1. 查看 `bomb.c`，大意就是说要通过 6 个阶段的“考验“ 才能顺利拆除炸弹的雷管，否则它就会爆炸。

2. 阶段 1

```
__int64 __fastcall phase_1(__int64 a1)
{
  __int64 result; // rax

  result = strings_not_equal(a1, "Border relations with Canada have never been better.");
  if ( (_DWORD)result )
    explode_bomb();
  return result;
}
```

阶段 1 需要输入的是 `Border relations with Canada have never been better.`

3. 阶段 2

```
__int64 __fastcall phase_2(__int64 a1)
{
  __int64 result; // rax
  char *now; // rbx
  int v3; // [rsp+0h] [rbp-38h]
  char v4; // [rsp+4h] [rbp-34h]
  char v5; // [rsp+18h] [rbp-20h]

  // 这里 IDA 的逆向在细节上有点问题，但基本思想没错，其实就是先限制 v1 = 1, 然后 now 指针从 v2 开始遍历。
  read_six_numbers(a1, (__int64)&v3);           // 要求输入 6 个数, v1, v2, ..., v6
  if ( v3 != 1 )                                // v3 = 1
    explode_bomb();
  now = &v4;                                    // now 指针一开始指向 v4
  do
  {                                             // 记 now 指针指向的数字为 now_num, now_num 的前一个为 pre_num
    result = (unsigned int)(2 * *((_DWORD *)now - 1));// 要求 pre_num*2 == now_num
    if ( *(_DWORD *)now != (_DWORD)result )
      explode_bomb();
    now += 4;                                   // now 指针指向下一个数字
  }
  while ( now != &v5 );
  return result;
}
```

阶段 2 需要输入的是 `1 2 4 8 16 32`

4. 阶段 3

```
  if ( (signed int)__isoc99_sscanf(a1, "%d %d", &v6, &v7, a5) <= 1 )
    explode_bomb();
  switch ( v6 )
  {
    case 0:
      result = 207LL;
      break;
    case 1:
      result = 311LL;
      break;
    case 2:
      result = 707LL;
      break;
    case 3:
      result = 256LL;
      break;
    case 4:
      result = 389LL;
      break;
    case 5:
      result = 206LL;
      break;
    case 6:
      result = 682LL;
      break;
    case 7:
      result = 327LL;
      break;
    default:
      explode_bomb();
      return result;
  }
  if ( (_DWORD)result != v7 )
    explode_bomb();
  return result;
}
```

要求输入 v6 和 v7，然后 result 会根据 v6 来赋值，要求 result == v7。我选择 `0 207`。

5. 阶段 4

```
__int64 __fastcall phase_4(__int64 a1, __int64 a2, __int64 a3, __int64 a4, __int64 a5)
{
  __int64 result; // rax
  unsigned int v6; // [rsp+8h] [rbp-10h]
  int v7; // [rsp+Ch] [rbp-Ch]

  if ( (unsigned int)__isoc99_sscanf(a1, "%d %d", &v6, &v7, a5) != 2 || v6 > 0xE )
    explode_bomb();
  result = func4(v6, 0LL, 14LL);
  if ( (_DWORD)result || v7 )
    explode_bomb();
  return result;
}
```

要求输入 v6 和 v7，并且 v7 等于 0，而 v6 输入函数 func4() 后得到的结果值也为 0。

```
__int64 __fastcall func4(__int64 v6, __int64 num1, __int64 num2)
{
  signed int v3; // ecx
  __int64 result; // rax

  v3 = ((signed int)num2 - (signed int)num1) / 2 + num1;
  if ( v3 > (signed int)v6 )
    return 2 * (unsigned int)func4(v6, num1, (unsigned int)(v3 - 1));
  result = 0LL;
  if ( v3 < (signed int)v6 )
    result = 2 * (unsigned int)func4(v6, (unsigned int)(v3 + 1), num2) + 1;
  return result;
}
```

`num1 = 0, num2 = 14`，这是一个递归函数，但只要稍加分析就能发现只要令 `v6 = 7` 即可返回 0，不用递归。

6. 阶段 5

```
unsigned __int64 __fastcall phase_5(__int64 a1)
{
  __int64 v1; // rax
  char v3[6]; // [rsp+10h] [rbp-18h]
  char v4; // [rsp+16h] [rbp-12h]
  unsigned __int64 v5; // [rsp+18h] [rbp-10h]

  v5 = __readfsqword(0x28u);
  if ( (unsigned int)string_length() != 6 )
    explode_bomb();
  v1 = 0LL;
  do
  {
    // array = "maduiersnfotvbyl"
    v3[v1] = array[*(_BYTE *)(a1 + v1) & 0xF];
    ++v1;
  }
  while ( v1 != 6 );
  // "flyers" 对应 array 中的下标为 [9, 15, 14, 5, 6, 7]。
  // 对照一下 ascii 表，找出低 4 位是这 6 个数字的字符就行。
  v4 = 0;
  if ( (unsigned int)strings_not_equal(v3, "flyers") )
    explode_bomb();
  return __readfsqword(0x28u) ^ v5;
}
```

输入 `ionefg` 即可。

7. 阶段 6

这种情况下直接看代码就不如去 peda 一点一点动调。这个阶段其实就是在做如下几件事：

a. 输入 6 个互不相同的数字，$a_1, a_2, a_3, a_4, a_5, a_6$；

b. 求 $b_i = 7 - a_i, i = 1, 2, 3, 4, 5, 6$；

c. 求 $c_i = F(b_i), i = 1, 2, 3, 4, 5, 6$。其中

$$F(x) = \begin{cases} 0x14c, x = 1 \\ 0xa8, x = 2 \\ 0x39c, x = 3 \\ 0x2b3, x = 4 \\ 0x1dd, x = 5 \\ 0x1bb, x = 6 \end{cases}$$

d. 要求 c_i 数组单调递减，最终的输入为 `4 3 2 1 6 5`