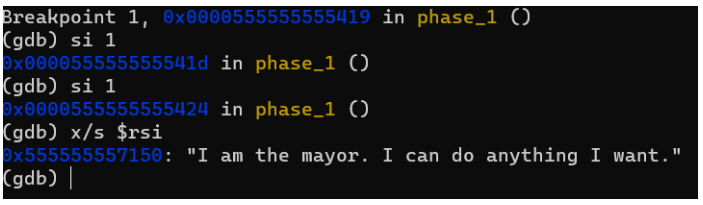
吴佳启bomblab解题报告

phase_1

1. 首先我们从主函数入手，我们先调用了read_line函数，仔细查看可以发现它的功能和描述一样，读入一个字符串，然后将返回值存在rax里，接下来赋给rdi，然后调用phase_1函数。

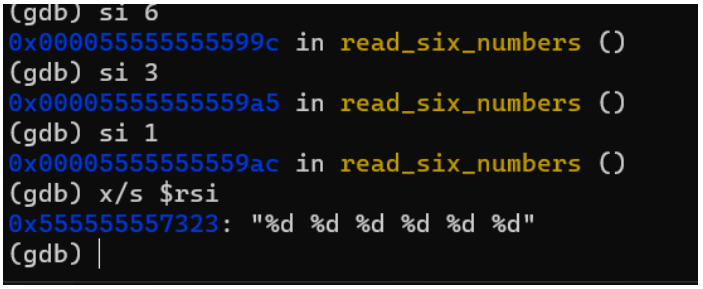
```
131c:       e8 a8 06 00 00          call   19c9 <read_line>
1321:       48 89 c7                mov    %rax,%rdi
1324:       e8 f0 00 00 00          call   1419 <phase_1>
```

2. phase_1部分代码比较简单，我们可以先查看这个函数的大概思路：它调用了“strings_not_equal”的函数。然后判断返回值eax，如果非零则引爆炸弹，否则则完成炸弹一的解除。
3. 函数整体逻辑十分简单，主要问题在于“strings_not_equal”函数的实现，在这之前我们只将rsi的值进行了初始化，则“strings_not_equal”函数只有两个参数，只需要将rsi和rdi的值进行比较，如果不同则返回非零。进入“strings_not_equal”函数内部，我们发现它将rdi和rsi的长度进行比较，然后再一位一位进行比较，这和我们的设想相同。那接下来只需要知道rsi里的内容究竟是什么了，我使用"x/s $rsi"的命令获得答案："I am the mayor. I can do anything I want."，完成炸弹一的拆解。



phase_2

1. 我们可以发现开始的时候调用了"read_six_numbers"函数，意味这读入6个数字，为了确定，我们进入这个函数，发现他实际上是在调用sscanf函数来读入数据，查看rsi里的值，发现读入的是六个"%d"类型的数据，彼此之间用空格间隔开，由此确定了输入的格式。



2. 进入函数内部

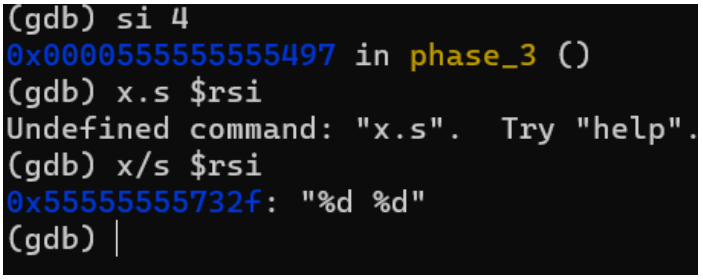
- 首先他的第一个元素需要大于等于0，否则引爆炸弹

```
1447:       83 3c 24 00             cmpl   $0x0,(%rsp)
144b:       78 0a                   js     1457 <phase_2+0x1e>
```

- 不难发现，接下来从146a开始，我们进入了一个循环，这个循环会遍历一边我们读入后放到栈上的六个数，注意到这里的ebx每次会有一个加一的操作，我们可以得到递推公式：i + 1 + a[i] = a[i + 1]。只有满足这个条件才能进入下一层循环，否则会产生爆炸
- 构造数据，完成代码框架认知后只需要构造数据即可，简单起见，我让a[0] = 1，然后可以得到序列"1 2 4 7 11 16"，满足题意，得到最后的答案。

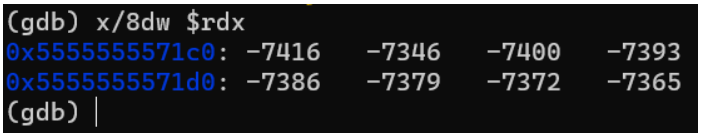
phase_3

1. 查看读入，阅读汇编代码我们发现，这里是先调用了sscanf函数进行数据的读入，然后再进行后续操作，这里我们还是查看rsi内部的值，发现是读入"%d, %d"的两个整数，如果不满足要求则会爆炸。



2. 进入函数

- 我们先判断了读入的第一个参数再无符号的比较情况下是否大于7，如果是则爆炸；
- 接下来代码实现了根据第一参数的不同值进行跳转：具体来说，它将第一个参数存在eax里，然后使用rdx+4*eax来得到一个数，这里我们很自然地发现这是一个数组，接下来我通过x/8dw指令获得了这个数组里的元素，即各个部分的偏移量。



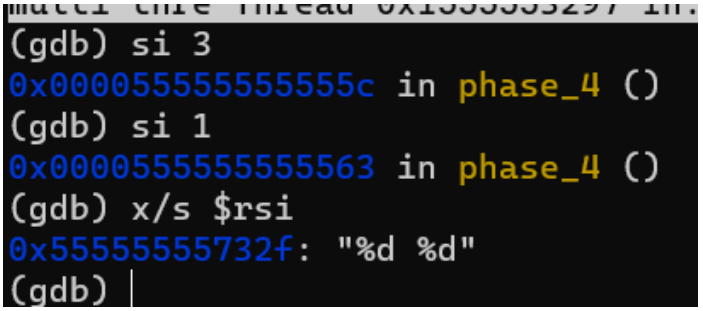
- 接下来我们把这个偏移量在rax上获得地址，这里我们不妨输入第一个参数为0，这样得到rdx最开始的跳转点：14c8这个位置，得到这个位置与其偏移量以后，我们便可以得到跳转的每个位置了

```
    14ad:       8b 44 24 0c             mov    0xc(%rsp),%eax
    14b1:       48 8d 15 08 1d 00 00    lea    0x1d08(%rip),%rdx        # 31c0
<_IO_stdin_used+0x1c0>
    14b8:       48 63 04 82             movslq (%rdx,%rax,4),%rax
    14bc:       48 01 d0                add    %rdx,%rax
    14bf:       ff e0                   jmp    *%rax
```

- 对跳转后的每个指令进行分析，无论跳转后是多少，接下来的操作就是把一个立即数放在eax里，然后跳转至同一处，把这个数和我们读入的第二个参数进行比较即可，如果相等那么解除炸弹，否则爆炸。举个例子比如输入0时，第二个参数的值需要为0x9b，即155, 输入1时第二个值需要为0x5f等等。有多个答案，我这里选择了最简单的0,155。

phase_4

1. 查看读入，和之前一样，我们利用x/s命令查看rdi，得到读入为两个间隔的整数



2. 进入函数

- 首先判断之前读入的数据是否符合规范，然后先判断得到的第一参数是否再无符号数判断下小于0xe，如果小于才进行下一步的判断。
- 接下来我们看到这里给三个寄存器edx,esi,edi分别用e,0,第一个参数进行赋值，然后调用func4函数，判断返回值是否为2b，如果是才进行下一步判断，这里我们查看func4进行计算。

```
  157e:       ba 0e 00 00 00          mov    $0xe,%edx
  1583:       be 00 00 00 00          mov    $0x0,%esi
  1588:       8b 7c 24 0c             mov    0xc(%rsp),%edi
  158c:       e8 8b ff ff ff          call   151c <func4>
  1591:       83 f8 2b                cmp    $0x2b,%eax
  1594:       75 07                   jne    159d <phase_4+0x4f>
```

- 查看func4不难发现，他是一个比较简单的递归函数，出于其考虑条件较多，加上输入值edi有一个大于等于0小于e的限制，我将汇编代码手写更改为以下的c++代码，枚举不同输入来得到答案，最终得到第一参数的值为12.

```
int func(int edx, int esi, int edi) {
    int eax, ebx;
    eax = edx,eax -= esi,ebx = eax;
    if (eax > 0)
        ebx = 0;
    else
        ebx = 1;
    ebx += eax,ebx = ebx / 2,ebx += esi;
    if (ebx > edi) {
        edx = ebx - 1;
        ebx += func(edx, esi, edi);
    }
    else if (ebx < edi) {
        esi = ebx + 1;
        ebx += func(edx, esi, edi);
    }
    return ebx;
}
int main() {
    for (int i = 0; i < 14; i++) {
        if (func(14, 0, i) == 43) {
```

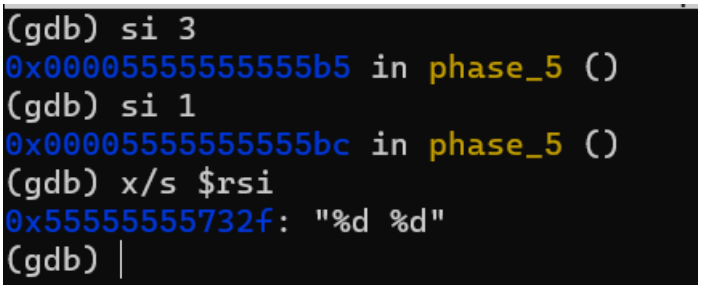
```
            cout << "The answer is " << i << endl;
        }
    }
}
```

- 接着往下就十分简单，只需要满足第二个参数的值也为2b即可，否则无法跳转炸弹爆炸，最终得到答案：12 43.

```
  1596:       83 7c 24 08 2b          cmpl   $0x2b,0x8(%rsp)
  159b:       74 05                   je     15a2 <phase_4+0x54>
  159d:       e8 c0 03 00 00          call   1962 <explode_bomb>
```

phase_5

1. 查看读入，和上面操作类似，这里发现我们还是读入两个间隔的整数，然后判断返回值是否为1，为1再进入下一层。

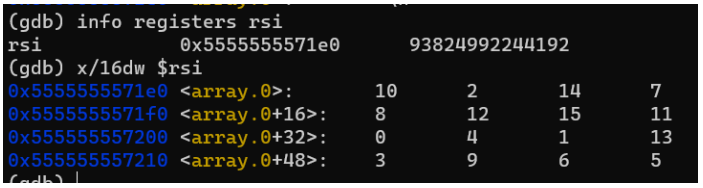


2. 分析函数

- 对输入进行预处理：接下来这五行操作让我思考了很久，他主要目的是在于让eax不为0xf，否则引爆炸弹，同时如果传入的第一个参数大于15， 可以通过这个and操作把他的值放在15以内，方便接下来数组的查询操作。

```
  15cb:       8b 44 24 0c             mov    0xc(%rsp),%eax
  15cf:       83 e0 0f                and    $0xf,%eax
  15d2:       89 44 24 0c             mov    %eax,0xc(%rsp)
  15d6:       83 f8 0f                cmp    $0xf,%eax
  15d9:       74 33                   je     160e <phase_5+0x67>
```

- 接下来让rsi指向了一个数组，这里我们可以发现数组中存放的是int类型的数据，于是可以用x/16dw这个指令来访问里面的元素。



- 最后就是对数组的操作并进行一次循环，这个循环是指：设读入的参数为i，那我们访问a[i]，接着访问a[a[i]]，每次访问会有一个sum，这里sum需要添加上访问的值，最终访问的值为15时停止，每次循环计数器edx加1，最终需要为15。这里我先通过c++计算答案，然后手写了一遍进行倒推，得到了一样的答案。同时第二个参数的值需要为sum停止，否则引爆炸弹。最终得到答案为5，115。

```
int a[15] = { 10, 2, 14, 7, 8, 12, 15, 11, 0 , 4, 1, 13, 3, 9, 6 };
int main() {
    for (int i = 0; i < 15; i++) {
        int k = i;
        int sum = 0;
        int id = 1;
        for (int j = 0; j < 15; j++) {
            k = a[k];
            sum += k;
            if (k == 15 && j != 14) {
                id = 0;
                break;
            }
        }
        if (k == 15 && id == 1) {
            cout << i << " " << sum << endl;
        }
    }
}
```

phase_6

1. 查看读入，这里我们也是调用"read_six_numbers"函数，这里把这六个参数压入栈里保存。
2. 接下来我们初始化了一些参数，然后会进入一个循环，

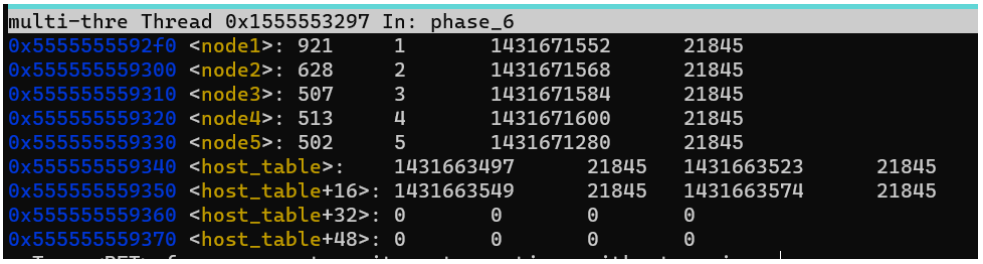
- 首先我们判断当前这个数，并且得到这个数的范围应该是1-6之间，否则触发炸弹。
- 然后我们会继续遍历我们得到的数组，将当前这个数和其他数进行比较，要求两两不同。
- 不难发现这里其实是一个二重循环，这里这个循环主要进行的操作是把a[i]和a[i + 1],简而言之就是这六个数应该分别为1,2,3,4,5,6,只是顺序有待考证。

```
  16d1:       41 8b 04 9c             mov    (%r12,%rbx,4),%eax
   16d5:      39 45 00                cmp    %eax,0x0(%rbp)
   16d8:      75 ee                   jne    16c8 <phase_6+0xa9>
   16da:      e8 83 02 00 00          call   1962 <explode_bomb>
   16df:      eb e7                   jmp    16c8 <phase_6+0xa9>
   16e1:      49 83 c6 04             add    $0x4,%r14
   16e5:      49 83 c5 01             add    $0x1,%r13
   16e9:      49 83 fd 07             cmp    $0x7,%r13
   16ed:      0f 84 62 ff ff ff       je     1655 <phase_6+0x36>
   16f3:      4c 89 f5                mov    %r14,%rbp
   16f6:      41 8b 06                mov    (%r14),%eax
   16f9:      83 e8 01                sub    $0x1,%eax
   16fc:      83 f8 05                cmp    $0x5,%eax
   16ff:      0f 87 41 ff ff ff       ja     1646 <phase_6+0x27>
   1705:      41 83 fd 05             cmp    $0x5,%r13d
   1709:      7f d6                   jg     16e1 <phase_6+0xc2>
   170b:      4c 89 eb                mov    %r13,%rbx
   170e:      eb c1                   jmp    16d1 <phase_6+0xb2>
```

3. 接下来这部分让我开始有点困惑——就是这个rdx，他是一个node，里面存储了很多数据和地址，那怎么将这个数据和我们的读入结合呢？仔细阅读下面代码，我们不难发现，如果当前在访问第i个元素，同时他的值为j，那我们会取得一个地址为rdx + (j - 1) * 8的数，把他放在栈上第i个元素对应的位置，即rsp + (i - 1) * 8上，这样便完成了这些数据的存放。

```
   1655:       be 00 00 00 00          mov    $0x0,%esi
   165a:       8b 4c b4 30             mov    0x30(%rsp,%rsi,4),%ecx
   165e:       b8 01 00 00 00          mov    $0x1,%eax
   1663:       48 8d 15 86 3c 00 00    lea    0x3c86(%rip),%rdx        # 52f0
<node1>
   166a:       83 f9 01                cmp    $0x1,%ecx
   166d:       7e 0b                   jle    167a <phase_6+0x5b>
   166f:       48 8b 52 08             mov    0x8(%rdx),%rdx
   1673:       83 c0 01                add    $0x1,%eax
   1676:       39 c8                   cmp    %ecx,%eax
   1678:       75 f5                   jne    166f <phase_6+0x50>
   167a:       48 89 14 f4             mov    %rdx,(%rsp,%rsi,8)
   167e:       48 83 c6 01             add    $0x1,%rsi
   1682:       48 83 fe 06             cmp    $0x6,%rsi
   1686:       75 d2                   jne    165a <phase_6+0x3b>
```

4. 完成数据存放之后呢？我们应该如何获得数据呢？这里我本来准备使用x/48dw的命令来查看node的内容，但我发现如下图所示，只能访问到后五个点，我于是陷入了迷茫，然后先往下看接下来的操作。



5. 接下来一串移动的操作拯救了我，我直接通过访问这些寄存器的值来获取数据,以无符号整数形式来看，最终得到六个结点内的数据分别为921,628,507,513,502,676。他们分别由1,2,3,4,5,6存放

```
  1688:       48 8b 1c 24             mov    (%rsp),%rbx
  168c:       48 8b 44 24 08          mov    0x8(%rsp),%rax ...
```

6. 然后就是最后的操作了，这里将栈内每个最先存入的数对应node里的值拿出来，要求这个值比下一个存入数的node大，即满足这个node是递减关系，我们可以得到最总这六个数的顺序应该是1 6 2 4 3 5 即为答案。

```
mov    0x8(%rbx),%rax
    171d:       8b 00                   mov    (%rax),%eax
    171f:       39 03                   cmp    %eax,(%rbx)
    1721:       7d ed                   jge    1710 <phase_6+0xf1>
```

完成感想

1. 每个关卡的难度层层递增，我在前两关主要是对gdb命令的适应，在三四五关逐渐掌握了解题的思路与一定的套路（比如r=rdi的输入格式等），第六关则是一个十分综合的问题，由非常多的小函数构成，需要对每个函数的功能仔细挖掘，耐心探索，才能将各个函数串起来，最终得到答案，这也是我在做的过程中用时最久的一关。我感觉这个实验对我linux基本命令的掌握有很大的帮助
2. 一点小建议：在第四关中的递归函数我感觉本身似乎没有明显的意义？我在想如果更换为一些其他的有意义的递归函数，比如阶乘、遍历树、BFS和DFS等等，由于其难度可能较大放在第六关考察递归似乎也可以。
3. 新的实验设计：BFS与DFS的一些典型题目，比如栈的卡特兰数，数楼梯等等。