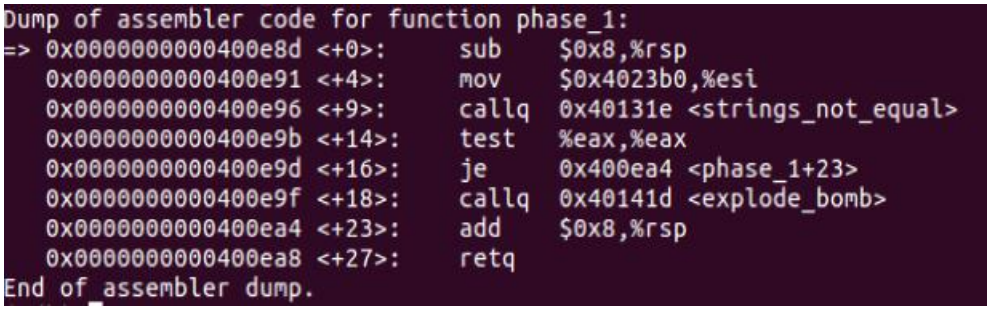
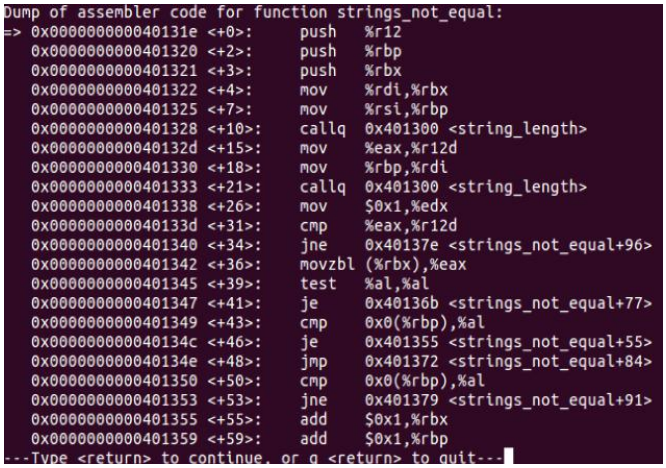
phase_1：（字符串比较）

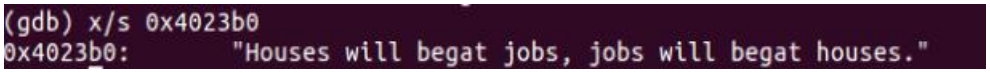
首先在 phase_1 处打上断电，然后运行 bomb，之后查看函数汇编实现



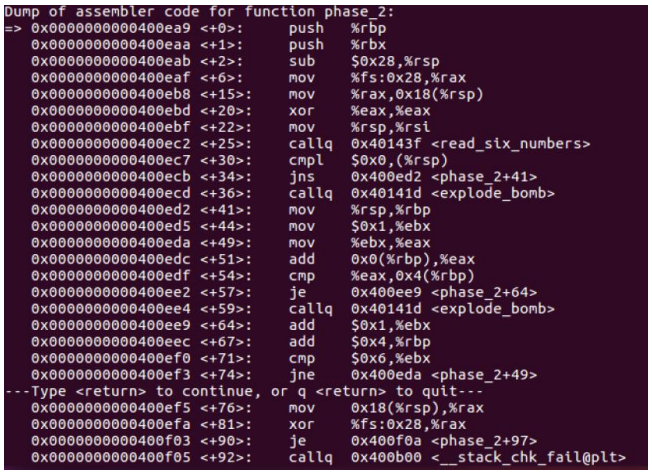
可以看到，调用了 string_not_equal 函数，猜测是判断字符串是否相等的函数，查看函数的汇编实现，%eax 应该是储存了函数的返回值，再看<+14>和<+16>的地方，可以看见如果%eax 的值不是零，炸弹就会爆炸，因此应该是要输入一个和炸弹中一样的字符串，不然炸弹就会爆炸，通过查看韩大户的汇编实现可以发现炸弹里的字符串应该是储存在$0x4023b0 的内存



中，再查看内存地址$0x4023b0 中的内容得到答案：Houses will begat jobs, jobs will begat houses.



phase_2：（循环）



依旧是先查看 phase_2 的汇编实现，可以看到调用了一个 read_six_number 的函数，推测应该要输入六个数，记为 a1-a6；可以看到 34 行先将我们输入的第一个数字 a1 与 0x0 进行比较，如果第一个输入 a1 比 0 小的话，炸弹就会直接爆炸！因此，我们输入一个大于等于 0 的 a1 后，程序跳转到 41 行。我们观察这一段代码，这应该是一段循环。

```
0x0000000000400ed2 <+41>:    mov    %rsp,%rbp
0x0000000000400ed5 <+44>:    mov    $0x1,%ebx
0x0000000000400eda <+49>:    mov    %ebx,%eax
0x0000000000400edc <+51>:    add    0x0(%rbp),%eax
0x0000000000400edf <+54>:    cmp    %eax,0x4(%rbp)
0x0000000000400ee2 <+57>:    je     0x400ee9 <phase_2+64>
0x0000000000400ee4 <+59>:    callq  0x40141d <explode_bomb>
0x0000000000400ee9 <+64>:    add    $0x1,%ebx
0x0000000000400eec <+67>:    add    $0x4,%rbp
0x0000000000400ef0 <+71>:    cmp    $0x6,%ebx
0x0000000000400ef3 <+74>:    jne    0x400eda <phase_2+49>
```

可以看出%ebx 起到计数器的作用，让循环进行 6 次，并且应该有 a2=a1+1,a3=a2+2,……,a6=a5+5，因此答案应该为 6 位数，并且满足 a1>=0,

a2=a1+1,a3=a2+2,……,a6=a5+5。

phase_3：（switch）
还是老规矩，先查看 phase_3 的汇编代码，

```
=> 0x0000000000400f11 <+0>:     sub    $0x18,%rsp
   0x0000000000400f15 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400f1e <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000400f23 <+18>:    xor    %eax,%eax
   0x0000000000400f25 <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000400f2a <+25>:    mov    %rsp,%rdx
   0x0000000000400f2d <+28>:    mov    $0x4025af,%esi
   0x0000000000400f32 <+33>:    callq  0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000400f37 <+38>:    cmp    $0x1,%eax
   0x0000000000400f3a <+41>:    jg     0x400f41 <phase_3+48>
   0x0000000000400f3c <+43>:    callq  0x40141d <explode_bomb>
   0x0000000000400f41 <+48>:    cmpl   $0x7,(%rsp)
   0x0000000000400f45 <+52>:    ja     0x400f82 <phase_3+113>
   0x0000000000400f47 <+54>:    mov    (%rsp),%eax
   0x0000000000400f4a <+57>:    jmpq   *0x402420(,%rax,8)
   0x0000000000400f51 <+64>:    mov    $0x126,%eax
   0x0000000000400f56 <+69>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f58 <+71>:    mov    $0x32d,%eax
   0x0000000000400f5d <+76>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f5f <+78>:    mov    $0x273,%eax
   0x0000000000400f64 <+83>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f66 <+85>:    mov    $0xfa,%eax
   0x0000000000400f6b <+90>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f6d <+92>:    mov    $0x154,%eax
   0x0000000000400f72 <+97>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f74 <+99>:    mov    $0x6a,%eax
   0x0000000000400f79 <+104>:   jmp    0x400f93 <phase_3+130>
```

看下面这几行，%eax 应该是保存的 scanf 的返回值，从鞋机航可以看出，%eax 的值一定要比 0x1 大，否则炸弹就会直接爆炸，因此，至少需要输入两个内容，通过查看%rsp 的内容

```
0x0000000000400f32 <+33>:    callq  0x400bb0 <__isoc99_sscanf@plt>
0x0000000000400f37 <+38>:    cmp    $0x1,%eax
0x0000000000400f3a <+41>:    jg     0x400f41 <phase_3+48>
0x0000000000400f3c <+43>:    callq  0x40141d <explode_bomb>
```

通过以下几行代码可以知道，第一个输入必须小于 7，否则炸弹就会直接爆炸；

```
0x0000000000400f41 <+48>:    cmpl   $0x7,(%rsp)
0x0000000000400f45 <+52>:    ja     0x400f82 <phase_3+113>
0x0000000000400f47 <+54>:    mov    (%rsp),%eax
```

然后看接下来的部分，这里应该是一个 switch 语句，第一个输入被储存在%eax 中，然后跳转到对应的地方，将第二个输入与指定的值比较，如果相同则通过，所以这一关的答案并不

```
0x0000000000400f47 <+54>:    mov    (%rsp),%eax
0x0000000000400f4a <+57>:    jmpq   *0x402420(,%rax,8)
0x0000000000400f51 <+64>:    mov    $0x126,%eax
0x0000000000400f56 <+69>:    jmp    0x400f93 <phase_3+130>
0x0000000000400f58 <+71>:    mov    $0x32d,%eax
0x0000000000400f5d <+76>:    jmp    0x400f93 <phase_3+130>
0x0000000000400f5f <+78>:    mov    $0x273,%eax
0x0000000000400f64 <+83>:    jmp    0x400f93 <phase_3+130>
0x0000000000400f66 <+85>:    mov    $0xfa,%eax
0x0000000000400f6b <+90>:    jmp    0x400f93 <phase_3+130>
0x0000000000400f6d <+92>:    mov    $0x154,%eax
0x0000000000400f72 <+97>:    jmp    0x400f93 <phase_3+130>
0x0000000000400f74 <+99>:    mov    $0x6a,%eax
0x0000000000400f79 <+104>:   jmp    0x400f93 <phase_3+130>
Type <return> to continue, or q <return> to quit---
0x0000000000400f7b <+106>:   mov    $0x51,%eax
0x0000000000400f80 <+111>:   jmp    0x400f93 <phase_3+130>
0x0000000000400f82 <+113>:   callq  0x40141d <explode_bomb>
0x0000000000400f87 <+118>:   mov    $0x0,%eax
0x0000000000400f8c <+123>:   jmp    0x400f93 <phase_3+130>
0x0000000000400f8e <+125>:   mov    $0x1c7,%eax
0x0000000000400f93 <+130>:   cmp    0x4(%rsp),%eax
0x0000000000400f97 <+134>:   je     0x400f9e <phase_3+141>
0x0000000000400f99 <+136>:   callq  0x40141d <explode_bomb>
```

唯一，2 和 813 是一个正确答案。

phase_4：（递归函数）

```
Dump of assembler code for function phase_4:
=> 0x0000000000400ff3 <+0>:     sub    $0x18,%rsp
   0x0000000000400ff7 <+4>:     mov    %fs:0x28,%rax
   0x0000000000401000 <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000401005 <+18>:    xor    %eax,%eax
   0x0000000000401007 <+20>:    mov    %rsp,%rcx
   0x000000000040100a <+23>:    lea    0x4(%rsp),%rdx
   0x000000000040100f <+28>:    mov    $0x4025af,%esi
   0x0000000000401014 <+33>:    callq  0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000401019 <+38>:    cmp    $0x2,%eax
   0x000000000040101c <+41>:    jne    0x401029 <phase_4+54>
   0x000000000040101e <+43>:    mov    (%rsp),%eax
   0x0000000000401021 <+46>:    sub    $0x2,%eax
   0x0000000000401024 <+49>:    cmp    $0x2,%eax
   0x0000000000401027 <+52>:    jbe    0x40102e <phase_4+59>
   0x0000000000401029 <+54>:    callq  0x40141d <explode_bomb>
   0x000000000040102e <+59>:    mov    (%rsp),%esi
   0x0000000000401031 <+62>:    mov    $0x7,%edi
   0x0000000000401036 <+67>:    callq  0x400fb8 <func4>
   0x000000000040103b <+72>:    cmp    0x4(%rsp),%eax
   0x000000000040103f <+76>:    je     0x401046 <phase_4+83>
   0x0000000000401041 <+78>:    callq  0x40141d <explode_bomb>
   0x0000000000401046 <+83>:    mov    0x8(%rsp),%rax
   0x000000000040104b <+88>:    xor    %fs:0x28,%rax
   0x0000000000401054 <+97>:    je     0x40105b <phase_4+104>
   0x0000000000401056 <+99>:    callq  0x400b00 <__stack_chk_fail@plt>
   0x000000000040105b <+104>:   add    $0x18,%rsp
   0x000000000040105f <+108>:   retq
```

还是先查看汇编代码，可以发现这一关需要两个输入，并且第一个输入存在%rsp+0x4 处，第二个存在%rsp 处，

```
0x000000000040101e <+43>:    mov    (%rsp),%eax
0x0000000000401021 <+46>:    sub    $0x2,%eax
0x0000000000401024 <+49>:    cmp    $0x2,%eax
0x0000000000401027 <+52>:    jbe    0x40102e <phase_4+59>
0x0000000000401029 <+54>:    callq  0x40141d <explode_bomb>
```

通过这几行发现第二个输入必须大于 2 且小于 4，左移它只能等于 3，然后进入函数 func4，

```
Dump of assembler code for function func4:
   0x0000000000400fb8 <+0>:     test   %edi,%edi
   0x0000000000400fba <+2>:     jle    0x400fe7 <func4+47>
   0x0000000000400fbc <+4>:     mov    %esi,%eax
   0x0000000000400fbe <+6>:     cmp    $0x1,%edi
   0x0000000000400fc1 <+9>:     je     0x400ff1 <func4+57>
   0x0000000000400fc3 <+11>:    push   %r12
   0x0000000000400fc5 <+13>:    push   %rbp
   0x0000000000400fc6 <+14>:    push   %rbx
   0x0000000000400fc7 <+15>:    mov    %esi,%ebp
   0x0000000000400fc9 <+17>:    mov    %edi,%ebx
   0x0000000000400fcb <+19>:    lea    -0x1(%rdi),%edi
   0x0000000000400fce <+22>:    callq  0x400fb8 <func4>
   0x0000000000400fd3 <+27>:    lea    0x0(%rbp,%rax,1),%r12d
   0x0000000000400fd8 <+32>:    lea    -0x2(%rbx),%edi
   0x0000000000400fdb <+35>:    mov    %ebp,%esi
   0x0000000000400fdd <+37>:    callq  0x400fb8 <func4>
   0x0000000000400fe2 <+42>:    add    %r12d,%eax
   0x0000000000400fe5 <+45>:    jmp    0x400fed <func4+53>
   0x0000000000400fe7 <+47>:    mov    $0x0,%eax
   0x0000000000400fec <+52>:    retq
   0x0000000000400fed <+53>:    pop    %rbx
   0x0000000000400fee <+54>:    pop    %rbp
   0x0000000000400fef <+55>:    pop    %r12
   0x0000000000400ff1 <+57>:    repz retq
```

发现这是一个递归函数，而在 phase_4 函数中，下面这几行是将第一个输入与 func4 的返回

```
0x0000000000401036 <+67>:    callq  0x400fb8 <func4>
0x000000000040103b <+72>:    cmp    0x4(%rsp),%eax
0x000000000040103f <+76>:    je     0x401046 <phase_4+83>
0x0000000000401041 <+78>:    callq  0x40141d <explode_bomb>
```

值作比较，他们必须一致才能成功，因此，只需要在调用 func4 后查看寄存器%eax 中的值，便是答案，最后得到答案应该是两个输入 99,3

phase_5：

```
Dump of assembler code for function phase_5:
=> 0x0000000000401060 <+0>:     sub    $0x18,%rsp
   0x0000000000401064 <+4>:     mov    %fs:0x28,%rax
   0x000000000040106d <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000401072 <+18>:    xor    %eax,%eax
   0x0000000000401074 <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000401079 <+25>:    mov    %rsp,%rdx
   0x000000000040107c <+28>:    mov    $0x4025af,%esi
   0x0000000000401081 <+33>:    callq  0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000401086 <+38>:    cmp    $0x1,%eax
   0x0000000000401089 <+41>:    jg     0x401090 <phase_5+48>
   0x000000000040108b <+43>:    callq  0x40141d <explode_bomb>
   0x0000000000401090 <+48>:    mov    (%rsp),%eax
   0x0000000000401093 <+51>:    and    $0xf,%eax
   0x0000000000401096 <+54>:    mov    %eax,(%rsp)
   0x0000000000401099 <+57>:    cmp    $0xf,%eax
   0x000000000040109c <+60>:    je     0x4010cd <phase_5+109>
   0x000000000040109e <+62>:    mov    $0x0,%ecx
   0x00000000004010a3 <+67>:    mov    $0x0,%edx
   0x00000000004010a8 <+72>:    add    $0x1,%edx
   0x00000000004010ab <+75>:    cltq
   0x00000000004010ad <+77>:    mov    0x402460(,%rax,4),%eax
   0x00000000004010b4 <+84>:    add    %eax,%ecx
   0x00000000004010b6 <+86>:    cmp    $0xf,%eax
   0x00000000004010b9 <+89>:    jne    0x4010a8 <phase_5+72>
   0x00000000004010bb <+91>:    movl   $0xf,(%rsp)
   0x00000000004010c2 <+98>:    cmp    $0xf,%edx
   0x00000000004010c5 <+101>:   jne    0x4010cd <phase_5+109>
```

还是先查看 phase_5 的汇编代码，从<+38>和<+41>可以看出这一关需要两个输入，同样可以试出一个输入 a1 在%rsp，第二个输入 a2 在%rsp+4 处，之后将第一个输入取了最低一位，储存在%eax 中，同时第一个输入的最低位不能是 f，否则炸弹会直接爆炸。

接下来，内存 0x402460 中保存的是一个数组，

```
(gdb) x/16dw 0x402460
0x402460 <array.3597>:  10      2       14      7
0x402470 <array.3597+16>:       8       12      15      11
0x402480 <array.3597+32>:       0       4       1       13
0x402490 <array.3597+48>:       3       9       6       5
```

接下来这几行中，如果把寄存器%eax 的值记作是 i 的话，<+77>处的语句就相当于 i=arr[i]，<+84>就相当于 sum+=i 这个操作，当 i=15 时，循环就会退出，而且循环必须刚好进行 15 次，同时第二个输入应该和此时 15 和数字的和 sum 相等。

```
0x00000000004010a8 <+72>:    add    $0x1,%edx
0x00000000004010ab <+75>:    cltq
0x00000000004010ad <+77>:    mov    0x402460(,%rax,4),%eax
0x00000000004010b4 <+84>:    add    %eax,%ecx
0x00000000004010b6 <+86>:    cmp    $0xf,%eax
0x00000000004010b9 <+89>:    jne    0x4010a8 <phase_5+72>
0x00000000004010bb <+91>:    movl   $0xf,(%rsp)
0x00000000004010c2 <+98>:    cmp    $0xf,%edx
0x00000000004010c5 <+101>:   jne    0x4010cd <phase_5+109>
```

因此，我们可以退出%eax 为 5 必须是 5，也就是第一个输入是 5，第二个就是 15+6+14+2+1+10+0+8+4+9+13+11+7+3+12=115，因此答案为 5 和 115 。

phase_6：

第六关的汇编代码如下

```
Dump of assembler code for function phase_6:
=> 0x00000000004010ec <+0>:     push   %r13
   0x00000000004010ee <+2>:     push   %r12
   0x00000000004010f0 <+4>:     push   %rbp
   0x00000000004010f1 <+5>:     push   %rbx
   0x00000000004010f2 <+6>:     sub    $0x68,%rsp
   0x00000000004010f6 <+10>:    mov    %fs:0x28,%rax
   0x00000000004010ff <+19>:    mov    %rax,0x58(%rsp)
   0x0000000000401104 <+24>:    xor    %eax,%eax
   0x0000000000401106 <+26>:    mov    %rsp,%rsi
   0x0000000000401109 <+29>:    callq  0x40143f <read_six_numbers>
   0x000000000040110e <+34>:    mov    %rsp,%r12
   0x0000000000401111 <+37>:    mov    $0x0,%r13d
   0x0000000000401117 <+43>:    mov    %r12,%rbp
   0x000000000040111a <+46>:    mov    (%r12),%eax
   0x000000000040111e <+50>:    sub    $0x1,%eax
   0x0000000000401121 <+53>:    cmp    $0x5,%eax
   0x0000000000401124 <+56>:    jbe    0x40112b <phase_6+63>
   0x0000000000401126 <+58>:    callq  0x40141d <explode_bomb>
   0x000000000040112b <+63>:    add    $0x1,%r13d
   0x000000000040112f <+67>:    cmp    $0x6,%r13d
   0x0000000000401133 <+71>:    je     0x401172 <phase_6+134>
   0x0000000000401135 <+73>:    mov    %r13d,%ebx
   0x0000000000401138 <+76>:    movslq %ebx,%rax
   0x000000000040113b <+79>:    mov    (%rsp,%rax,4),%eax
   0x000000000040113e <+82>:    cmp    %eax,0x0(%rbp)
   0x0000000000401141 <+85>:    jne    0x401148 <phase_6+92>
   0x0000000000401143 <+87>:    callq  0x40141d <explode_bomb>
```

```
0x0000000000401148 <+92>:    add    $0x1,%ebx
0x000000000040114b <+95>:    cmp    $0x5,%ebx
0x000000000040114e <+98>:    jle    0x401138 <phase_6+76>
0x0000000000401150 <+100>:   add    $0x4,%r12
0x0000000000401154 <+104>:   jmp    0x401117 <phase_6+43>
0x0000000000401156 <+106>:   mov    0x8(%rdx),%rdx
0x000000000040115a <+110>:   add    $0x1,%eax
0x000000000040115d <+113>:   cmp    %ecx,%eax
0x000000000040115f <+115>:   jne    0x401156 <phase_6+106>
0x0000000000401161 <+117>:   mov    %rdx,0x20(%rsp,%rsi,2)
0x0000000000401166 <+122>:   add    $0x4,%rsi
0x000000000040116a <+126>:   cmp    $0x18,%rsi
0x000000000040116e <+130>:   jne    0x401177 <phase_6+139>
0x0000000000401170 <+132>:   jmp    0x40118b <phase_6+159>
0x0000000000401172 <+134>:   mov    $0x0,%esi
0x0000000000401177 <+139>:   mov    (%rsp,%rsi,1),%ecx
0x000000000040117a <+142>:   mov    $0x1,%eax
0x000000000040117f <+147>:   mov    $0x6032f0,%edx
0x0000000000401184 <+152>:   cmp    $0x1,%ecx
0x0000000000401187 <+155>:   jg     0x401156 <phase_6+106>
0x0000000000401189 <+157>:   jmp    0x401161 <phase_6+117>
0x000000000040118b <+159>:   mov    0x20(%rsp),%rbx
0x0000000000401190 <+164>:   lea    0x20(%rsp),%rax
0x0000000000401195 <+169>:   lea    0x48(%rsp),%rsi
0x000000000040119a <+174>:   mov    %rbx,%rcx
0x000000000040119d <+177>:   mov    0x8(%rax),%rdx
0x00000000004011a1 <+181>:   mov    %rdx,0x8(%rcx)
0x00000000004011a5 <+185>:   add    $0x8,%rax
0x00000000004011a9 <+189>:   mov    %rdx,%rcx
0x00000000004011ac <+192>:   cmp    %rsi,%rax
0x00000000004011af <+195>:   jne    0x40119d <phase_6+177>
0x00000000004011b1 <+197>:   movq   $0x0,0x8(%rdx)
```

```
0x00000000004011b9 <+205>:   mov    $0x5,%ebp
0x00000000004011be <+210>:   mov    0x8(%rbx),%rax
0x00000000004011c2 <+214>:   mov    (%rax),%eax
0x00000000004011c4 <+216>:   cmp    %eax,(%rbx)
0x00000000004011c6 <+218>:   jge    0x4011cd <phase_6+225>
0x00000000004011c8 <+220>:   callq  0x40141d <explode_bomb>
0x00000000004011cd <+225>:   mov    0x8(%rbx),%rbx
0x00000000004011d1 <+229>:   sub    $0x1,%ebp
0x00000000004011d4 <+232>:   jne    0x4011be <phase_6+210>
0x00000000004011d6 <+234>:   mov    0x58(%rsp),%rax
0x00000000004011db <+239>:   xor    %fs:0x28,%rax
0x00000000004011e4 <+248>:   je     0x4011eb <phase_6+255>
0x00000000004011e6 <+250>:   callq  0x400b00 <__stack_chk_fail@plt>
0x00000000004011eb <+255>:   add    $0x68,%rsp
0x00000000004011ef <+259>:   pop    %rbx
0x00000000004011f0 <+260>:   pop    %rbp
0x00000000004011f1 <+261>:   pop    %r12
0x00000000004011f3 <+263>:   pop    %r13
```

可以看到<+29>调用了 read_six_number 函数，从其后的语句可以看出要求必须输入的是六个数，否则炸弹会爆炸，后面直到<+104>这一部分是看输入的 6 个数组是否都小于 6 且不重复，否则炸弹会直接爆炸，故数组应该是 1~6 的一个排列。
接下来的部分这其实是一个链表数据结构，根据我们输入的数组，按照数组元素的值将对应结构体数组中的元素的首地址存储到内存的某个位置，并且要求单链表要有按照节点递减的形式。
查看内存 0x6032f0 的内容，按照节点大小排序可以得到答案为 2 5 1 6 3 4.

```
(gdb) x/32dw 0x6032f0
0x6032f0 <node1>:       817     1       6304512 0
0x603300 <node2>:       888     2       6304528 0
0x603310 <node3>:       548     3       6304544 0
0x603320 <node4>:       347     4       6304560 0
0x603330 <node5>:       826     5       6304576 0
0x603340 <node6>:       729     6       0       0
0x603350:       0       0       0       0
0x603360 <host table>:  4204041 0       4204067 0
```

然后就成功拆除了炸弹。

```
Congratulations! You've defused the bomb!
[Inferior 1 (process 2316) exited normally]
```