这部分内容打过 Pwn 的应该都能秒吧，gdb 调试就完了

phase_1

直接下断点 `b phase_1` 之后运行到一个 `strings_not_equal` 函数，在里面看见了我们应当输入的字符串

```
▶ 0x400ee9 <phase_1+9>     call   strings_not_equal <0x401338>
       rdi: 0x603780 (input_strings) ◂— 0x647361 /* 'asd' */
       rsi: 0x402400 ◂— outsd  dx, dword ptr [rsi] /* 'Border relations with Canada have never been better.' */
       rdx: 0x1
       rcx: 0x3
```

所以第一步输入 `Border relations with Canada have never been better.` 就能绕过了

phase_2

直接下断点 `b phase_2`，在正确输入前面的字符串后来到 `read_six_numbers` 函数中

```
▶ 0x400f05 <phase_2+9>     call   read_six_numbers <0x40145c>
       rdi: 0x6037d0 (input_strings+80) ◂— 0x6473 /* 'sd' */
       rsi: 0x7fffffffdc40 ◂— 0x0
       rdx: 0x2
       rcx: 0x2
```

在其中我们可以看到负责输入的 `sscanf` 函数

```
▶ 0x40148a <read_six_numbers+46>    call   __isoc99_sscanf@plt <0x400bf0>
       s: 0x6037d0 (input_strings+80) ◂— 0x6473 /* 'sd' */
       format: 0x4025c3 ◂— '%d %d %d %d %d %d'
       vararg: 0x7fffffffdc40 ◂— 0x0
```

运行完程序会依靠返回值查看你到底输入了几个数

```
▶ 0x40148f <read_six_numbers+51>    cmp    eax, 5
  0x401492 <read_six_numbers+54>    jg     read_six_numbers+61 <0x401499>
```

直接用 `set $eax=6` 修改掉 eax 的值，使程序向下继续运行

```
▶ 0x400f0a <phase_2+14>             cmp    dword ptr [rsp], 1
  0x400f0e <phase_2+18>             je     phase_2+52 <0x400f30>
```

第一个数是 1

```
RAX  0x2
▶ 0x400f1c <phase_2+32>    cmp    dword ptr [rbx], eax
  0x400f1e <phase_2+34>    je     phase_2+41 <0x400f25>
```

第二个数是 2

```
▶ 0x400f29 <phase_2+45>    cmp    rbx, rbp
  0x400f2c <phase_2+48>    jne    phase_2+27 <0x400f17>
```

之后的汇编跟用于判断第二个数的汇编都一样了，每次看 RAX 的值就可以知道是什么了

得到的 6 个数分别为：`1 2 4 8 16 32`

phase_3

直接下断点 `b phase_3`，在正确输入前面的字符串后来到 `sscanf` 函数

```
▶ 0x400f5b <phase_3+24>    call   __isoc99_sscanf@plt <0x400bf0>
       s: 0x603820 (input_strings+160) ◂— 0x333231 /* '123' */
       format: 0x4025cf ◂— 0x7245006425206425 /* '%d %d' */
       vararg: 0x7fff45abc358 —▸ 0x4014ac (read_line+14) ◂— test   rax, rax
```

可以看到我们应当输入两个数

```
▶ 0x400f6a <phase_3+39>     cmp    dword ptr [rsp + 8], 7
  0x400f6f <phase_3+44>     ja     phase_3+106 <0x400fad>
pwndbg> x/1gx $rsp + 8
0x7ffc25205f78: 0x0000001700000001
```

所以第一个数应该输入 7

```
 RAX  0x147
▶ 0x400fbe <phase_3+123>    cmp    eax, dword ptr [rsp + 0xc]
  0x400fc2 <phase_3+127>    je     phase_3+134 <0x400fc9>
pwndbg> x/1gx $rsp + 0xc
0x7ffc25205f7c: 0x00400c9000000017
```

所以第二个数应该输入 327

phase_4

直接下断点 `b phase_4`，在正确输入前面的字符串后来到 `sscanf` 函数

```
▶ 0x401024 <phase_4+24>    call   __isoc99_sscanf@plt <0x400bf0>
       s: 0x603870 (input_strings+240) ◂— 0x33342032 /* '2 43' */
       format: 0x4025cf ◂— 0x7245006425206425 /* '%d %d' */
       vararg: 0x7ffc25205f78 —▸ 0x4014ac (read_line+14) ◂— test   rax, rax
```

可以看到我们应当输入两个数

```
▶ 0x40102e <phase_4+34>    cmp    dword ptr [rsp + 8], 0xe
  0x401033 <phase_4+39>    jbe    phase_4+46 <0x40103a>
pwndbg> x/1gx $rsp + 8
0x7ffc25205f78: 0x0000002b00000002
```

所以第一个数要小于等于 14，之后回来到 func 函数

```
▶ 0x401048 <phase_4+60>    call   func4 <0x400fce>
        rdi: 0x2
        rsi: 0x0
        rdx: 0xe
        rcx: 0x20
```

之后比较难调，然后我想到了 IDA。。有 IDA 好像随便做了

```
__int64 __fastcall phase_4(__int64 a1)
{
  __int64 v1; // rdi
  __int64 result; // rax
  unsigned int v3; // [rsp+8h] [rbp-10h]
  int v4; // [rsp+Ch] [rbp-Ch]

  if ( __isoc99_sscanf(a1, "%d %d", &v3, &v4) != 2 || v3 > 0xE )
    explode_bomb(a1, "%d %d");
  v1 = v3;
  result = func4(v3, 0LL, 14LL);
  if ( result || v4 )
    explode_bomb(v1, 0LL);
  return result;
}
```

可以看到函数及传参方式是 `func4(v3, 0LL, 14LL)`，并且 v4 和最后的 result 都要等于 0

```
__int64 __fastcall func4(__int64 a1, __int64 a2, __int64 a3)
{
  signed int v3; // ecx
  __int64 result; // rax

  v3 = (a3 - a2) / 2 + a2;
  if ( v3 > a1 )
    return 2 * func4(a1, a2, (v3 - 1));
  result = 0LL;
  if ( v3 < a1 )
    result = 2 * func4(a1, (v3 + 1), a3) + 1;
  return result;
}
```

这里说明最后要达成式子 `v3 == a1` 且 `result == 0`

设 v3 为 y，a2 为 x，a1 为 z，那么式子为

$$y = \frac{14 + x}{2}$$

因为 x 的初始值是 0，那么一开始 y 就等于 7

而且也能看出，只要 `y == z`，那么返回值就是 0

也就是说 z 为 7 就能达成条件，所以这道题应该输入 `7 0`

phase_5

```
unsigned __int64 __fastcall phase_5(__int64 a1, __int64 a2)
{
  __int64 v2; // rax
  char v4[6]; // [rsp+10h] [rbp-18h]
  char v5; // [rsp+16h] [rbp-12h]
  unsigned __int64 v6; // [rsp+18h] [rbp-10h]

  v6 = __readfsqword(0x28u);
  if ( string_length() != 6 )
    explode_bomb(a1, a2);
  v2 = 0LL;
  do
  {
    v4[v2] = array_3449[*(a1 + v2) & 0xF];
    ++v2;
  }
  while ( v2 != 6 );
  v5 = 0;
  if ( strings_not_equal(v4, "flyers") )
    explode_bomb(v4, "flyers");
  return __readfsqword(0x28u) ^ v6;
}
```

所以这题应该输入的字符串来源脚本如下：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

a = list('flyers')
table = list('maduiersnfotvbyl')
res = ''
for i in range(0, len(a)):
    for j in range(0, len(table)):
        if table[j] == a[i]:
            res += '\\x' + hex(j)[2:].rjust(2, '0')
            break
print res
```

得到字符串为 `\x09\x0f\x0e\x05\x06\x07`

phase_6

```
__int64 __fastcall phase_6(__int64 a1)
{
  int *v1; // r13
  signed int v2; // er12
  signed int v3; // ebx
  char *v4; // rax
  unsigned __int64 v5; // rsi
  int *v6; // rdx
  signed int v7; // eax
  int v8; // ecx
  __int64 v9; // rbx
  char *v10; // rax
  __int64 i; // rcx
  __int64 v12; // rdx
```

```
15  signed int v13; // ebp
16  __int64 result; // rax
17  int v15[6]; // [rsp+0h] [rbp-78h]
18  char v16; // [rsp+18h] [rbp-60h]
19  __int64 v17; // [rsp+20h] [rbp-58h]
20  char v18; // [rsp+28h] [rbp-50h]
21  char v19; // [rsp+50h] [rbp-28h]
22
23  v1 = v15;
24  read_six_numbers(a1, v15);
25  v2 = 0;
26  while ( 1 )
27  {
28    if ( (*v1 - 1) > 5 )                          // 必须满足 0 <= v15[v2] <= 6
29      explode_bomb(a1, v15);
30    if ( ++v2 == 6 )                              // 使这个 while 循环只循环 6
次
31      break;
32    v3 = v2;
33    do
34    {
35      if ( *v1 == v15[v3] )                       // 必须满足输入的 6 个数都不相同
36        explode_bomb(a1, v15);
37      ++v3;
38    }
39    while ( v3 <= 5 );
40    ++v1;
41  }
42  v4 = v15;
43  do
44  {
45    *v4 = 7 - *v4;                                // 将每一个数改为用 7 减去它本身
46    v4 += 4;
47  }
48  while ( v4 != &v16 );
49  v5 = 0LL;
50  do
51  {
52    v8 = v15[v5 / 4];                             // v8 是我们输入的值，在后面会担
任链表的下标
53    if ( v8 <= 1 )
54    {
55      v6 = &node1;
56    }
57    else
58    {
59      v7 = 1;
60      v6 = &node1;
61      do
62      {
63        v6 = *(v6 + 1);                           // 更新 v6 为下一个节点
64        ++v7;
65      }
66      while ( v7 != v8 );
67    }
68    *(&v17 + 2 * v5) = v6;
69    v5 += 4LL;
70  }
```

```
  while ( v5 != 24 );
  v9 = v17;                                     // 把 node6 的地址赋给 v9
  v10 = &v18;                                   // v18 为 node5 的地址
  for ( i = v17; ; i = v12 )                    // 这部分是把链表逆连
  {
    v12 = *v10;
    *(i + 8) = *v10;
    v10 += 8;
    if ( v10 == &v19 )
      break;
  }
  *(v12 + 8) = 0LL;
  v13 = 5;
  do
  {
    result = **(v9 + 8);
    if ( *v9 < result )                         // 必须满足 v9 下一个节点的地址上的值小于等于 v9 地址上的值
      explode_bomb(a1, &v19);                   // 所以这个函数是对节点内的数据做一个降序排序
    v9 = *(v9 + 8);                             // 而我们要输入的数，就是经过排序后，节点下标的顺序
    --v13;
  }
  while ( v13 );
  return result;
}
```

下面放单链表的内容

```
.data:00000000006032D0                  public node1
.data:00000000006032D0 node1            dd 14Ch                 ; DATA XREF: phase_6:loc_401183↑o
.data:00000000006032D0                                          ; phase_6+B0↑o
.data:00000000006032D4                  dd 1
.data:00000000006032D8                  dd 6032E0h
.data:00000000006032DC                  dd 0
.data:00000000006032E0                  public node2
.data:00000000006032E0 node2            dd 0A8h
.data:00000000006032E4                  dd 2
.data:00000000006032E8                  dd 6032F0h
.data:00000000006032EC                  dd 0
.data:00000000006032F0                  public node3
.data:00000000006032F0 node3            dd 39Ch
.data:00000000006032F4                  dd 3
.data:00000000006032F8                  dd 603300h
.data:00000000006032FC                  dd 0
.data:0000000000603300                  public node4
.data:0000000000603300 node4            dd 2B3h
.data:0000000000603304                  dd 4
.data:0000000000603308                  dd 603310h
.data:000000000060330C                  dd 0
.data:0000000000603310                  public node5
.data:0000000000603310 node5            dd 1DDh
.data:0000000000603314                  dd 5
.data:0000000000603318                  dd 603320h
```

```
.data:000000000060331C                  dd 0
.data:0000000000603320                  public node6
.data:0000000000603320 node6            dd 1BBh
.data:0000000000603324                  dd 6
.data:0000000000603328                  dd 0
.data:000000000060332C                  dd 0
```

可以看到，如果要对节点内第一个元素进行降序排序，又因为之前每个数都被 7 减了

所以我们要输入的节点顺序在降序排序后，都要再分别被 7 减，经过排序后节点的排列顺序就是 `3 4 5 6 1 2`

那么我们要输入的数据就是 `4 3 2 1 6 5`

secret_phase

用 `ALT + T` 在 IDA 找到它的调用函数是 phase_defused 函数

```
unsigned __int64 phase_defused()
{
  char v1; // [rsp+8h] [rbp-70h]
  char v2; // [rsp+Ch] [rbp-6Ch]
  char v3; // [rsp+10h] [rbp-68h]
  unsigned __int64 v4; // [rsp+68h] [rbp-10h]

  v4 = __readfsqword(0x28u);
  if ( num_input_strings == 6 )
  {
    if ( __isoc99_sscanf(&unk_603870, "%d %d %s", &v1, &v2) == 3 &&
!strings_not_equal(&v3, "DrEvil") )
    {
      puts("Curses, you've found the secret phase!");
      puts("But finding it and solving it are quite different...");
      secret_phase("But finding it and solving it are quite different...");
    }
    puts("Congratulations! You've defused the bomb!");
  }
  return __readfsqword(0x28u) ^ v4;
}
```

发现这个条件之一是要完成前 6 个挑战，另外就是需要在输入两个整型数字的后面输入字符串 `DrEvil` 才能打开挑战

而且我们能发现前面符合输入两个整型数字且能成功开启挑战的只有第四关，接下来分析隐藏挑战关的源码

```
unsigned __int64 __fastcall secret_phase(__int64 a1)
{
  const char *v1; // rdi
  unsigned int v2; // ebx

  v1 = read_line();
  v2 = strtol(v1, 0LL, 10);
  if ( v2 - 1 > 0x3E8 )
    explode_bomb(v1, 0LL);
  if ( fun7(&n1, v2) != 2 )
    explode_bomb(&n1, v2);
```

```
  puts("Wow! You've defused the secret stage!");
  return phase_defused();
}
```

接下来看 fun7 函数的源码

```
signed __int64 __fastcall fun7(__int64 a1, int a2)
{
  signed __int64 result; // rax

  if ( !a1 )
    return 0xFFFFFFFFLL;
  if ( *a1 > a2 )
    return 2 * fun7(*(a1 + 8));
  result = 0LL;
  if ( *a1 != a2 )
    result = 2 * fun7(*(a1 + 16)) + 1;
  return result;
}
```

最终要让 result 的值为 2，也就是说不能使 `*a1 == a2` 成立

在 secret_phase 函数中，还限制了 a2 的值不能大于 1001

那么非递归情况下，最终 result 的值一定会是 0，也就是说我们要让这个 0 在不断地返回递归函数的时候变成 2

那倒数第二步一定是通过了 `*a1 > a2` 的条件，倒数第三步（正数第一步）一定是通过了 `*a1 != a2` 也就是 `*a1 < a2` 的条件

那先看 a1 在内存里的数值吧

```
.data:00000000006030F0                 public n1
.data:00000000006030F0 n1              dq 24h                  ; DATA XREF:
secret_phase+2C↑o
.data:00000000006030F8                 dq offset n21
.data:0000000000603100                 dq offset n22
.data:0000000000603108                 dq 0
.data:0000000000603110                 public n21
.data:0000000000603110 n21             dq 8                    ; DATA XREF:
.data:00000000006030F8↑o
.data:0000000000603118                 dq offset n31
.data:0000000000603120                 dq offset n32
.data:0000000000603128                 dq 0
.data:0000000000603130                 public n22
.data:0000000000603130 n22             dq 32h                  ; DATA XREF:
.data:0000000000603100↑o
.data:0000000000603138                 dq offset n33
.data:0000000000603140                 dq offset n34
.data:0000000000603148                 dq 0
.data:0000000000603150                 public n32
.data:0000000000603150 n32             dq 16h                  ; DATA XREF:
.data:0000000000603120↑o
.data:0000000000603158                 dq offset n43
.data:0000000000603160                 dq offset n44
.data:0000000000603168                 dq 0
.data:0000000000603170                 public n33
```

```
.data:0000000000603170 n33             dq 2Dh                  ; DATA XREF:
.data:0000000000603138↑o
.data:0000000000603178                 dq offset n45
.data:0000000000603180                 dq offset n46
.data:0000000000603188                 dq 0
.data:0000000000603190                 public n31
.data:0000000000603190 n31             dq 6                    ; DATA XREF:
.data:0000000000603118↑o
.data:0000000000603198                 dq offset n41
.data:00000000006031A0                 dq offset n42
.data:00000000006031A8                 dq 0
.data:00000000006031B0                 public n34
.data:00000000006031B0 n34             dq 6Bh                  ; DATA XREF:
.data:0000000000603140↑o
.data:00000000006031B8                 dq offset n47
.data:00000000006031C0                 dq offset n48
.data:00000000006031C8                 dq 0
.data:00000000006031D0                 public n45
.data:00000000006031D0 n45             dq 28h                  ; DATA XREF:
.data:0000000000603178↑o
.data:00000000006031F0                 public n41
.data:00000000006031F0 n41             dq 1                    ; DATA XREF:
.data:0000000000603198↑o
.data:0000000000603210                 public n47
.data:0000000000603210 n47             dq 63h                  ; DATA XREF:
.data:00000000006031B8↑o
.data:0000000000603230                 public n44
.data:0000000000603230 n44             dq 23h                  ; DATA XREF:
.data:0000000000603160↑o
.data:0000000000603250                 public n42
.data:0000000000603250 n42             dq 7                    ; DATA XREF:
.data:00000000006031A0↑o
.data:0000000000603270                 public n43
.data:0000000000603270 n43             dq 14h                  ; DATA XREF:
.data:0000000000603158↑o
.data:0000000000603290                 public n46
.data:0000000000603290 n46             dq 2Fh                  ; DATA XREF:
.data:0000000000603180↑o
.data:00000000006032B0                 public n48
.data:00000000006032B0 n48             dq 3E9h
```

这题我没动脑子，肯定最后进入这些只有一个数字的地址，挨个试

然后试出来 n43 也就是 0x14 满足要求，完毕

解题脚本

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from pwn import *

context(arch='i386', endian='el', os='linux')
# context.log_level = 'debug'
p = process(['./bomb'])
elf = ELF('./bomb', checksec=False)

# gdb.attach(p, "b *0x401225\nc")
```

```
p.sendlineafter('nice day!\n', 'Border relations with Canada have never been better.')
p.sendlineafter('next one?\n', '1 2 4 8 16 32')
p.sendlineafter('going!\n', '7 327')
p.sendlineafter('there!\n', '7 0 DrEvil')
p.sendlineafter('one.\n', '\x09\x0f\x0e\x05\x06\x07')
p.sendlineafter('next...\n', '4 3 2 1 6 5')
p.sendlineafter('nt...\n', '20')
p.interactive()
```