line>  //输入一行字符串
 8048e0c:   89 45 ec                mov    %eax,-0x14(%ebp)  //将输入值保存到-20(%ebp)
 8048e0f:   83 ec 0c                sub    $0xc,%esp
 8048e12:   ff 75 ec                pushl  -0x14(%ebp)
 8048e15:   e8 46 f7 ff ff          call   8048560 <atoi@plt>  //转变为整数
 8048e1a:   83 c4 10                add    $0x10,%esp
 8048e1d:   89 45 f0                mov    %eax,-0x10(%ebp)  //将整数保存到-16(%ebp)
 8048e20:   83 7d f0 00             cmpl   $0x0,-0x10(%ebp)  //比较整数和 0 的大小
 8048e24:   7e 09                   jle    8048e2f <secret_phase+0x3e>  //若小于等于 0，则爆炸
 8048e26:   81 7d f0 e9 03 00 00    cmpl   $0x3e9,-0x10(%ebp)  //比较整数和 1001
 8048e2d:   7e 05                   jle    8048e34 <secret_phase+0x43>  若小于等于 1001，则继续
 8048e2f:   e8 9c fa ff ff          call   80488d0 <explode_bomb>
 8048e34:   83 ec 08                sub    $0x8,%esp
 8048e37:   ff 75 f0                pushl  -0x10(%ebp)  //压入-16(%ebp)，即整数
 8048e3a:   68 54 b1 04 08          push   $0x804b154  //压入地址 0x0804b154 的值
 8048e3f:   e8 4a ff ff ff          call   8048d8e <fun7>
 8048e44:   83 c4 10                add    $0x10,%esp
 8048e47:   89 45 f4                mov    %eax,-0xc(%ebp)  //将返回值保存到-12(%ebp)
 8048e4a:	83 7d f4 00          	cmpl   $0x0,-0xc(%ebp)   //比较返回值和 0 的大小
 8048e4e:	74 05                	je     8048e55 <secret_phase+0x64>  //若等于 0，则成功，否则
失败
 8048e50:	e8 7b fa ff ff       	call   80488d0 <explode_bomb>
 8048e55:	83 ec 0c             	sub    $0xc,%esp
 8048e58:	68 d8 90 04 08       	push   $0x80490d8
 8048e5d:	e8 9e f6 ff ff       	call   8048500 <puts@plt>
 8048e62:	83 c4 10             	add    $0x10,%esp
 8048e65:	e8 60 fa ff ff       	call   80488ca <phase_defused>
 8048e6a:	90                   	nop
 8048e6b:	c9                   	leave  
 8048e6c:	c3                   	ret    
 8048e6d:	66 90                	xchg   %ax,%ax
 8048e6f:	90                   	nop

08048d8e <fun7>:
 8048d8e:	55                   	push   %ebp
 8048d8f:	89 e5                	mov    %esp,%ebp
 8048d91:	83 ec 08             	sub    $0x8,%esp
 8048d94:	83 7d 08 00          	cmpl   $0x0,0x8(%ebp)  //将地址参宿与 0 比较
 8048d98:	75 07                	jne    8048da1 <fun7+0x13>  //若不等于 0，则跳至 a1
 8048d9a:	b8 ff ff ff ff       	mov    $0xffffffff,%eax  //若等于 0，将-1 加载%eax
 8048d9f:	eb 4e                	jmp    8048def <fun7+0x61>  //直接跳至 ef
 8048da1:	8b 45 08             	mov    0x8(%ebp),%eax  //将地址地址加载到%eax
 8048da4:	8b 00                	mov    (%eax),%eax  //求地址的值
 8048da6:	3b 45 0c             	cmp    0xc(%ebp),%eax  //比较地址储存值与整数参数储存的值
 8048da9:	7e 19                	jle    8048dc4 <fun7+0x36>  //若小于等于，则跳至 c4
 8048dab:	8b 45 08             	mov    0x8(%ebp),%eax  //将地址值加载到%eax
 8048dae:	8b 40 04             	mov    0x4(%eax),%eax  //%eax+4
 8048db1:	83 ec 08             	sub    $0x8,%esp
 8048db4:	ff 75 0c             	pushl  0xc(%ebp)
 8048db7:	50                   	push   %eax
 8048db8:	e8 d1 ff ff ff       	call   8048d8e <fun7>  //递归调用
 8048dbd:	83 c4 10             	add    $0x10,%esp
 8048dc0:	01 c0                	add    %eax,%eax  //返回值*2
 8048dc2:	eb 2b                	jmp    8048def <fun7+0x61>  //直接跳至 ef
 8048dc4:	8b 45 08             	mov    0x8(%ebp),%eax  //将地址参数加载到%eax
 8048dc7:	8b 00                	mov    (%eax),%eax  //求地址储存值
 8048dc9:	3b 45 0c             	cmp    0xc(%ebp),%eax  //比较整数参数和地址储存的值
 8048dcc:	75 07                	jne    8048dd5 <fun7+0x47>  //若不等，跳至 d5
 8048dce:	b8 00 00 00 00       	mov    $0x0,%eax  //若相等，将 0 保存到%eax
 8048dd3:	eb 1a                	jmp    8048def <fun7+0x61>  //跳至 ef
 8048dd5:	8b 45 08             	mov    0x8(%ebp),%eax  //将地址值加载到%eax
 8048dd8:	8b 40 08             	mov    0x8(%eax),%eax  //%eax+8
 8048ddb:	83 ec 08             	sub    $0x8,%esp
 8048dde:	ff 75 0c             	pushl  0xc(%ebp)
 8048de1:	50                   	push   %eax
```

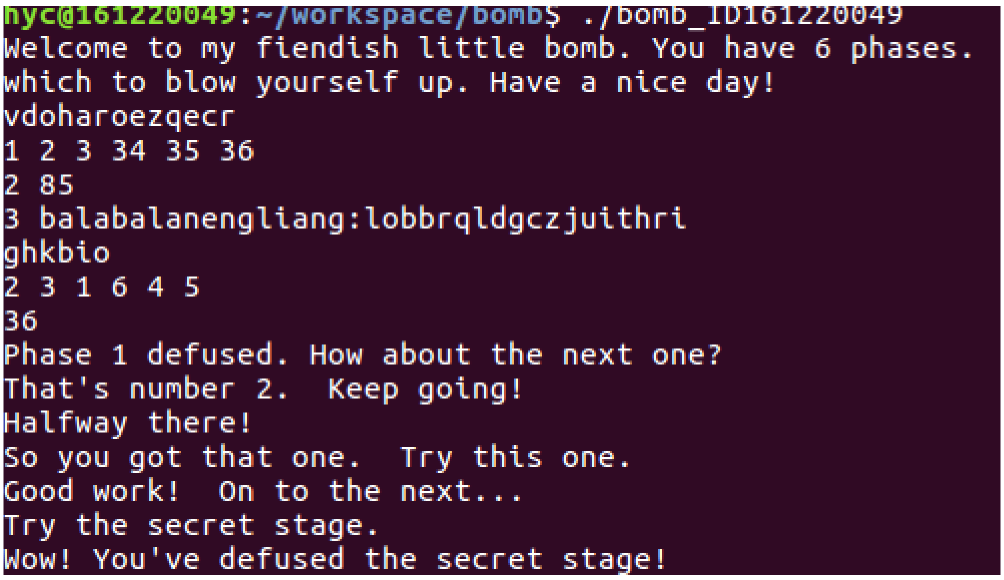
```
8048de2:    e8 a7 ff ff ff        call   8048d8e <fun7>  //递归调用
8048de7:    83 c4 10              add    $0x10,%esp
8048dea:    01 c0                 add    %eax,%eax  //%eax*2
8048dec:    83 c0 01              add    $0x1,%eax  //%eax+1
8048def:    c9               leave
8048df0:    c3               ret
```

可转化为 C 代码：

```
func7(int * address, int input)
{
        if (address==NULL)
                return -1;
        int ret;
        if (*address > input)
        {
                ret=func7(*(address+4),input);
                ret*=2;
        }
        else if ( *address < input)
        {
                ret=func7(*(address+8),input);
                ret=ret*2+1;
        }
        else
                return 0;
        return ret;
}
```

由于需要返回 0，故*address=36，输入 36 即可

因此，完成了拆弹实验：



```
hyc@161220049:~/workspace/bomb$ ./bomb_ID161220049
Welcome to my fiendish little bomb. You have 6 phases.
which to blow yourself up. Have a nice day!
vdoharoezqecr
1 2 3 34 35 36
2 85
3 balabalanengliang:lobbrqldgczjuithri
ghkbio
2 3 1 6 4 5
36
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Try the secret stage.
Wow! You've defused the secret stage!
```