BombLab 实验报告

涂奕腾 2020201018

1.phase_1

```
0000000000001449 <phase_1>:
    1449:       48 83 ec 08             sub    $0x8,%rsp
    144d:       48 8d 35 fc 1c 00 00    lea    0x1cfc(%rip),%rsi        # 3150 <_IO_stdin_used+0x150>
    1454:       e8 bd 05 00 00          call   1a16 <strings_not_equal>
    1459:       85 c0                   test   %eax,%eax
    145b:       75 05                   jne    1462 <phase_1+0x19>
    145d:       48 83 c4 08             add    $0x8,%rsp
    1461:       c3                      ret
    1462:       e8 49 08 00 00          call   1cb0 <explode_bomb>
    1467:       eb f4                   jmp    145d <phase_1+0x14>
```

根据反汇编代码以及注释，我们可以判断需要输入一个字符串与目标字符串相等。输出寄存器%rsi 所存储的地址中的内容，可以知道需要输入的字符串为：“The future will be better tomorrow.”

```
(gdb) x/s $rsi
0x555555557150: "The future will be better tomorrow."
```

2.phase_2

根据函数名，我们可以知道这一问需要我们输入 6 个数字。

```
1482:       e8 65 08 00 00          call   1cec <read_six_numbers>
1487:       83 3c 24 01             cmpl   $0x1,(%rsp)
148b:       75 0a                   jne    1497 <phase 2+0x2e>
```

由此可见，输入的第一个数字应当是 1。

```
149e:       48 83 c3 04             add    $0x4,%rbx
14a2:       48 39 eb                cmp    %rbp,%rbx
14a5:       74 10                   je     14b7 <phase_2+0x4e>
14a7:       8b 03                   mov    (%rbx),%eax
14a9:       01 c0                   add    %eax,%eax
14ab:       39 43 04                cmp    %eax,0x4(%rbx)
14ae:       74 ee                   je     149e <phase_2+0x35>
14b0:       e8 fb 07 00 00          call   1cb0 <explode_bomb>
14b5:       eb e7                   jmp    149e <phase_2+0x35>
14b7:       48 8b 44 24 18          mov    0x18(%rsp),%rax
14bc:       64 48 2b 04 25 28 00    sub    %fs:0x28,%rax
14c3:       00 00
```

分析反汇编代码，%rbx 记录当前位置的数组中的元素。由 0x14a7 到 0x14ab 可知，数组中对于某一个位置上的元素，其下一个元素应当是该位置上元素的两倍。由此，我们知道输入的 6 个数应当是以 1 为首项，2 为公比的等比数列：“1 2 4 8 16 32”。

3.phase_3

观察汇编代码，发现有多个重复的 je, call,mov,jmp,mov,cmpl 结构，猜测其为 switch

语句。首先从输入开始，打印%rsi 所指地址的内容：

```
14f1:       4c 8d 44 24 14          lea    0x14(%rsp),%r8
14f6:       48 8d 35 a1 1c 00 00    lea    0x1ca1(%rip),%rsi        # 319e <_IO_stdin_used+0x19e>
```

```
(gdb) x/s $rsi
0x55555555719e: "%d %c %d"
```

可知输入的是“int,char,int”。再观察前面的 lea 语句：

```
lea    0xf(%rsp),%rcx
lea    0x10(%rsp),%rdx
lea    0x14(%rsp),%r8
```

经过尝试后知道输入的字符在%rsp+15 的位置；第一个整数在%rsp+16 的位置；第二个整数在%rsp+20 的位置。再往下观察，知道输入的第一个整数视作无符号数应当<=7:

```
1507:       83 7c 24 10 07          cmpl   $0x7,0x10(%rsp)
150c:       0f 87 05 01 00 00       ja     1617 <phase_3+0x144>
```

我们不妨设输入的第一个整数就是 7，我们逐步运行，依次运行了反汇编代码中的如下部分：

```
15ff:       b8 6b 00 00 00          mov    $0x6b,%eax
1604:       83 7c 24 14 64          cmpl   $0x64,0x14(%rsp)
1609:       74 16                   je     1621 <phase_3+0x14e>

1621:       38 44 24 0f             cmp    %al,0xf(%rsp)
1625:       75 15                   jne    163c <phase_3+0x169>
1627:       48 8b 44 24 18          mov    0x18(%rsp),%rax
162c:       64 48 2b 04 25 28 00    sub    %fs:0x28,%rax
1633:       00 00
```

可知输入的第二个整数应当是 0x64=100，字符的 ascii 码值是 0x6b=107 为’k’。至此得出 phase_3 的一个答案是“7 k 100”。

4.phase_4

打印%rsi 所指地址的内容，知输入两个整数。

```
169b:       48 8d 35 a3 1d 00 00    lea    0x1da3(%rip),%rsi        # 3445 <array.0+0x265>
16a2:       e8 b9 fa ff ff          call   1160 <__isoc99_sscanf@plt>
```

```
(gdb) x/s $rsi
0x555555557445: "%d %d"
```

经过测试，(%rsp)存储的应当是输入的第二个整数，(%rsp+4)存储的是输入的第一个整数。知第二个整数视作无符号数应当<=4，不妨设输入的第二个整数是 3。

```
16ac:       8b 04 24                mov    (%rsp),%eax
16af:       83 e8 02                sub    $0x2,%eax
16b2:       83 f8 02                cmp    $0x2,%eax
16b5:       76 05                   jbe    16bc <phase_4+0x3d>
16b7:       e8 f4 05 00 00          call   1cb0 <explode_bomb>
16bc:       8b 34 24                mov    (%rsp),%esi
```

随后 3 会用作函数 func4 的参数，返回值存在%eax 中,当我们输入的第一个值与返

回值相等时则解决问题。于是我们输出%eax 的值：

```
16bc:       8b 34 24                mov    (%rsp),%esi
16bf:       bf 08 00 00 00          mov    $0x8,%edi
16c4:       e8 7f ff ff ff          call   1648 <func4>
16c9:       39 44 24 04             cmp    %eax,0x4(%rsp)
16cd:       75 15                   jne    16e4 <phase_4+0x65>
16cf:       48 8b 44 24 08          mov    0x8(%rsp),%rax
16d4:       64 48 2b 04 25 28 00    sub    %fs:0x28,%rax
16db:       00 00
```

```
(gdb) p $eax
$1 = 162
```

由此可知 phase_4 的一个答案为“162 3”。

5.phase_5

还是先打印输入信息，知输入两个整数。

```
1709:       48 89 e2                mov    %rsp,%rdx
170c:       48 8d 35 32 1d 00 00    lea    0x1d32(%rip),%rsi        # 3445 <array.0+0x265>
```

```
(gdb) x/s $rsi
0x555555557445: "%d %d"
```

往后走，可以发现输入的第一个数字不能为 0xf =15：

```
171d:       8b 04 24                mov    (%rsp),%eax
1720:       83 e0 0f                and    $0xf,%eax
1723:       89 04 24                mov    %eax,(%rsp)
1726:       83 f8 0f                cmp    $0xf,%eax
1729:       74 32                   je     175d <phase_5+0x6d>
```

接着进入代码的核心部分。

```
172b:       b9 00 00 00 00          mov    $0x0,%ecx
1730:       ba 00 00 00 00          mov    $0x0,%edx
1735:       48 8d 35 a4 1a 00 00    lea    0x1aa4(%rip),%rsi        # 31e0 <array.0>
173c:       83 c2 01                add    $0x1,%edx
173f:       48 98                   cltq
1741:       8b 04 86                mov    (%rsi,%rax,4),%eax
1744:       01 c1                   add    %eax,%ecx
1746:       83 f8 0f                cmp    $0xf,%eax
1749:       75 f1                   jne    173c <phase_5+0x4c>
174b:       c7 04 24 0f 00 00 00    movl   $0xf,(%rsp)
1752:       83 fa 0f                cmp    $0xf,%edx
1755:       75 06                   jne    175d <phase_5+0x6d>
1757:       39 4c 24 04             cmp    %ecx,0x4(%rsp)
175b:       74 05                   je     1762 <phase_5+0x72>
175d:       e8 4e 05 00 00          call   1cb0 <explode_bomb>
1762:       48 8b 44 24 08          mov    0x8(%rsp),%rax
1767:       64 48 2b 04 25 28 00    sub    %fs:0x28,%rax
176e:       00 00
```

容易观察到这是一个循环的结构，当%edx 中的值=15（即进行了 15 次循环后）循环结束，再比较%ecx 和输入的第二个数。观察代码，%rsi 中应该是存了一个数组（记为 a[]）的地址，而每次循环则是进行%eax=a[%eax]的迭代，此时%ecx 会加上%eax 的值，所以%ecx 的作用就是记录所有迭代到的数组元素的值。我们打印$rsi 所指向的数组：

```
0x5555555571e0 <array.0>:       0x0a    0x00    0x00    0x00    0x02    0x00    0x00    0x00
0x5555555571e8 <array.0+8>:     0x0e    0x00    0x00    0x00    0x07    0x00    0x00    0x00
0x5555555571f0 <array.0+16>:    0x08    0x00    0x00    0x00    0x0c    0x00    0x00    0x00
0x5555555571f8 <array.0+24>:    0x0f    0x00    0x00    0x00    0x0b    0x00    0x00    0x00
0x555555557200 <array.0+32>:    0x00    0x00    0x00    0x00    0x04    0x00    0x00    0x00
0x555555557208 <array.0+40>:    0x01    0x00    0x00    0x00    0x0d    0x00    0x00    0x00
0x555555557210 <array.0+48>:    0x03    0x00    0x00    0x00    0x09    0x00    0x00    0x00
0x555555557218 <array.0+56>:    0x06    0x00    0x00    0x00    0x05    0x00    0x00    0x00
```

可知数组的定义为：int a[16]={10,2,14,7,8,12,15,11,0,4,1,13,3,9,6,5};

我们可以用 C 程序简单模拟这个过程查看对于每个输入一共进行多少轮循环：

```
#include<bits/stdc++.h>
using namespace std;
int a[16]={10,2,14,7,8,12,15,11,0,4,1,13,3,9,

int cal(int x){
    int cnt=0;
    while(x!=15){
        x=a[x];
        ++cnt;
    }
    return cnt;
}

int main()
{
    for(int i=0;i<=15;++i){
        cout<<i<<' '<<cal(i)<<endl;
    }
}
```

```
0 6
1 4
2 3
3 13
4 8
5 15
6 1
7 12
8 7
9 9
10 5
11 11
12 14
13 10
14 2
15 0
```

由此可知，输入的第一个整数应该是 5，第二个整数是 0~15 求和后减去 5 即 115，故 phase_5 的答案为“5 115”。

6.phase_6

首先根据 read_six_numbers 函数知此问应输入 6 个整数，按顺序进入如下部分：

(1)Part1

该部分的主体是一个双重循环（其中%r15，%rbx 相当于循环的计数器），其作用是保证输入的每个数都是<=6 的正整数（phase_6+269，phase_6+272 行），且每个数互不相等（phase_6+93，phase_6+96 行），由此可见本问的输入应当是 1-6 的一个排列。

```
0x555555555882 <phase_6+255>    add    $0x1,%r15
0x555555555886 <phase_6+259>    add    $0x4,%r14
0x55555555588a <phase_6+263>    mov    %r14,%rbp
0x55555555588d <phase_6+266>    mov    (%r14),%eax
0x555555555890 <phase_6+269>    sub    $0x1,%eax
0x555555555893 <phase_6+272>    cmp    $0x5,%eax
0x555555555896 <phase_6+275>    ja     0x5555555557c4 <phase_6+65>
0x55555555589c <phase_6+281>    cmp    $0x5,%r15d
0x5555555558a0 <phase_6+285>    jg     0x5555555557ec <phase_6+105>
0x5555555558a6 <phase_6+291>    mov    %r15,%rbx
0x5555555558a9 <phase_6+294>    jmpq   0x5555555557db <phase_6+88>
```

```
0x5555555557ce <phase_6+75>     add    $0x1,%rbx
0x5555555557d2 <phase_6+79>     cmp    $0x5,%ebx
0x5555555557d5 <phase_6+82>     jg     0x555555555882 <phase_6+255>
0x5555555557db <phase_6+88>     mov    0x0(%r13,%rbx,4),%eax
0x5555555557e0 <phase_6+93>     cmp    %eax,0x0(%rbp)
0x5555555557e3 <phase_6+96>     jne    0x5555555557ce <phase_6+75>
0x5555555557e5 <phase_6+98>     callq  0x555555555cb0 <explode_bomb>
```

(2)Part2

这一部分也是一个循环，将数组中第一个元素的地址传给%rdx 后再让%rdx 指向数组中最后一个元素（用作循环结束标志），然后对于每个元素，用 7 减去该元素的值后再将用这个值替换该位置上的元素。由此，数组上的六个数都变成了 7 减去原数。

```
0x5555555557ec <phase_6+105>    mov    0x8(%rsp),%rdx
0x5555555557f1 <phase_6+110>    add    $0x18,%rdx
0x5555555557f5 <phase_6+114>    mov    $0x7,%ecx
0x5555555557fa <phase_6+119>    mov    %ecx,%eax
0x5555555557fc <phase_6+121>    sub    (%r12),%eax
0x555555555800 <phase_6+125>    mov    %eax,(%r12)
0x555555555804 <phase_6+129>    add    $0x4,%r12
0x555555555808 <phase_6+133>    cmp    %r12,%rdx
0x55555555580b <phase_6+136>    jne    0x5555555557fa <phase_6+119>
```

(3)Part3

随后我们发现了一个奇怪的地址<node1>，打印地址信息，我们可以猜测这是一个链表结构，其中第一个元素为 val 值，第二个元素为编号，第三个元素为下一节点的地址。随后进入一个两重循环，其作用是将链表中节点按照输入的编号进行排序。

```
0x555555559330 <node1>: 531     1       1431671616      21845
0x555555559340 <node2>: 310     2       1431671632      21845
0x555555559350 <node3>: 548     3       1431671648      21845
0x555555559360 <node4>: 515     4       1431671664      21845
0x555555559370 <node5>: 603     5       1431671312      21845
0x555555559380 <host_table>:    1431663775      21845   1431663786      21845
```

```
0x555555555812 <phase_6+143>    mov    0x10(%rsp,%rsi,4),%ecx
0x555555555816 <phase_6+147>    mov    $0x1,%eax
0x55555555581b <phase_6+152>    lea    0x3b0e(%rip),%rdx        # 0x555555559330 <node1>
0x555555555822 <phase_6+159>    cmp    $0x1,%ecx
0x555555555825 <phase_6+162>    jle    0x555555555832 <phase_6+175>
0x555555555827 <phase_6+164>    mov    0x8(%rdx),%rdx
0x55555555582b <phase_6+168>    add    $0x1,%eax
```

```
0x55555555582e <phase_6+171>    cmp    %ecx,%eax
0x555555555830 <phase_6+173>    jne    0x555555555827 <phase_6+164>
0x555555555832 <phase_6+175>    mov    %rdx,0x30(%rsp,%rsi,8)
0x555555555837 <phase_6+180>    add    $0x1,%rsi
0x55555555583b <phase_6+184>    cmp    $0x6,%rsi
0x55555555583f <phase_6+188>    jne    0x555555555812 <phase_6+143>
```

(4)Part4

输出节点的内容可知，这部分的作用是让节点按照我们确定的位置连接。

```
0x555555555841 <phase_6+190>    mov    0x30(%rsp),%rbx
0x555555555846 <phase_6+195>    mov    0x38(%rsp),%rax
0x55555555584b <phase_6+200>    mov    %rax,0x8(%rbx)
0x55555555584f <phase_6+204>    mov    0x40(%rsp),%rdx
0x555555555854 <phase_6+209>    mov    %rdx,0x8(%rax)
0x555555555858 <phase_6+213>    mov    0x48(%rsp),%rax
0x55555555585d <phase_6+218>    mov    %rax,0x8(%rdx)
0x555555555861 <phase_6+222>    mov    0x50(%rsp),%rdx
0x555555555866 <phase_6+227>    mov    %rdx,0x8(%rax)
```

```
0x55555555586a <phase_6+231>    mov    0x58(%rsp),%rax
0x55555555586f <phase_6+236>    mov    %rax,0x8(%rdx)
0x555555555873 <phase_6+240>    movq   $0x0,0x8(%rax)
0x55555555587b <phase_6+248>    mov    $0x5,%ebp
```

```
0x555555559330 <node1>: 0x00000213      0x00000001      0x55559360      0x00005555
0x555555559340 <node2>: 0x00000136      0x00000002      0x55559210      0x00005555
0x555555559350 <node3>: 0x00000224      0x00000003      0x55559330      0x00005555
0x555555559360 <node4>: 0x00000203      0x00000004      0x55559340      0x00005555
0x555555559370 <node5>: 0x0000025b      0x00000005      0x55559350      0x00005555
```

(5)Part5

这一部分的循环作用是确保重新连接后的链表中元素的 val 值递减，否则炸弹爆炸（phase_6+314 到 phase_6 到 316）。

```
0x5555555558ae <phase_6+299>    mov    0x8(%rbx),%rbx
0x5555555558b2 <phase_6+303>    sub    $0x1,%ebp
0x5555555558b5 <phase_6+306>    je     0x5555555558c8 <phase_6+325>
0x5555555558b7 <phase_6+308>    mov    0x8(%rbx),%rax
0x5555555558bb <phase_6+312>    mov    (%rax),%eax
0x5555555558bd <phase_6+314>    cmp    %eax,(%rbx)
0x5555555558bf <phase_6+316>    jge    0x5555555558ae <phase_6+299>
0x5555555558c1 <phase_6+318>    callq  0x555555555cb0 <explode_bomb>
```

根据之前输出的节点信息，我们知道节点值有 val5>val3>val1>val4>val2，由于在 Part2 中编号进行了替换，所以输入的顺序为 2,4,6,3,5。但由于只输出了 5 个节点信息，我们猜测剩下的一个 1 应该在最后一位或第一位。经检验，答案为“2 4 6 3 5 1”。

7.secret_phase

(1)如何进入 secret_phase

通过在反汇编代码中查找 secret_phase，知其一共出现了六次，而 5 次都是在该函数本身内出现，只有一次是在 phase_defused 函数中出现，故在 phase_defused 函数中

查找线索。

```
  1e7f:       83 3d 8a 39 00 00 06    cmpl   $0x6,0x398a(%rip)        # 5810 <num_input_strings>
  1e86:       74 19                   je     1ea1 <phase_defused+0x40>
  1e88:       48 8b 44 24 68          mov    0x68(%rsp),%rax
  1e8d:       64 48 2b 04 25 28 00    sub    %fs:0x28,%rax
  1e94:       00 00
  1e96:       0f 85 84 00 00 00       jne    1f20 <phase_defused+0xbf>
  1e9c:       48 83 c4 78             add    $0x78,%rsp
  1ea0:       c3                      ret
```

首先应当满足 0x398a(%rip)中存储的值等于 6 才能继续进行。我们查找 num_input_strings 出现的位置：

```
0000000000001d2d <read_line>:
    1d2d:       55                      push   %rbp
    1d2e:       53                      push   %rbx
    1d2f:       48 83 ec 08             sub    $0x8,%rsp
    1d33:       b8 00 00 00 00          mov    $0x0,%eax
    1d38:       e8 5a fe ff ff          call   1b97 <skip>
    1d3d:       48 85 c0                test   %rax,%rax
    1d40:       74 5d                   je     1d9f <read_line+0x72>
    1d42:       8b 2d c8 3a 00 00       mov    0x3ac8(%rip),%ebp        # 5810 <num_input_strings>
    1d48:       48 63 c5                movslq %ebp,%rax
    1d4b:       48 8d 1c 80             lea    (%rax,%rax,4),%rbx
    1d4f:       48 c1 e3 04             shl    $0x4,%rbx
    1d53:       48 8d 05 c6 3a 00 00    lea    0x3ac6(%rip),%rax        # 5820 <input_strings>
    1d5a:       48 01 c3                add    %rax,%rbx
    1d5d:       48 89 df                mov    %rbx,%rdi
    1d60:       e8 2b f3 ff ff          call   1090 <strlen@plt>
    1d65:       83 f8 4e                cmp    $0x4e,%eax
    1d68:       0f 8f a9 00 00 00       jg     1e17 <read_line+0xea>
    1d6e:       83 e8 01                sub    $0x1,%eax
    1d71:       48 98                   cltq
    1d73:       48 63 d5                movslq %ebp,%rdx
    1d76:       48 8d 0c 92             lea    (%rdx,%rdx,4),%rcx
    1d7a:       48 c1 e1 04             shl    $0x4,%rcx
    1d7e:       48 8d 15 9b 3a 00 00    lea    0x3a9b(%rip),%rdx        # 5820 <input_strings>
    1d85:       48 01 ca                add    %rcx,%rdx
    1d88:       c6 04 02 00             movb   $0x0,(%rdx,%rax,1)
    1d8c:       83 c5 01                add    $0x1,%ebp
    1d8f:       89 2d 7b 3a 00 00       mov    %ebp,0x3a7b(%rip)        # 5810 <num_input_strings>
    1d95:       48 89 d8                mov    %rbx,%rax
    1d98:       48 83 c4 08             add    $0x8,%rsp
    1d9c:       5b                      pop    %rbx
    1d9d:       5d                      pop    %rbp
    1d9e:       c3                      ret
```

在 read_line 函数中，根据这两处 num_input_strings 出现的位置可知，每次运行一次 read_line 函数，num_input_strings 的数值+1。查找 read_line 函数出现的位置，在从 main 函数进入每个 phase 之前都会调用一次 read_line，由此可知，只有在完成 phase_6 之后才会在 phase_defused 中继续运行。

回到 phase_defused 的汇编代码，继续探索如何进行 secret_phase。注意到函数中的两个地址并查询这两个内存中的信息：

```
    1eb0:       48 8d 35 d8 15 00 00    lea    0x15d8(%rip),%rsi        # 348f <array.0+0x2af>
    1eb7:       48 8d 3d 52 3a 00 00    lea    0x3a52(%rip),%rdi        # 5910 <input_strings+0xf0>
```

```
(gdb) x/s 0x348f
0x348f: "%d %d %s"
(gdb) x/s 0x5910
0x5910 <input_strings+240>:      ""
```

在后面的部分，出现了 phase_1 中的 strings_not_equal 函数，我们在查询地址 0x3498 中的内容：

```
    1eec:       48 8d 35 a5 15 00 00    lea    0x15a5(%rip),%rsi        # 3498 <array.0+0x2b8>
    1ef3:       e8 1e fb ff ff          call   1a16 <strings_not_equal>
```

```
(gdb) x/s 0x3498
0x3498: "DrEvil"
```

综合以上两点，我们可以知道，想要进入 secret_phase，我们需要输入两个 int 型数和一个字符串“DrEvil”。结合 phase_1~phase_6 的输入，我们猜测应该在 phase_4 或 phase_5 的输入后加上一个字符串“DrEvil”即可。经过尝试，在 phase_4 的输入后加上一个字符串“DrEvil”时，完成 phase_1~phase_6 后即可进入 secret_phase。

(后经查询资料得知，对于打印出来是空串的地址值 0x5910，在其所在行设置断点，运行后知只有一次被断点且此时该内存中存储的内容为第四关的答案，由此知道进入条件为在 phase_4 的输入后加上“DrEvil”)

(2)拆炸弹

```
0000000000001929 <secret_phase>:
    1929:       53                      push   %rbx
    192a:       e8 fe 03 00 00          call   1d2d <read_line>
    192f:       48 89 c7                mov    %rax,%rdi
    1932:       ba 0a 00 00 00          mov    $0xa,%edx
    1937:       be 00 00 00 00          mov    $0x0,%esi
    193c:       e8 ff f7 ff ff          call   1140 <strtol@plt>
    1941:       89 c3                   mov    %eax,%ebx
    1943:       83 e8 01                sub    $0x1,%eax
    1946:       3d e8 03 00 00          cmp    $0x3e8,%eax
    194b:       77 25                   ja     1972 <secret_phase+0x49>
    194d:       89 de                   mov    %ebx,%esi
    194f:       48 8d 3d fa 38 00 00    lea    0x38fa(%rip),%rdi        # 5250 <n1>
    1956:       e8 91 ff ff ff          call   18ec <fun7>
    195b:       85 c0                   test   %eax,%eax
    195d:       75 1a                   jne    1979 <secret_phase+0x50>
    195f:       48 8d 3d 12 18 00 00    lea    0x1812(%rip),%rdi        # 3178 <_IO_stdin_used+0x178>
    1966:       e8 05 f7 ff ff          call   1070 <puts@plt>
    196b:       e8 f1 04 00 00          call   1e61 <phase_defused>
    1970:       5b                      pop    %rbx
    1971:       c3                      ret
    1972:       e8 39 03 00 00          call   1cb0 <explode_bomb>
    1977:       eb d4                   jmp    194d <secret_phase+0x24>
    1979:       e8 32 03 00 00          call   1cb0 <explode_bomb>
    197e:       eb df                   jmp    195f <secret_phase+0x36>
```

对于输入部分，将一个字符串转当作十进制的数存在%eax 中，在根据 sub 语句和 cmp 语句可知我们需要输入一个整数且这个数应当介于 1 到 0x3e9(1001)之间。此时我们输入的数会作为 fun7 的第二个参数传入(第一个参数为%rdi 所存储的地址)，同时我们的目的是让 fun7 的返回值等于 0（由 test-jne 语句可知）。我们查看 fun7：

```
00000000000018ec <fun7>:
    18ec:       48 85 ff                test   %rdi,%rdi
    18ef:       74 32                   je     1923 <fun7+0x37>
    18f1:       48 83 ec 08             sub    $0x8,%rsp
    18f5:       8b 17                   mov    (%rdi),%edx
    18f7:       39 f2                   cmp    %esi,%edx
    18f9:       7f 0c                   jg     1907 <fun7+0x1b>
    18fb:       b8 00 00 00 00          mov    $0x0,%eax
    1900:       75 12                   jne    1914 <fun7+0x28>
    1902:       48 83 c4 08             add    $0x8,%rsp
    1906:       c3                      ret
    1907:       48 8b 7f 08             mov    0x8(%rdi),%rdi
    190b:       e8 dc ff ff ff          call   18ec <fun7>
    1910:       01 c0                   add    %eax,%eax
    1912:       eb ee                   jmp    1902 <fun7+0x16>
    1914:       48 8b 7f 10             mov    0x10(%rdi),%rdi
    1918:       e8 cf ff ff ff          call   18ec <fun7>
    191d:       8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
    1921:       eb df                   jmp    1902 <fun7+0x16>
    1923:       b8 ff ff ff ff          mov    $0xffffffff,%eax
    1928:       c3                      ret
```

容易看出这是一个递归函数，且 0x1907-0x1912 的结构和 0x1914-0x1921 的结构基本一致， 都是对 fun7 进行递归后返回值*2 或*2+1。我们再观察%rdi 所存储地址的内容寻找线索：

```
0x555555559250 <n1>:     36      0       1431671408      21845
0x555555559260 <n1+16>: 1431671440      21845   0       0
0x555555559270 <n21>:    8       0       1431671536      21845
0x555555559280 <n21+16>:         1431671472      21845   0       0
0x555555559290 <n22>:    50      0       1431671504      21845
0x5555555592a0 <n22+16>:         1431671568      21845   0       0
0x5555555592b0 <n32>:    22      0       1431671216      21845
0x5555555592c0 <n32+16>:         1431671152      21845   0       0
```

可以观察到，这是一个二叉树的结构，且对于一个节点其地址为(%rdi)时，val 信息存储在(%rdi)，不妨设左儿子 ls 的地址为(%rdi)+8,右儿子 rs 的地址为 (%rdi)+16,由此我们不难写出 fun7 的 c 语言代码：

```c
int func7(Node *t, int v){
    if(t == NULL) return -1;
    int u = t->val;
    if(u > v) return 2 * fun7(t->ls, v);
    else if(u < v) return 2 * fun7(t->rs, v) + 1;
    else return 0;
}
```

由此可知，我们若想让 fun7 的返回值为 0，则输入的整数应当为这颗二叉树树根的 val 值，通过之前打印%rdi 地址上的内容知这个值为 36，故 secret_phase 的答案为“36”。

综上，所有 phase 的答案为：（其中“DeEvil”为进入 secret_phase 的条件）

The future will be better tomorrow.

1 2 4 8 16 32

7 k 100

162 3 DrEvil

5 115

2 4 6 3 5 1

36