**Explanation of how to defuse the bomb** **Cheung Chi Wang (1155158048)**

**Phase 1:**

In the assembly code of phase one, there is an address that stored to %rsi in this line: lea 0x1b96(%rip),%rsi. And as the phase will call a function named as <strings_not_equal>, therefore we need to find out the corresponding strings as the key of defusing this phase. I used gdb to set a breakpoint at phase_1 and run till that line which call the function by nexti. The corresponding address of rsi is `0x55555555714c` and I used x/s to find the string stored in that address. The required string is "`Crikey! I have lost my mojo!`"

**Phase 2:**

For phase 2, I first find that it calls the function <read_six_number> so I assume the required key is consist of six numbers. There is a loop from line 1609 to 161d and the counter of the loop is ebx as it adds by 1 in every iteration and the iteration is 6. In line 1615 to 1617:

mov %ebx,%eax; add 0x0(%rbp),%eax, it adds the first number of six to the counter. Then compare it to the next number (%eax,0x4(%rbp)) and if the condition which the above are equal, the loop continue, else explode the bomb. In every loop, %rbp will be updated which will refer to the next number. In conclusion, second number == first number +1, third number == second number +2, fourth number == third number +3,etc. Therefore, my key for this phase is 1,2,4,7,11,16.

**Phase 3:**

For phase 3, I find that it calls a function to scan our input. I find the format of input by looking into the memory address of %rsi in line 165d, the corresponding address is `0x555555557317` and I printed the required format which is "%d %d". Therefore I need to input two integers.

At line 166e to 1672: cmpl $0x7,(%rsp); ja 1712; it compares the first integer to 7 and if it is above 7, the bomb explodes. Therefore the first digit must be smaller or equal to 7. From 167b to 1686, it assigns a memory address to rax which will be used for jump table in 1689. Different first integer will require different second integer and I have chosen 3 as my first integer. Therefore, in the jump table, it jumps into line 16ef which assign 0 to eax then will be jump again to 16a2. From 16a2 to 16b6, eax undergo some maths calculation and eventually become -0x266. At line 16c1: cmp %eax,0x4(%rsp), the second integer will need to be equal to eax, ie. -0x266/ -614, to avoid exploding the bomb. Therefore, my key for this phase is 3, -614.

**Phase 4:**

For phase 4, it also calls a function to scan our input so I take a look to the format of the input again by looking into the memory address of rsi. The required input format is "%d %d" again.

At 1795: cmpl $0xe,(%rsp), it will jump over the explode_bomb if the first integer is below or equal to 14. Then it assign edx=14, esi=0, edi=first integer, then call func4. At 17b7: cmpl $0x1,0x4(%rsp), therefore the second integer must be 1 to avoid bomb explodes.

In func 4, it is actually a recursion that at first recursion, it assigns eax=edx=14, then minus eax by esi=0, then assign ecx=eax. Therefore here can written as: eax=edx-esi; ecx=eax; From 1734 to 1736, ecx=eax+ecx then divided by two, ie. equal to ecx itself. Then compare edi (first integer) and ecx. If ecx is greater than edi, edx=ecx-1 and call func4 again. If ecx is less than edi, edx=ecx+1 and call func4 again. Therefore, the function will end until ecx==edi.

As a result, only number 8,9,11 can be used to end the recursion. I used 11,1 as my key for this phase.

**Phase 5:**

For phase5, I find that it will scan our input once again so I check the input format. The required format is "%d %d" again. At 1817, it checks that the first integer cannot be 0xf,which is 15. At 181c to 1826, it assigns ecx=0, edx=0 and there are some memory address assigned to rsi. I took a look to that address and I found an array of number from 0 to 15. The order is: `10,2,14,7,8,12,15,11,0,4,1,13,3,9,6,5.` Then there is a loop from 182d to 183a. The counter is edx. At line 1832: mov (%rsi,%rax,4),%eax, it update eax with "eax"th+1 element in the array above. For example, if the current eax= 3, it will update eax by the fourth element in the array, which is "7".Then it will add the number to ecx. The condition of this loop to end is that eax is equal to 15. However, before the loop end, it need to iterate 15 times as at line 1843: cmp $0xf,%edx, the counter edx need to equal to 15 to avoid bomb explode. Moreover, at line 1848, it indicates that ecx, which is the sum of eax in the loop, will be the second integer to be input. Therefore, by marking the value of eax at the end of loop is 15, we can find the other 14 eax value by finding the position of the next eax in the array. The position of "15" in the array is "6", the last eax is then equal to "6". The position of "6" in the array is "14", therefore the 13rd eax is "14",etc. The 15 eax required by ordering is "15,6,14,2,1,10,0,8,4,9,13,11,7,3,12". The sum of them is 115 and the first integer should be the initial eax which is the position of "12", ie. "5". Therefore my key for phase 5 will be 5, 115.

**Phase 6:**

For phase 6, the required input will be 6 numbers as there is a function of <read_six_number>. There is some loops from 18b1 to 18ea, rbx is the counter and the iteration is 6. It checks the six number input that each of them cannot be greater than six (this condition can be found at 18d9 to 18dc). Moreover, it also update the value of eax at line 18ba by mov (%r12,%rbx,4),%eax. Therefore, eax is assigned to xth number of the six numbers in xth iteration of the loop. It checks that the eax cannot be equal to the first number in rbp. As in 18ce to 18d2, rbp is assigned to point to the next number of the six number, ie the first number becomes second number and third becomes second, etc. This can check that all six number cannot be repeated. In conclusion, the input need to be 6 numbers that cannot be greater than six and cannot be repeated, probably any permutation of 1,2,3,4,5,6.

The other part of the assembly code is determining the order of the numbers. At 18f9, a memory address to assigned to rdx. I had a look to that address by using x/12gx, I found 5 nodes and their address. I wonder where is node 6 then I figured out by trying to look into the address. I found:

```
0x555555559210 <node1>: 0x0000000100000175      0x0000555555559220
0x555555559220 <node2>: 0x00000002000000a5      0x0000555555559230
0x555555559230 <node3>: 0x000000030000022c      0x0000555555559240
0x555555559240 <node4>: 0x0000000400000058      0x0000555555559250
0x555555559250 <node5>: 0x00000005000002ba      0x0000555555559110
0x555555559110 <node6>: 0x0000000600000274      0x0000000000000000
```

Then I find out that it assigns these node to the address (%rsp+0x20), (%rsp+0x28), (%rsp+0x30), (%rsp+0x38), (%rsp+0x40), (%rsp+0x48) respectively. Then I find out node 1 is corresponding node of number 1, node 2 is that of number 2,etc. From 1960 to 196f, it compares the next node to the current node and next node need to be greater than the current node to avoid explosion. It uses the value from the nodes (example: node1:175, node2:a5, node 3: 22c,etc.) for comparing. Therefore, the first node is node 4, then 2, 1, 3, 6, 5. And the order of the six number is then 4,2,1,3,6,5 and my key for phase_6 is also 4,2,1,3,6,5.

**Bomb defused.**