Phase_1:

First disas in phase_1, I see it moves $0x4023a8 to $esi and then it call the function String_not_equal which means a compare function. So I need to know what's in that location.

```
(gdb) x/s 0x4023a8
0x4023a8 <__dso_handle+384>:     "He is evil and fits easily into most overhead
```

So the string ""He is evil and fits easily into most overhead storage bins." Is the answer.

Phase_2:

In phase_2 I see the function <read_six_numbers> which means in this phase there will be six numbers as inputs.

```
0x0000000000401068 <+14>:    cmpl   $0x0,(%rsp)
0x000000000040106c <+18>:    jne    0x401075 <phase_2+27>
0x000000000040106e <+20>:    cmpl   $0x1,0x4(%rsp)
0x0000000000401073 <+25>:    je     0x40107a <phase_2+32>
0x0000000000401075 <+27>:    callq  0x4013ef <explode_bomb>
```

In red box, system compares two numbers to %rsp which is stack pointer saved what I inputted. The first number I inputted is saved at the bottom of the stack so it is at %rsp, and the second number is at (%rsp) + 4 and so on. From the picture, it tells that the first number is 0 and the second is 1.

```
0x000000000040107a <+32>:    mov    %rsp,%rbp
0x000000000040107d <+35>:    lea    0x8(%rsp),%rbx
0x0000000000401082 <+40>:    add    $0x18,%rbp
0x0000000000401086 <+44>:    mov    -0x4(%rbx),%eax
0x0000000000401089 <+47>:    add    -0x8(%rbx),%eax
0x000000000040108c <+50>:    cmp    %eax,(%rbx)
0x000000000040108e <+52>:    je     0x401095 <phase_2+59>
0x0000000000401090 <+54>:    callq  0x4013ef <explode_bomb>
0x0000000000401095 <+59>:    add    $0x4,%rbx
0x0000000000401099 <+63>:    cmp    %rbp,%rbx
0x000000000040109c <+66>:    jne    0x401086 <phase_2+44>
```

In this picture, there is a loop that compare the third number to the sum of the first and second numbers. So the following numbers are the sum of the two numbers before them. The answer is 0 1 1 2 3 5.

Phase_3:

```
0x000000000040118b <+14>:    mov    $0x4024aa,%esi
```

I see it moves 0x4024aa into %esi and then calls a function. I print that location out, the output is "%d %d", which means I need 2 decimal inputs.

```
0x00000000004011a4 <+39>:    cmpl   $0x7,0xc(%rsp)
0x00000000004011a9 <+44>:    ja     0x4011ee <phase_3+113>
```

Because phase_3+113 is explode_bomb, so 0xc(%rsp) has to be less than 0x7

```
0x00000000004011ab <+46>:    mov    0xc(%rsp),%eax
0x00000000004011af <+50>:    jmpq   *0x4023f0(,%rax,8)
```

There is a jump and this jump is decided by 0xc(%rsp) which is my first input. I choose 0 here and it will jump to 0x4011bd. %eax is 0x10d here, which is 269 in decimal. Then jump to phase_3+123.

```
0x00000000004011f8 <+123>:   cmp    0x8(%rsp),%eax
0x00000000004011fc <+127>:   je     0x401203 <phase_3+134>
0x00000000004011fe <+129>:   callq  0x4013ef <explode_bomb>
0x0000000000401203 <+134>:   add    $0x18,%rsp
0x0000000000401207 <+138>:   retq
```

At this line, it compare the %eax with 0x8(%rsp) which is our second input. So the second input should equals to %eax which is 269.
Two inputs are 0 and 269. This is one solution.

Phase_4:

```
0x000000000040112f <+14>:    mov    $0x4024aa,%esi
```

This is very similar to Phase_3, first I print the location out: "%d %d". The inputs are two decimals.

```
0x0000000000401143 <+34>:    mov    0xc(%rsp),%eax
0x0000000000401147 <+38>:    test   %eax,%eax
0x0000000000401149 <+40>:    js     0x401150 <phase_4+47>
```

It first move the first input to %eax and test it. Because <+47> is explode_bomb, so %eax should greater than zero.

```
0x000000000040114b <+42>:    cmp    $0xe,%eax
0x000000000040114e <+45>:    jle    0x401155 <phase_4+52>
0x0000000000401150 <+47>:    callq  0x4013ef <explode_bomb>
```

From this picture, I notice that %eax should also less than 14.

```
0x000000000040116c <+75>:    cmpl   $0x0,0x8(%rsp)
0x0000000000401171 <+80>:    je     0x401178 <phase_4+87>
0x0000000000401173 <+82>:    callq  0x4013ef <explode_bomb>
0x0000000000401178 <+87>:    add    $0x18,%rsp
```

From this picture, I notice that my second input should equals to 0.
So I try 1 and 0 as my two inputs, the phase_4 defused.

Phase_5

```
0x00000000004010ad <+8>:     callq  0x401210 <string_length>
0x00000000004010b2 <+13>:    cmp    $0x6,%eax
```

At the beginning, the program shows me that the input should be a String and the length is 6.

```
0x00000000004010c3 <+30>:    mov    $0x402430,%edx
```

The program moves 0x402430 into %edx, I print it out here: "maduiersnfotvbylInvalid phase%s\n" but I find that %edx is not used in the future, so it doesn't matter.

```
0x00000000004010c8 <+35>:    movsbq (%rbx),%rcx
0x00000000004010cc <+39>:    and    $0xf,%ecx
0x00000000004010cf <+42>:    movzbl (%rdx,%rcx,1),%ecx
0x00000000004010d3 <+46>:    mov    %cl,(%rax)
0x00000000004010d5 <+48>:    add    $0x1,%rbx
0x00000000004010d9 <+52>:    add    $0x1,%rax
0x00000000004010dd <+56>:    cmp    %rsi,%rbx
0x00000000004010e0 <+59>:    jne    0x4010c8 <phase_5+35>
```

This is a loop that make some changes on the value of a letter so that it will become to a new letter.

```
0x00000000004010ea <+69>:    mov    $0x4023e4,%esi
```

Here is the target string, I print it out here: "flyers"

```
0x00000000004010ef <+74>:    callq  0x40122c <strings_not_equal>
0x00000000004010f4 <+79>:    test   %eax,%eax
0x00000000004010f6 <+81>:    je     0x4010fd <phase_5+88>
```

In the function Strings_not_equal, it is a loop to compare each letter in the input be equal to the letters that saved in %esi. So my mission is to find the letters that after doing the loop will become to "flyers". The letters are "yonefg"

Phase_6:

In fact, this is a really long function and I have little confuse on it. So I find some instructions about bomb lab online and try to understand what this function means.

```
0x0000000000400f4d <+18>:    callq  0x401425 <read_six_numbers>
```

This picture shows that I need 6 number inputs.

```
0x0000000000400fc3 <+136>:   mov    $0x603670,%esi
```

At this location, I find out that it has six nodes, that's are:

```
(gdb) x/4w 0x603670
0x603670 <node1>:       656     1       6305376 0
(gdb) x/4w 0x603660
0x603660 <node2>:       537     2       6305360 0
(gdb) x/4w 0x603650
0x603650 <node3>:       205     3       6305344 0
(gdb) x/4w 0x603640
0x603640 <node4>:       889     4       6305328 0
(gdb) x/4w 0x603630
0x603630 <node5>:       811     5       6305312 0
(gdb) x/4w 0x603620
0x603620 <node6>:       57      6       0       0
```

My mission is to rearrange this six nodes. The answer is 3 2 6 5 4 1 (7-4, 7-5, 7-1, 7-2, 7-3, 7-6)

```
0x0000000000400f9c <+97>:    mov    $0x7,%edx
0x0000000000400fa1 <+102>:   mov    %edx,%eax
0x0000000000400fa3 <+104>:   sub    (%r12),%eax
0x0000000000400fa7 <+108>:   mov    %eax,(%r12)
0x0000000000400fab <+112>:   add    $0x4,%r12
0x0000000000400faf <+116>:   cmp    %rcx,%r12
0x0000000000400fb2 <+119>:   jne    0x400fa1 <phase_6+102>
```

Done;