天津大学

计算机系统基础上机实验报告

实验题目 3：拆弹专家 bomb

学院名称 智能与计算学部

专　　业 工科试验班

学生姓名 张明君

学　　号 6319000359

年　　级 2019 级

班　　级 留学生班

时　　间 2020 年 6 月 20 日

实验 3：拆弹专家

Bomb

1. 实验目的

进一步掌握程序的机器级表示一章的知识。理解程序控制、过程调用的汇编级实现，熟练掌握 汇编语言程序的阅读。

2. 实验内容

程序 bomb 是一个电子炸弹，当该程序运行时，需要按照一定的顺序输入口令，才能阻 止炸弹的引爆。当输入错误的密码时，炸弹将会引爆。此时控制台将会产生如下输出，并结束 程序

```
BOOM!!!
The bomb has blown up.
```

在炸弹程序中，你需要输入多组口令，且每一组口令都正确才能够防止引爆。

目前已知的内容只有炸弹程序的二进制可执行文件 bomb（目标平台为：x86-64）和 bomb 的 main 函数框架代码，见main.c。其他的细节均不会以 c 语言的方式呈现。 你的任务是：利用现有的资源以及相关的工具，猜出炸弹的全部口令，并输入至炸弹程序中，以完成最终的拆弹工作。

3. 实验要求

1）在 Unbuntu18.04LTS 操作系统下，按照实验指导说明书，使用 gdb 和 objdump 等工具，以反向工程方式完成 Bomb 拆弹。

2）需提交：拆弹口令文本文件、电子版实验报告全文。

4. 实验结果

In this part we use objdump to get the assembly code we use

```
objdump -d bomb > bomb.asm
```

and get the following file (not the full code)

```
0000000000400ee0 <phase_1>:
  400ee0:       48 83 ec 08             sub    $0x8,%rsp
  400ee4:       be 00 24 40 00          mov    $0x402400,%esi
  400ee9:       e8 4a 04 00 00          callq  401338 <strings_not_equal>
  400eee:       85 c0                   test   %eax,%eax
  400ef0:       74 05                   je     400ef7 <phase_1+0x17>
  400ef2:       e8 43 05 00 00          callq  40143a <explode_bomb>
  400ef7:       48 83 c4 08             add    $0x8,%rsp
  400efb:       c3                      retq
```

Phase 1

We enter gdb, set a breakpoint at the phase 1. Then we take a look at the assembly code above, we see one register eax and an address 0x402400. Enter a random string and then we stop at the phase 1 position, then we try printing out the information around 0x402400. We get the following part

```
(gdb) p/x $eax
$1 = 0x603780
(gdb) x /25c 0x603780
0x603780 <input_strings>:       116 't' 101 'e' 115 's' 116 't' 0 '\000'
        0 '\000' 0 '\000' 0 '\000'
0x603788 <input_strings+8>:     0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
        0 '\000' 0 '\000' 0 '\000'
0x603790 <input_strings+16>:    0 '\000' 0 '\000' 0 '\000' 0 '\000' 0 '\000'
        0 '\000' 0 '\000' 0 '\000'
0x603798 <input_strings+24>:    0 '\000'
(gdb) x /25c 0x402400
0x402400:       66 'B'  111 'o' 114 'r' 100 'd' 101 'e' 114 'r' 32 ' '
        114 'r'
0x402408:       101 'e' 108 'l' 97 'a'  116 't' 105 'i' 111 'o' 110 'n'
        115 's'
0x402410:       32 ' '  119 'w' 105 'i' 116 't' 104 'h' 32 ' '  67 'C'
        97 'a'
0x402418:       110 'n'
```

We see a critical keyword Border, right? Then we use strings command to find out the answer

```
$ strings bomb | grep Border
Border relations with Canada have never been better.
```

The first bomb is successfully defused.

Phase 2

Firstly, let's have a look at the code

```
0000000000400efc <phase_2>:
  400efc:       55                      push   %rbp
  400efd:       53                      push   %rbx
  400efe:       48 83 ec 28             sub    $0x28,%rsp
  400f02:       48 89 e6                mov    %rsp,%rsi
  400f05:       e8 52 05 00 00          callq  40145c <read_six_numbers>
  400f0a:       83 3c 24 01             cmpl   $0x1,(%rsp)
  400f0e:       74 20                   je     400f30 <phase_2+0x34>
  400f10:       e8 25 05 00 00          callq  40143a <explode_bomb>
  400f15:       eb 19                   jmp    400f30 <phase_2+0x34>
  400f17:       8b 43 fc                mov    -0x4(%rbx),%eax
  400f1a:       01 c0                   add    %eax,%eax
  400f1c:       39 03                   cmp    %eax,(%rbx)
  400f1e:       74 05                   je     400f25 <phase_2+0x29>
  400f20:       e8 15 05 00 00          callq  40143a <explode_bomb>
  400f25:       48 83 c3 04             add    $0x4,%rbx
  400f29:       48 39 eb                cmp    %rbp,%rbx
  400f2c:       75 e9                   jne    400f17 <phase_2+0x1b>
  400f2e:       eb 0c                   jmp    400f3c <phase_2+0x40>
  400f30:       48 8d 5c 24 04          lea    0x4(%rsp),%rbx
  400f35:       48 8d 6c 24 18          lea    0x18(%rsp),%rbp
  400f3a:       eb db                   jmp    400f17 <phase_2+0x1b>
  400f3c:       48 83 c4 28             add    $0x28,%rsp
  400f40:       5b                      pop    %rbx
  400f41:       5d                      pop    %rbp
  400f42:       c3                      retq
```

Having a look at the code structure, you should notice that there exists a loop structure. What' more, there's a function call to read_six_numbers(), we can inspect it

```
000000000040145c <read_six_numbers>:
  40145c:       48 83 ec 18             sub    $0x18,%rsp
  401460:       48 89 f2                mov    %rsi,%rdx
  401463:       48 8d 4e 04             lea    0x4(%rsi),%rcx
  401467:       48 8d 46 14             lea    0x14(%rsi),%rax
  40146b:       48 89 44 24 08          mov    %rax,0x8(%rsp)
  401470:       48 8d 46 10             lea    0x10(%rsi),%rax
  401474:       48 89 04 24             mov    %rax,(%rsp)
  401478:       4c 8d 4e 0c             lea    0xc(%rsi),%r9
  40147c:       4c 8d 46 08             lea    0x8(%rsi),%r8
  401480:       be c3 25 40 00          mov    $0x4025c3,%esi
  401485:       b8 00 00 00 00          mov    $0x0,%eax
  40148a:       e8 61 f7 ff ff          callq  400bf0
<__isoc99_sscanf@plt>
  40148f:       83 f8 05                cmp    $0x5,%eax
  401492:       7f 05                   jg     401499
<read_six_numbers+0x3d>
  401494:       e8 a1 ff ff ff          callq  40143a <explode_bomb>
  401499:       48 83 c4 18             add    $0x18,%rsp
  40149d:       c3                      retq
```

Up till now, you should be able to find out that in this part, we are required to enter six numbers. Considering this line of code

```
400f0a: 83 3c 24 01             cmpl   $0x1,(%rsp)
```

It's obvious that the first number should be 1. We can find the latter numbers from the loop structure.

```
400f17: 8b 43 fc                mov    -0x4(%rbx),%eax
  400f1a:       01 c0                   add    %eax,%eax
  400f1c:       39 03                   cmp    %eax,(%rbx)
  400f1e:       74 05                   je     400f25 <phase_2+0x29>
  400f20:       e8 15 05 00 00          callq  40143a <explode_bomb>
  400f25:       48 83 c3 04             add    $0x4,%rbx
  400f29:       48 39 eb                cmp    %rbp,%rbx
  400f2c:       75 e9                   jne    400f17 <phase_2+0x1b>
  400f2e:       eb 0c                   jmp    400f3c <phase_2+0x40>
  400f30:       48 8d 5c 24 04          lea    0x4(%rsp),%rbx
  400f35:       48 8d 6c 24 18          lea    0x18(%rsp),%rbp
  400f3a:       eb db                   jmp    400f17 <phase_2+0x1b>
```

We multiply the number by 2 each step, so we guess the sequence to be 1, 2, 4, 8, 16, 32, which is the answer.

Phase 3

The third bomb is about the switch expression. Firstly, let's have a look at the asm code.

```
0000000000400f43 <phase_3>:
  400f43:       48 83 ec 18             sub    $0x18,%rsp
  400f47:       48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
  400f4c:       48 8d 54 24 08          lea    0x8(%rsp),%rdx
  400f51:       be cf 25 40 00          mov    $0x4025cf,%esi
  400f56:       b8 00 00 00 00          mov    $0x0,%eax
  400f5b:       e8 90 fc ff ff          callq  400bf0
<__isoc99_sscanf@plt>
  400f60:       83 f8 01                cmp    $0x1,%eax
  400f63:       7f 05                   jg     400f6a <phase_3+0x27>
  400f65:       e8 d0 04 00 00          callq  40143a <explode_bomb>
  400f6a:       83 7c 24 08 07          cmpl   $0x7,0x8(%rsp)
  400f6f:       77 3c                   ja     400fad <phase_3+0x6a>
  400f71:       8b 44 24 08             mov    0x8(%rsp),%eax
  400f75:       ff 24 c5 70 24 40 00    jmpq   *0x402470(,%rax,8)
  400f7c:       b8 cf 00 00 00          mov    $0xcf,%eax
  400f81:       eb 3b                   jmp    400fbe <phase_3+0x7b>
  400f83:       b8 c3 02 00 00          mov    $0x2c3,%eax
  400f88:       eb 34                   jmp    400fbe <phase_3+0x7b>
  400f8a:       b8 00 01 00 00          mov    $0x100,%eax
  400f8f:       eb 2d                   jmp    400fbe <phase_3+0x7b>
  400f91:       b8 85 01 00 00          mov    $0x185,%eax
  400f96:       eb 26                   jmp    400fbe <phase_3+0x7b>
  400f98:       b8 ce 00 00 00          mov    $0xce,%eax
  400f9d:       eb 1f                   jmp    400fbe <phase_3+0x7b>
  400f9f:       b8 aa 02 00 00          mov    $0x2aa,%eax
  400fa4:       eb 18                   jmp    400fbe <phase_3+0x7b>
  400fa6:       b8 47 01 00 00          mov    $0x147,%eax
  400fab:       eb 11                   jmp    400fbe <phase_3+0x7b>
  400fad:       e8 88 04 00 00          callq  40143a <explode_bomb>
  400fb2:       b8 00 00 00 00          mov    $0x0,%eax
  400fb7:       eb 05                   jmp    400fbe <phase_3+0x7b>
  400fb9:       b8 37 01 00 00          mov    $0x137,%eax
  400fbe:       3b 44 24 0c             cmp    0xc(%rsp),%eax
  400fc2:       74 05                   je     400fc9 <phase_3+0x86>
  400fc4:       e8 71 04 00 00          callq  40143a <explode_bomb>
  400fc9:       48 83 c4 18             add    $0x18,%rsp
```

```
  400fcd:       c3                      retq
```

From the first few lines, we guess that there are two arguments to enter. Using gdb we can convince our guess.

```
(gdb) x /s 0x4025cf
0x4025cf:       "%d %d"
```

Then we can get the range of the first argument from the line

```
400f6a: 83 7c 24 08 07          cmpl   $0x7,0x8(%rsp)
```

The first argument must be less than 7, right? Then we encounter with an optimized switch expression.

```
  400f75:       ff 24 c5 70 24 40 00    jmpq   *0x402470(,%rax,8)
  400f7c:       b8 cf 00 00 00          mov    $0xcf,%eax
  400f81:       eb 3b                   jmp    400fbe <phase_3+0x7b>
  400f83:       b8 c3 02 00 00          mov    $0x2c3,%eax
  400f88:       eb 34                   jmp    400fbe <phase_3+0x7b>
  400f8a:       b8 00 01 00 00          mov    $0x100,%eax
  400f8f:       eb 2d                   jmp    400fbe <phase_3+0x7b>
  400f91:       b8 85 01 00 00          mov    $0x185,%eax
  400f96:       eb 26                   jmp    400fbe <phase_3+0x7b>
  400f98:       b8 ce 00 00 00          mov    $0xce,%eax
  400f9d:       eb 1f                   jmp    400fbe <phase_3+0x7b>
  400f9f:       b8 aa 02 00 00          mov    $0x2aa,%eax
  400fa4:       eb 18                   jmp    400fbe <phase_3+0x7b>
  400fa6:       b8 47 01 00 00          mov    $0x147,%eax
  400fab:       eb 11                   jmp    400fbe <phase_3+0x7b>
  400fad:       e8 88 04 00 00          callq  40143a <explode_bomb>
  400fb2:       b8 00 00 00 00          mov    $0x0,%eax
  400fb7:       eb 05                   jmp    400fbe <phase_3+0x7b>
  400fb9:       b8 37 01 00 00          mov    $0x137,%eax
```

We can inspect its structure directly using gdb

```
(gdb) x /10a 0x402470
0x402470:       0x400f7c <phase_3+57>    0x400fb9 <phase_3+118>
0x402480:       0x400f83 <phase_3+64>    0x400f8a <phase_3+71>
0x402490:       0x400f91 <phase_3+78>    0x400f98 <phase_3+85>
0x4024a0:       0x400f9f <phase_3+92>    0x400fa6 <phase_3+99>
0x4024b0 <array.3449>:   0x737265697564616d       0x6c796276746f666e
```

We can see that the last line shouldn't be contained in this switch structure, while the first four should be. Actually in this part, the answer isn't unique. You just choose a number arbitarily from 0 to 6 and go through the switch expression, and you get your second argument. I choose the first argument as 1 and then the second one should be 311.

Phase 4

In this part, we are given two functions phase_4() and func4(). The key part is the latter one. Let's inspect the code at first.

```
0000000000400fce <func4>:
  400fce:       48 83 ec 08             sub    $0x8,%rsp
  400fd2:       89 d0                   mov    %edx,%eax
  400fd4:       29 f0                   sub    %esi,%eax
  400fd6:       89 c1                   mov    %eax,%ecx
  400fd8:       c1 e9 1f                shr    $0x1f,%ecx
  400fdb:       01 c8                   add    %ecx,%eax
  400fdd:       d1 f8                   sar    %eax
  400fdf:       8d 0c 30                lea    (%rax,%rsi,1),%ecx
  400fe2:       39 f9                   cmp    %edi,%ecx
  400fe4:       7e 0c                   jle    400ff2 <func4+0x24>
  400fe6:       8d 51 ff                lea    -0x1(%rcx),%edx
  400fe9:       e8 e0 ff ff ff          callq  400fce <func4>
  400fee:       01 c0                   add    %eax,%eax
  400ff0:       eb 15                   jmp    401007 <func4+0x39>
  400ff2:       b8 00 00 00 00          mov    $0x0,%eax
  400ff7:       39 f9                   cmp    %edi,%ecx
  400ff9:       7d 0c                   jge    401007 <func4+0x39>
  400ffb:       8d 71 01                lea    0x1(%rcx),%esi
  400ffe:       e8 cb ff ff ff          callq  400fce <func4>
  401003:       8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
  401007:       48 83 c4 08             add    $0x8,%rsp
  40100b:       c3                      retq
```

Ahhhh, recursion, right? However, you do need to handle recursion actually. Let's have a look at the phase_4 function.

```
000000000040100c <phase_4>:
  40100c:       48 83 ec 18             sub    $0x18,%rsp
  401010:       48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
  401015:       48 8d 54 24 08          lea    0x8(%rsp),%rdx
  40101a:       be cf 25 40 00          mov    $0x4025cf,%esi
  40101f:       b8 00 00 00 00          mov    $0x0,%eax
  401024:       e8 c7 fb ff ff          callq  400bf0
<__isoc99_sscanf@plt>
  401029:       83 f8 02                cmp    $0x2,%eax
  40102c:       75 07                   jne    401035 <phase_4+0x29>
  40102e:       83 7c 24 08 0e          cmpl   $0xe,0x8(%rsp)
  401033:       76 05                   jbe    40103a <phase_4+0x2e>
  401035:       e8 00 04 00 00          callq  40143a <explode_bomb>
  40103a:       ba 0e 00 00 00          mov    $0xe,%edx
  40103f:       be 00 00 00 00          mov    $0x0,%esi
  401044:       8b 7c 24 08             mov    0x8(%rsp),%edi
  401048:       e8 81 ff ff ff          callq  400fce <func4>
  40104d:       85 c0                   test   %eax,%eax
  40104f:       75 07                   jne    401058 <phase_4+0x4c>
  401051:       83 7c 24 0c 00          cmpl   $0x0,0xc(%rsp)
  401056:       74 05                   je     40105d <phase_4+0x51>
  401058:       e8 dd 03 00 00          callq  40143a <explode_bomb>
  40105d:       48 83 c4 18             add    $0x18,%rsp
  401061:       c3                      retq
```

As an experienced engineer, I believe you can figure out that there are two arguments, each of which should be integers. Moreover, it's obvious that the second one must be zero being aware of the line

```
  401056:       74 05                   je     40105d <phase_4+0x51>
  401058:       e8 dd 03 00 00          callq  40143a <explode_bomb>
```

So the problem becomes easier. The problem requires that the return value of the func4 should also be zero. Thinking of the func4 function, we put two lines together to see more clearly

```
  400fe2:       39 f9                   cmp    %edi,%ecx
  ......
  400ff2:       b8 00 00 00 00          mov    $0x0,%eax
  400ff7:       39 f9                   cmp    %edi,%ecx
```

These lines indicate that if the first argument equal the last one(right before this line), then we get 0. From phase_4, we call the four arguments of func4 to be a, b(known, 0), c(known, 14), d(known, 0). Going through func4, we get the value of d at 400ff7 and 400fe2 to be (14 + 0) >> 1 = 7. So a should be 7, too. Then you get the answer to be the pair(7, 0).

Phase 5

This part is a little bit trickier. The code shows as follows:

```
0000000000401062 <phase_5>:
  401062:       53                      push   %rbx
  401063:       48 83 ec 20             sub    $0x20,%rsp
  401067:       48 89 fb                mov    %rdi,%rbx
  40106a:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  401071:       00 00
  401073:       48 89 44 24 18          mov    %rax,0x18(%rsp)
  401078:       31 c0                   xor    %eax,%eax
  40107a:       e8 9c 02 00 00          callq  40131b <string_length>
  40107f:       83 f8 06                cmp    $0x6,%eax
  401082:       74 4e                   je     4010d2 <phase_5+0x70>
  401084:       e8 b1 03 00 00          callq  40143a <explode_bomb>
  401089:       eb 47                   jmp    4010d2 <phase_5+0x70>
  40108b:       0f b6 0c 03             movzbl (%rbx,%rax,1),%ecx
  40108f:       88 0c 24                mov    %cl,(%rsp)
  401092:       48 8b 14 24             mov    (%rsp),%rdx
  401096:       83 e2 0f                and    $0xf,%edx
  401099:       0f b6 92 b0 24 40 00    movzbl 0x4024b0(%rdx),%edx
  4010a0:       88 54 04 10             mov    %dl,0x10(%rsp,%rax,1)
  4010a4:       48 83 c0 01             add    $0x1,%rax
  4010a8:       48 83 f8 06             cmp    $0x6,%rax
  4010ac:       75 dd                   jne    40108b <phase_5+0x29>
  4010ae:       c6 44 24 16 00          movb   $0x0,0x16(%rsp)
  4010b3:       be 5e 24 40 00          mov    $0x40245e,%esi
  4010b8:       48 8d 7c 24 10          lea    0x10(%rsp),%rdi
  4010bd:       e8 76 02 00 00          callq  401338 <strings_not_equal>
  4010c2:       85 c0                   test   %eax,%eax
  4010c4:       74 13                   je     4010d9 <phase_5+0x77>
```

```
  4010c6:       e8 6f 03 00 00          callq  40143a <explode_bomb>
  4010cb:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
  4010d0:       eb 07                   jmp    4010d9 <phase_5+0x77>
  4010d2:       b8 00 00 00 00          mov    $0x0,%eax
  4010d7:       eb b2                   jmp    40108b <phase_5+0x29>
  4010d9:       48 8b 44 24 18          mov    0x18(%rsp),%rax
  4010de:       64 48 33 04 25 28 00    xor    %fs:0x28,%rax
  4010e5:       00 00
  4010e7:       74 05                   je     4010ee <phase_5+0x8c>
  4010e9:       e8 42 fa ff ff          callq  400b30
<__stack_chk_fail@plt>
  4010ee:       48 83 c4 20             add    $0x20,%rsp
  4010f2:       5b                      pop    %rbx
  4010f3:       c3                      retq
```

After inspecting the code, you should figure out that the length of the string must be 6. Then you set a breakpoint at 4010b3 and find the target string to be "flyers". Up till now, there shouldn't be any difficulties.

alpha	after transition
a	97
i	102
o	108
n	121
e	101
f	114
g	115

Then the tricky part comes. You encounter with a loop and you can't find out what it is doing easily. However, you know that the loop is doing some transitions on your input string.

I try a input sequence "aaaaaa" and get the value after transitions doesn't change at all, which means that the output of a given input is unique. Then you can solve this problem by making a table(Yeah, it may seem silly, but I think it's the most convenient way). I will list some transitions here:

The ascii code of "flyers" should be "102, 108, 121, 101, 114, 115". You create a table using the method above, and then you get the answer to be "ionefg".

Phase 6

This part is really long. So now we take a look at this phase .There have to many codes here so it's the complicated once ,So we have to solve it step by step.

```
00000000004010f4 <phase_6>:
  4010f4:       41 56                   push   %r14
  4010f6:       41 55                   push   %r13
  4010f8:       41 54                   push   %r12
  4010fa:       55                      push   %rbp
  4010fb:       53                      push   %rbx
  4010fc:       48 83 ec 50             sub    $0x50,%rsp
  401100:       49 89 e5                mov    %rsp,%r13
  401103:       48 89 e6                mov    %rsp,%rsi
  401106:       e8 51 03 00 00          callq  40145c <read_six_numbers>
  40110b:       49 89 e6                mov    %rsp,%r14
  40110e:       41 bc 00 00 00 00       mov    $0x0,%r12d
  401114:       4c 89 ed                mov    %r13,%rbp
  401117:       41 8b 45 00             mov    0x0(%r13),%eax
  40111b:       83 e8 01                sub    $0x1,%eax
  40111e:       83 f8 05                cmp    $0x5,%eax
  401121:       76 05                   jbe    401128 <phase_6+0x34>
  401123:       e8 12 03 00 00          callq  40143a <explode_bomb>
  401128:       41 83 c4 01             add    $0x1,%r12d
  40112c:       41 83 fc 06             cmp    $0x6,%r12d
  401130:       74 21                   je     401153 <phase_6+0x5f>
  401132:       44 89 e3                mov    %r12d,%ebx
  401135:       48 63 c3                movslq %ebx,%rax
  401138:       8b 04 84                mov    (%rsp,%rax,4),%eax
  40113b:       39 45 00                cmp    %eax,0x0(%rbp)
  40113e:       75 05                   jne    401145 <phase_6+0x51>
  401140:       e8 f5 02 00 00          callq  40143a <explode_bomb>
  401145:       83 c3 01                add    $0x1,%ebx
  401148:       83 fb 05                cmp    $0x5,%ebx
  40114b:       7e e8                   jle    401135 <phase_6+0x41>
```

```
40114d:       49 83 c5 04             add    $0x4,%r13
401151:       eb c1                   jmp    401114 <phase_6+0x20>
401153:       48 8d 74 24 18          lea    0x18(%rsp),%rsi
401158:       4c 89 f0                mov    %r14,%rax
40115b:       b9 07 00 00 00          mov    $0x7,%ecx
401160:       89 ca                   mov    %ecx,%edx
401162:       2b 10                   sub    (%rax),%edx
401164:       89 10                   mov    %edx,(%rax)
401166:       48 83 c0 04             add    $0x4,%rax
40116a:       48 39 f0                cmp    %rsi,%rax
40116d:       75 f1                   jne    401160 <phase_6+0x6c>
40116f:       be 00 00 00 00          mov    $0x0,%esi
401174:       eb 21                   jmp    401197 <phase_6+0xa3>
401176:       48 8b 52 08             mov    0x8(%rdx),%rdx
40117a:       83 c0 01                add    $0x1,%eax
40117d:       39 c8                   cmp    %ecx,%eax
40117f:       75 f5                   jne    401176 <phase_6+0x82>
401181:       eb 05                   jmp    401188 <phase_6+0x94>
401183:       ba d0 32 60 00          mov    $0x6032d0,%edx
401188:       48 89 54 74 20          mov    %rdx,0x20(%rsp,%rsi,2)
40118d:       48 83 c6 04             add    $0x4,%rsi
401191:       48 83 fe 18             cmp    $0x18,%rsi
401195:       74 14                   je     4011ab <phase_6+0xb7>
401197:       8b 0c 34                mov    (%rsp,%rsi,1),%ecx
40119a:       83 f9 01                cmp    $0x1,%ecx
40119d:       7e e4                   jle    401183 <phase_6+0x8f>
40119f:       b8 01 00 00 00          mov    $0x1,%eax
4011a4:       ba d0 32 60 00          mov    $0x6032d0,%edx
4011a9:       eb cb                   jmp    401176 <phase_6+0x82>
4011ab:       48 8b 5c 24 20          mov    0x20(%rsp),%rbx
4011b0:       48 8d 44 24 28          lea    0x28(%rsp),%rax
4011b5:       48 8d 74 24 50          lea    0x50(%rsp),%rsi
4011ba:       48 89 d9                mov    %rbx,%rcx
4011bd:       48 8b 10                mov    (%rax),%rdx
4011c0:       48 89 51 08             mov    %rdx,0x8(%rcx)
4011c4:       48 83 c0 08             add    $0x8,%rax
4011c8:       48 39 f0                cmp    %rsi,%rax
4011cb:       74 05                   je     4011d2 <phase_6+0xde>
4011cd:       48 89 d1                mov    %rdx,%rcx
4011d0:       eb eb                   jmp    4011bd <phase_6+0xc9>
4011d2:       48 c7 42 08 00 00 00    movq   $0x0,0x8(%rdx)
4011d9:       00
4011da:       bd 05 00 00 00          mov    $0x5,%ebp
4011df:       48 8b 43 08             mov    0x8(%rbx),%rax
4011e3:       8b 00                   mov    (%rax),%eax
```

```
  4011e5:       39 03                   cmp    %eax,(%rbx)
  4011e7:       7d 05                   jge    4011ee <phase_6+0xfa>
  4011e9:       e8 4c 02 00 00          callq  40143a <explode_bomb>
  4011ee:       48 8b 5b 08             mov    0x8(%rbx),%rbx
  4011f2:       83 ed 01                sub    $0x1,%ebp
  4011f5:       75 e8                   jne    4011df <phase_6+0xeb>
  4011f7:       48 83 c4 50             add    $0x50,%rsp
  4011fb:       5b                      pop    %rbx
  4011fc:       5d                      pop    %rbp
  4011fd:       41 5c                   pop    %r12
  4011ff:       41 5d                   pop    %r13
  401201:       41 5e                   pop    %r14
  401203:       c3                      retq
```

This really has the style of the last question... Looking at it, I feel dizzy. To put it simply, I input six numbers, check whether they are all less than 7 and not equal, and make each number be subtracted by 7 from < + 95 > to < + 128 > to get a sequence。

```
(gdb) x/24xw 0x6032d0
0x6032d0 <node1>:   0x0000014c    0x00000001    0x006032e0    0x00000000
0x6032e0 <node2>:   0x000000a8    0x00000002    0x006032f0    0x00000000
0x6032f0 <node3>:   0x0000039c    0x00000003    0x00603300    0x00000000
0x603300 <node4>:   0x000002b3    0x00000004    0x00603310    0x00000000
0x603310 <node5>:   0x000001dd    0x00000005    0x00603320    0x00000000
0x603320 <node6>:   0x000001bb    0x00000006    0x00000000    0x00000000
```

Later, the code mentions the address 0x6032d0, in which six nodes are stored in the list. The structure is as follows.

```
struct node{
    int num;
    int order;
    node *next;
};
```

The sequence processed before is the sequence number list arranged by these nodes from large to small. Later, the program will adjust the sequence and check it. For this list, the order from big to small is 3, 4, 5, 6, 1, 2, but before each number is subtracted by 7, so the answer is 4 3 2 1 6 5。 Now the bomb have defused!!