ICS Lab2-bomb lab

姓名：陈锐林 学号：21307130148

一、推演过程

1. phase_1（代码为完整的 phase_1 函数）

```
Dump of assembler code for function _Z7phase_1Pc:
0x0000555555555b12 <+0>:     endbr64
0x0000555555555b16 <+4>:     push   %rdx
0x0000555555555b17 <+5>:     movslq 0x2522(%rip),%rsi     # 0x555555558040 <phase_1_offset>
0x0000555555555b1e <+12>:    lea    0x253b(%rip),%rax     # 0x555555558060 <w1>
0x0000555555555b25 <+19>:    add    %rax,%rsi
0x0000555555555b28 <+22>:    callq  0x555555555a60 <_Z16string_not_equalPcS_>
0x0000555555555b2d <+27>:    test   %al,%al
0x0000555555555b2f <+29>:    jne    0x555555555b36 <_Z7phase_1Pc+36>
0x0000555555555b31 <+31>:    callq  0x555555555a46 <_Z12explode_bombv>
0x0000555555555b36 <+36>:    pop    %rax
0x0000555555555b37 <+37>:    retq
```

（1）首先观察函数 phase_1，触发炸弹的语句在<+31>，要跳过炸弹，那么%al 应该不为 0；前面没出现过，所以推断是在函数<string_not_equal>里赋值的。

（2）注意到调用函数<string_not_equal>前，对%rsi 先进行了赋值偏移量再加上了%rax；而%rsi 一般表示第二个参数，又是固定的，所以这个应该是题目的谜底，经过以下语句，即可知道答案。（查看%rdi 即发现其为输入的字符串）

```
(gdb) x/s $rsi
0x555555558146 <w1+230>:        "each line is important"
```

（3）事后仔细观察函数<string_not_equal>，可以发现它通过一个循环控制，判断每个字符是否对应，如果都对应，会对%al 赋 1，即满足 phase_1 的条件。

2. phase_2（代码为完整的 phase_2 函数）

```
0x0000555555555b38 <+0>:     endbr64
0x0000555555555b3c <+4>:     push   %rdx
0x0000555555555b3d <+5>:     lea    0x24dc(%rip),%rsi     # 0x555555558020 <phase_2_nums>
0x0000555555555b44 <+12>:    callq  0x555555555ad0 <_Z16read_six_numbersPcPi>
0x0000555555555b49 <+17>:    lea    0x24d0(%rip),%rax     # 0x555555558020 <phase_2_nums>
0x0000555555555b50 <+24>:    mov    0x24e6(%rip),%ecx     # 0x55555555803c <phase_2_nums+28>
0x0000555555555b56 <+30>:    lea    0x14(%rax),%rdx
0x0000555555555b5a <+34>:    mov    (%rax),%esi
0x0000555555555b5c <+36>:    imul   %ecx,%esi
0x0000555555555b5f <+39>:    cmp    %esi,0x4(%rax)
0x0000555555555b62 <+42>:    je     0x555555555b69 <_Z7phase_2Pc+49>
0x0000555555555b64 <+44>:    callq  0x555555555a46 <_Z12explode_bombv>
0x0000555555555b69 <+49>:    add    $0x4,%rax
0x0000555555555b6d <+53>:    cmp    %rax,%rdx
0x0000555555555b70 <+56>:    jne    0x555555555b5a <_Z7phase_2Pc+34>
0x0000555555555b72 <+58>:    pop    %rax
0x0000555555555b73 <+59>:    retq
```

（1）首先观察炸弹在<+44>，所以上一行的%esi 和 0x4（%rax）一定要判等

（2）前面不大理解，不过就是读入 6 个数字，然后储存在%rax 开始的地方，然后对%ecx 赋值，可以查看知道%ecx 就是-3。

（3）从<+34>开始到<+56>是一循环，结束循环条件为%rax 等于%rdx（为第六个数字的地址）；循环流程是从第一个数开始，将其×-3，与下一个数判断，等于就避免引爆炸弹，不等就失败，一个数判断成功就把%rax 增加 4（到下一个数），所以只要输入 6 个数，构成公比为-3 的等比数列即可，如 1 -3 9 -27 81 -243。

（4）仔细观察<read_six_numbers>，发现有对数目的判断，以及调用了库函数。到了 sscanf 之后的函数就很复杂，没有深究。并且不能输入 0。

```
0x0000555555555b06 <+54>:    cmpl   $0x0,(%rbx)   //<read_six_numbers>中，若为 0，此处直接爆炸！
0x0000555555555b09 <+57>:    jne    0x555555555b10 <_Z16read_six_numbersPcPi+64>
0x0000555555555b0b <+59>:    callq  0x555555555a46 <_Z12explode_bombv>
```

3. phase_3（代码只留下了涉及到的，太长了想必 TA 看着都烦）

```
0x0000555555555b93 <+31>:    lea    0xf(%rsp),%rcx       //此处可以推断这三个就存放了输入的参数
0x0000555555555b98 <+36>:    lea    0x10(%rsp),%rdx      //占据 15 16 20，大小说明数据类型
0x0000555555555b9d <+41>:    lea    0x14(%rsp),%r8
0x0000555555555ba2 <+46>:    callq  0x555555555170 <__isoc99_sscanf@plt>
0x0000555555555ba7 <+51>:    cmp    $0x3,%eax          //不等会跳到<157>
0x0000555555555bac <+56>:    cmpl   $0x7,0x10(%rsp)    //大于会跳到<157>
0x0000555555555bb3 <+63>:    mov    0x10(%rsp),%eax
0x0000555555555bb7 <+67>:    lea    0x5f6(%rip),%rdx        # 0x5555555561b4
0x0000555555555bbe <+74>:    movslq (%rdx,%rax,4),%rax
0x0000555555555bc2 <+78>:    add    %rdx,%rax
0x0000555555555bc5 <+81>:    notrack jmpq *%rax
0x0000555555555bd3 <+95>:    cmpl   $0xdd,0x14(%rsp)    //不等会跳到<157>
0x0000555555555bdb <+103>:   mov    $0x62,%al
0x0000555555555bdd <+105>:   je     0x555555555c16 <_Z7phase_3Pc+162>
0x0000555555555c11 <+157>:   callq  0x555555555a46 <_Z12explode_bombv>
0x0000555555555c16 <+162>:   cmp    %al,0xf(%rsp)    //不等会跳到<157>
0x0000555555555c31 <+189>:   add    $0x28,%rsp
0x0000555555555c35 <+193>:   retq
```

（1）首先炸弹在<+157>，要及时避开。

（2）前面不过多解释，<+46>的函数显示：

```
__GI___isoc99_sscanf (s=0x7fffffffdba8 "1 b 221", format=0x55555555603a "%d %c %d") at isoc99_sscanf.c:24
```

告诉我们要输入的格式为 数字 字符 数字。 <+51> 这里判断输入是否为 3 个，不是就炸了。往下判断第一个数是不是小于等于 7，不满足也炸了。

（3）接着走到 <+74> ，此时%eax 为输入的第一个数（如我输入 1），%rdx 为 20，再到 <+81> ，%rax 已为 44。经过 <+81> ， 会跳转到 <+95> ， 此时将 221 与 0x14(%rsp)进行判断，不等会触发炸弹；所以反推可以知道输入的第三个数应该是 221。这步完了之后%al 值为 98。

（4）下一步跳转到<+162>，将输入的第二个字符与 98 做对比，推测应该是比较 ascii 码，不等会引爆炸弹，查表可知第二个字符应该是’b’。

4. phase_4（代码为完整的 phase_4 函数）

```
0x0000555555555c36 <+0>:     endbr64
0x0000555555555c3a <+4>:     mov    %rdi,%rax
0x0000555555555c3d <+7>:     sar    $0x20,%rdi
0x0000555555555c41 <+11>:    push   %rdx
0x0000555555555c42 <+12>:    lea    -0x1(%rdi),%edx
0x0000555555555c45 <+15>:    cmp    $0xd,%edx
0x0000555555555c48 <+18>:    ja     0x555555555c51 <_Z7phase_4l+27>
0x0000555555555c4a <+20>:    dec    %eax
0x0000555555555c4c <+22>:    cmp    $0xd,%eax
0x0000555555555c4f <+25>:    jbe    0x555555555c56 <_Z7phase_4l+32>
0x0000555555555c51 <+27>:    callq  0x555555555a46 <_Z12explode_bombv>
0x0000555555555c56 <+32>:    callq  0x555555555548 <_ZL4hopei>
0x0000555555555c5b <+37>:    cmp    $0x1000000,%eax
0x0000555555555c60 <+42>:    jne    0x555555555c51 <_Z7phase_4l+27>
0x0000555555555c62 <+44>:    pop    %rax
0x0000555555555c63 <+45>:    retq
```

（1）炸弹在<+27>，要避开。看最后几行，解答成功的条件是%eax 等于 4^12

（2）前几行做了几件事：一是把%rdi（输入值）右移 32 位，将原先右边的 32 位放在了%eax。<+15> 的判断，意思是右移后的%rdi 必须小于等于 14。<+22> 的判断意味着%eax(未减一)必须小于等于 14;后面可以知道,递归函数只与%rdi 的左边 32 位有关，所以右边 32 位，取 1/2 即可了。

（3）此后正式进入 hope 这个递归函数。

```
0x0000555555555548 <+0>:     mov    $0x1,%r8d              //每次开始都让%r8d 为 1
0x000055555555554e <+6>:     test   %edi,%edi              //当%edi 为 0 的时候才结束递归，此处即递归基
0x0000555555555550 <+8>:     je     0x555555555575 <_ZL4hopei+45>
0x0000555555555552 <+10>:    push   %rbx
0x0000555555555553 <+11>:    mov    %edi,%ebx
0x0000555555555555 <+13>:    sar    %edi                   //将%edi 向右移 1 位，即/2
0x0000555555555557 <+15>:    callq  0x555555555548 <_ZL4hopei>  //进入下一轮递归
```

```
0x000055555555555c <+20>:    mov    %eax,%r8d
0x000055555555555f <+23>:    imul   %eax,%r8d                  //这步和上面那两步就是让%r8d 为%eax 的平方
0x0000555555555563 <+27>:    and    $0x1,%bl                   //按位与，这里就有点快速幂的特征了
0x0000555555555566 <+30>:    je     0x555555555570 <_ZL4hopei+40>
0x0000555555555568 <+32>:    lea    0x0(,%r8,4),%r8d           //%r8d 变成原来的 4 倍
0x0000555555555570 <+40>:    mov    %r8d,%eax                  //%eax 即前面要得到的返回值!
0x0000555555555573 <+43>:    pop    %rbx
0x0000555555555574 <+44>:    retq
0x0000555555555575 <+45>:    mov    %r8d,%eax
0x0000555555555578 <+48>:    retq
```

以具体例子来说明，如果投进去的参数是$(1101)_2$，那么只有当$(1101)_2$向右移 4 次（1）位后，会直接返回 1；然后根据第一位是 1，所以执行平方且×4 的操作，为 4^1；接着返回，第二位是 1，先平方、再×4，为 4^3；第三位是 0，仅平方，为 4^6;第 4 位为 1，先平方、再×4，得 4^13。所以这个函数就是求 4 的幂次，幂次即主函数传递的参数。所以就得知了，原来%rdi 的左边 32 位，即 12。构造一个数 16^8 * 12 + 2（51539607554），就满足题意了。

5. phase_5（代码为节选）

```
0x0000555555555c68 <+4>:     sub    $0x48,%rsp    //分配空间
.....（1）
0x0000555555555cb8 <+84>:    cmp    $0x1,%rdi
0x0000555555555cbc <+88>:    jg     0x555555555cc5 <_Z7phase_5lll+97>
0x0000555555555cbe <+90>:    lea    0x28(%rsp),%rdi
0x0000555555555cc3 <+95>:    jmp    0x555555555ceb <_Z7phase_5lll+135>
0x0000555555555ceb <+135>:   callq  0x555555555a93 <_Z13run_lock_testP8baselockii>
```

（1）省略号部分进行赋值操作，经过操作后有：
%rsp + 24 : <lock1+16>; %rsp + 56 : %fs:0x28; %rsp + 8 : <baselock+16>
%rsp + 40 : <lock2+16>; %rsp + 16/32/48 : 0xffffffff00000000

（2）进入一个判断语句，<+88>表示若输入的第一个数小于等于 1，就不会跳到<+97>，而是在对%rdi 赋值后跳到<+135>；那么不妨就让第一个输入数为 1，然后往下走。此后，%rdi 为<lock2 + 16>，进入函数

（3）函数 run_lock_test，前段

```
0x0000555555555a97 <+4>:     sub    $0x18,%rsp
0x0000555555555a9b <+8>:     mov    (%rdi),%rax
0x0000555555555a9e <+11>:    mov    %edx,0xc(%rsp)
0x0000555555555aa2 <+15>:    mov    %esi,0x8(%rsp)
0x0000555555555aa6 <+19>:    mov    %rdi,(%rsp)
0x0000555555555aaa <+23>:    callq  *(%rax)
```

操作依次为开空间（24），对%rsp + 12/8 赋输入的第三个/第二个数；
调用%rax 指向的函数，即（2）中提到的%rdi（<lock2+16>）

（4）进入函数 lock2::acquire(int)，前半段为：

```
0x00005555555555fe <+4>:     push   %rbp
0x00005555555555ff <+5>:     mov    %esi,%ebp
0x0000555555555601 <+7>:     push   %rbx
0x0000555555555602 <+8>:     mov    %rdi,%rbx
0x0000555555555605 <+11>:    push   %rcx
0x0000555555555606 <+12>:    mov    (%rdi),%rax
0x0000555555555609 <+15>:    callq  *0x10(%rax)
```

将%esi 所对应的值赋给%ebp（即输入的第二个值）

将%rdi 所对应的值赋给%rbx（<lock2 + 16>）；
最后把（%rdi)赋给%rax，调用函数（%rax)+16；根据前文应该就是%fs:0x28

（5）进入函数 lock2::is_holding(int)

```
0x0000555555555e52 <+4>:     mov    0xc(%rdi),%edx

0x0000555555555e55 <+7>:     cmp    %esi,%edx

0x0000555555555e57 <+9>:     not    %edx

0x0000555555555e59 <+11>:    sete   %al

0x0000555555555e5c <+14>:    shr    $0x1f,%edx

0x0000555555555e5f <+17>:    and    %edx,%eax

0x0000555555555e61 <+19>:    retq
```

首先，对%edx 进行赋值，之后与%esi 进行比较；
再对%edx 取反，对%al 赋值 1；将%edx 逻辑右移 31 位后与%eax 按位与。
根据前面的值。最后返回（4）中的 acquire 函数

（6）前几行意思是如果%eax 不等于 0，就跳到<+60>，直接返回；（从后来可知）那么就会遇到后半段的几行代码（%eax 在判断后被置 0 了），无法跳到<+46>而导致失败，所以肯定应有%eax 等于 0。继续在 acquire 函数中走，

```
0x000055555555560c <+18>:    mov    %eax,%r8d

0x000055555555560f <+21>:    xor    %eax,%eax

0x0000555555555611 <+23>:    test   %r8b,%r8b

0x0000555555555614 <+26>:    jne    0x555555555636 <_ZN5lock27acquireEi+60>

0x0000555555555616 <+28>:    mov    $0x1,%edx

0x000055555555561b <+33>:    mov    %edx,%eax

0x000055555555561d <+35>:    lock xchg %eax,0x8(%rbx) //!! ??

0x0000555555555621 <+39>:    test   %eax,%eax

0x0000555555555623 <+41>:    jne    0x55555555561b <_ZN5lock27acquireEi+33>

0x0000555555555625 <+43>:    mov    (%rbx),%rax

0x0000555555555628 <+46>:    mov    %rbx,%rdi

0x000055555555562b <+49>:    callq  *0x18(%rax)

0x000055555555562e <+52>:    mov    %ebp,0xc(%rbx)

0x0000555555555631 <+55>:    mov    $0xf,%eax

……retq
```

进行的操作有，对%edx/%eax 赋值为 1，<+35>这里有点不理解，但可知会把%eax 再置 0，然后调用（%rax+24)指向的函数，即 lock2::mem_sync()，没有具体代码，返回。
最后再对 %rbx+12 赋 %ebp，即前面提到的输入的第二个参数；对%eax 赋 15 后返回。

（7）回到 run_lock_test 函数

```
0x0000555555555aac <+25>:    mov    (%rsp),%rdi
0x0000555555555ab0 <+29>:    mov    0x8(%rsp),%esi
0x0000555555555ab4 <+33>:    test   %eax,%eax
0x0000555555555ab6 <+35>:    mov    0xc(%rsp),%edx
0x0000555555555aba <+39>:    jne    0x555555555ac1 <_Z13run_lock_testP8baselockii+46>
0x0000555555555abc <+41>:    callq  0x555555555a46 <_Z12explode_bombv>
0x0000555555555ac1 <+46>:    mov    (%rdi),%rax
0x0000555555555ac4 <+49>:    callq  *0x8(%rax)
```

前面两行赋值，（应该是出于对寄存器内容的保护，不多谈）；
接着对%eax 的值进行判断，等于 0 会触动炸弹（这就是（6）中一开始不能直接返回的原因）；接着再对%rax 赋值后就调用函数；

（8）进入函数 lock2::release()

```
0x000055555555551a <+4>:     push   %rbp
0x000055555555551b <+5>:     mov    %edx,%ebp
0x000055555555551d <+7>:     push   %rbx
0x000055555555551e <+8>:     mov    %rdi,%rbx
0x0000555555555521 <+11>:    push   %rcx
0x0000555555555522 <+12>:    mov    (%rdi),%rax
0x0000555555555525 <+15>:    callq  *0x10(%rax)
```

将%edx 赋给%ebp（输入第三个数），%rdi 赋给%rbx，(%rdi)赋给%rax，调用函数

（9）进入函数 lock2::is_holding(int)
函数和(5)相同，此时%edx 和%esi 所指向的都是第二个参数，%al 被置 1，但如果%esi 是负数，取反后会使得%eax 为 0，那么在 release 里会使得%eax 为 0，导致回到 run_lock_test 时，炸弹爆炸。

（10）返回函数 lock2::release（）

```
0x0000555555555525 <+15>:    callq  *0x10(%rax)

0x0000555555555528 <+18>:    cmp    $0xf,%ebp

0x000055555555552b <+21>:    jne    0x555555555542 <_ZN5lock27releaseEii+44>

0x000055555555552d <+23>:    dec    %al

0x000055555555552f <+25>:    jne    0x555555555542 <_ZN5lock27releaseEii+44>

0x0000555555555531 <+27>:    mov    $0xffffffff,%eax

0x0000555555555536 <+32>:    shl    $0x20,%rax

0x000055555555553a <+36>:    mov    %rax,0x8(%rbx)

0x000055555555553e <+40>:    mov    $0x1,%al

0x0000555555555540 <+42>:    jmp    0x555555555544 <_ZN5lock27releaseEii+46>

0x0000555555555542 <+44>:    xor    %eax,%eax
```

这里对输入的第三个数判断，不等于 15 就直接返回了，那么%al 就不能置 1，导致触发炸弹，所以第三个一定输入 15。

（10）回到函数 run_lock_test 函数

```
0x0000555555555ac7 <+52>:    test   %al,%al

0x0000555555555ac9 <+54>:    je     0x555555555abc <_Z13run_lock_testP8baselockii+41>
```

<+41>即炸弹所在，所以可以反推，此处%al 一定不能为 0；之后正常返回

（11）终于回到了 phase_5 函数

```
0x0000555555555cf0 <+140>:   mov    0x38(%rsp),%rax

0x0000555555555cf5 <+145>:   xor    %fs:0x28,%rax

0x0000555555555cfe <+154>:   je     0x555555555d05 <_Z7phase_5lll+161>

0x0000555555555d00 <+156>:   callq  0x5555555551a0 <__stack_chk_fail@plt>

0x0000555555555d05 <+161>:   add    $0x48,%rsp

0x0000555555555d09 <+165>:   retq
```

只要异或后不为 0 即正确返回。

（12）总结：题解为第一位数选择不同走向，输入小于等于 1 的数字或 1 到 3 的数字，会进入不同的 lock。如果输入 1 的话，较重要的是第三个数字要输入 15，以让%al 正确赋值，第二个数字只要是非负数就可以了。
1 2 15 / 1 3 15 / 1 15 15/ 1 100 15 都可行。

6. phase_6

```
0x0000555555555d0e <+4>:     push   %rdx
0x0000555555555d0f <+5>:     callq  0x555555555a7c <_Z10string_lenPc>
0x0000555555555d14 <+10>:    lea    0x27f0(%rip),%rdx        # 0x55555555850b <w2+11>
0x0000555555555d1b <+17>:    mov    %eax,%r8d
0x0000555555555d1e <+20>:    xor    %eax,%eax
0x0000555555555d20 <+22>:    cmp    $0x6,%r8d
0x0000555555555d24 <+26>:    je     0x555555555d2b <_Z7phase_6Pc+33>
0x0000555555555d26 <+28>:    callq  0x555555555a46 <_Z12explode_bombv>
0x0000555555555d2b <+33>:    mov    (%rdi,%rax,1),%cl
0x0000555555555d2e <+36>:    inc    %rax
0x0000555555555d31 <+39>:    mov    %cl,-0x1(%rax,%rdx,1)
0x0000555555555d35 <+43>:    cmp    $0x7,%rax
0x0000555555555d39 <+47>:    jne    0x555555555d2b <_Z7phase_6Pc+33>
0x0000555555555d3b <+49>:    mov    $0x11,%esi
0x0000555555555d40 <+54>:    lea    0x27b9(%rip),%rdi        # 0x555555558500 <w2>
0x0000555555555d47 <+61>:    callq  0x555555555765 <_Z31build_candidate_expression_treePci>
0x0000555555555d4c <+66>:    mov    $0x11,%edx
0x0000555555555d51 <+71>:    lea    0x26ac(%rip),%rdi        # 0x555555558404 <ans+132>
0x0000555555555d58 <+78>:    mov    %rax,%rsi
0x0000555555555d5b <+81>:    callq  0x555555555922 <_Z28compare_answer_and_candidateP9tree_nodeS0_i>
0x0000555555555d60 <+86>:    test   %al,%al
0x0000555555555d62 <+88>:    je     0x555555555d26 <_Z7phase_6Pc+28>
0x0000555555555d64 <+90>:    pop    %rax
0x0000555555555d65 <+91>:    retq
```

（1）从<+4>到<+28>：调用了函数 string_len，得到了输入字符串的长度；再进行比较，如果不等于 6，就会触发炸弹，这里可以得到应输入 6 个字符。

（2）从<+33>到<+47>这个循环，把已有的 11 个字符，与输入的 6 个字符拼在了一起，在结束后用 x/s $rdi，可以得到，前面的是“(1+2)*(9-0)”。

（3）两个函数 build_candidate_expression_tree 和 compare_answer_and_candidate 可以得知就是构造表达式树，然后进行比较。此处可以直接调用函数。假设我输入的是*(9+7)：

那么在对比函数之前，call (void)print_candidate_tree_inorder($rax)，并且call printf("\n"),之后会得到 1 + 2 * 9 - 0 * 9 + 7；在<+71>后

call (void)print_answer_tree_inorder($rdi)，并且call printf("\n"),之后会得到 1 + 2 * 9 - 0 / 3 - 4。
所以得到输入应为 /(3-4)。

7. secret_phase

如果按照上面的输入，会提示：
But isn't something... missing? Perhaps something was overlooked?
说明没有触发 secret_phase，在反复查看后发现，在 phase_5 中，有这么一段：

```
0x0000555555555cd2 <+110>:   cmp    $0x8,%rdi

0x0000555555555cd6 <+114>:   lea    0x8(%rsp),%rdi

0x0000555555555cdb <+119>:   jne    0x555555555ceb <_Z7phase_5lll+135>

0x0000555555555cdd <+121>:   callq  0x555555555a93 <_Z13run_lock_testP8baselockii>

0x0000555555555ce2 <+126>:   movb   $0x79,0x234f(%rip)       # 0x555555558038 <phase_2_nums+24>

0x0000555555555ce9 <+133>:   jmp    0x555555555cf0 <_Z7phase_5lll+140>
```

<+126>这行是选择其他路径不会遇到的，那么重新考查 phase_5，得到输入 8 12 4080（其中第二个为任意正数），能正确完成 phase_5，并且

在完成 phase_6 后就会跳到 secret_phase，如下：

```
0x0000555555555d6a <+4>:     cvtsi2ss %rdi,%xmm0

0x0000555555555d6f <+9>:     addss  %xmm0,%xmm0

0x0000555555555d73 <+13>:    addss  0x475(%rip),%xmm0       # 0x5555555561f0

0x0000555555555d7b <+21>:    cvtss2sd %xmm0,%xmm0

0x0000555555555d7f <+25>:    comisd 0x471(%rip),%xmm0       # 0x5555555561f8

0x0000555555555d87 <+33>:    jae    0x555555555d97 <_Z12secret_phasel+49>

0x0000555555555d89 <+35>:    movsd  0x46f(%rip),%xmm1       # 0x555555556200

0x0000555555555d91 <+43>:    comisd %xmm0,%xmm1

0x0000555555555d95 <+47>:    jb     0x555555555d9d <_Z12secret_phasel+55>

0x0000555555555d97 <+49>:    push   %rax

0x0000555555555d98 <+50>:    callq  0x555555555a46 <_Z12explode_bombv>

0x0000555555555d9d <+55>:    retq
```

（1）粗略看几处跳转，<+33>时不能跳转，<+47>时必须跳转。

（2）第一行，把输入转为浮点数 float，插入到%xmm0 中；第二行，浮点数加法，×2；第三行，将 0x475(%rip)加到%xmm0；第四行，转为 double;第五行，与 0x471(%rip)作比较，此处即要求要小于 0x471(%rip)；第六行不跳转；第七行对 double 数%xmm1 赋值;第八行判断;第九行要跳转,意思为%xmm1 要小于%xmm0。
（3）最后通过查看各个变量的值，联系 int 到 float 的类型转换，可以得到设 x 为输入后转化的 float 值；即有：2 * x + 0x475(%rip) <= 0x471(%rip) 并且 2 * x + 0x475(%rip) > 0x46f(%rip)。得出，满足的答案为 1905!
至此，炸弹都被拆除。

二、炸弹成功拆除的截图：

```
Starting program: /mnt/d/lab/bomb
You have 6 phases with which to blow yourself up. Have a nice day!
each line is important
Phase 1 defused. How about the next one?
1 -3 9 -27 81 -243
That's number 2.  Keep going!
1 b 221
Halfway there!
51539607554
So you got that one.  Try this one.
8 12 4080
Good work!  On to the next...
/(3-4)
You've enter the float point world! It's not hard o(*^ _ ^*)m
1905
Congratulations!
```

三、引用（无）

四、意见+建议：

需要自己主动 call 的函数，如 print_candidate_tree_inorder;
一开始没注意（闷着头调 gdb），所以或许可以说下这些还是挺重要的，早点注意到可能会比较高效。

感谢老师和助教，有被虐到！