Bomb_lab

> 作者：Xiaoma
>
> 完成时间：2023.1.2

实验目的

> 通过给定的bomb.c了解6个phase的结构，并通过反汇编破解并拆除炸弹

环境

Ubuntu18.04 + gdb

实验步骤与内容

首先我们运行可执行文件bomb，我们会得到如下结果

```
Starting program: /home/xiaoma/Desktop/CSAPP_LAB/Bomb_lab/solution/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
```

该实验需要我们在运行bomb时在6个phase输入6个字符串，来拆除炸弹，如果当前截断炸弹拆除成功，则执行下一个阶段，若拆除失败，则会提示

```
Starting program: /home/xiaoma/Desktop/CSAPP_LAB/Bomb_lab/solution/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
1

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

phase_1

我们通过如下步骤进行反汇编

1. 使用gdb运行可执行文件

```
gdb ./bomb
```

```
xiaoma@ubuntu:~/Desktop/CSAPP_LAB/Bomb_lab/solution$ gdb ./bomb
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./bomb...done.
(gdb)
```

2. 在phase_1的位置设置断点，启动程序，程序执行至断点产生中断

```
break 73
r
```

```
(gdb) break 73
Breakpoint 1 at 0x400e32: file bomb.c, line 73.
(gdb) r
Starting program: /home/xiaoma/Desktop/CSAPP_LAB/Bomb_lab/solution/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!

Breakpoint 1, main (argc=<optimized out>, argv=<optimized out>) at bomb.c:73
73          input = read_line();             /* Get input                   */
(gdb)
```

3. 进行反汇编，左侧箭头为pc指向位置

```
disassemble $pc
```

```
Dump of assembler code for function main:
   0x0000000000400da0 <+0>:     push   %rbx
   0x0000000000400da1 <+1>:     cmp    $0x1,%edi
   0x0000000000400da4 <+4>:     jne    0x400db6 <main+22>
   0x0000000000400da6 <+6>:     mov    0x20299b(%rip),%rax        # 0x603748 <stdin@@GLIBC_2.2.5>
   0x0000000000400dad <+13>:    mov    %rax,0x2029b4(%rip)        # 0x603768 <infile>
   0x0000000000400db4 <+20>:    jmp    0x400e19 <main+121>
   0x0000000000400db6 <+22>:    mov    %rsi,%rbx
   0x0000000000400db9 <+25>:    cmp    $0x2,%edi
   0x0000000000400dbc <+28>:    jne    0x400df8 <main+88>
   0x0000000000400dbe <+30>:    mov    0x8(%rsi),%rdi
   0x0000000000400dc2 <+34>:    mov    $0x4022b4,%esi
   0x0000000000400dc7 <+39>:    callq  0x400c10 <fopen@plt>
   0x0000000000400dcc <+44>:    mov    %rax,0x202995(%rip)        # 0x603768 <infile>
   0x0000000000400dd3 <+51>:    test   %rax,%rax
   0x0000000000400dd6 <+54>:    jne    0x400e19 <main+121>
   0x0000000000400dd8 <+56>:    mov    0x8(%rbx),%rcx
   0x0000000000400ddc <+60>:    mov    (%rbx),%rdx
   0x0000000000400ddf <+63>:    mov    $0x4022b6,%esi
   0x0000000000400de4 <+68>:    mov    $0x1,%edi
   0x0000000000400de9 <+73>:    callq  0x400c00 <__printf_chk@plt>
   0x0000000000400dee <+78>:    mov    $0x8,%edi
   0x0000000000400df3 <+83>:    callq  0x400c20 <exit@plt>
   0x0000000000400df8 <+88>:    mov    (%rsi),%rdx
   0x0000000000400dfb <+91>:    mov    $0x4022d3,%esi
   0x0000000000400e00 <+96>:    mov    $0x1,%edi
   0x0000000000400e05 <+101>:   mov    $0x0,%eax
   0x0000000000400e0a <+106>:   callq  0x400c00 <__printf_chk@plt>
   0x0000000000400e0f <+111>:   mov    $0x8,%edi
   0x0000000000400e14 <+116>:   callq  0x400c20 <exit@plt>
   0x0000000000400e19 <+121>:   callq  0x4013a2 <initialize_bomb>
   0x0000000000400e1e <+126>:   mov    $0x402338,%edi
   0x0000000000400e23 <+131>:   callq  0x400b10 <puts@plt>
   0x0000000000400e28 <+136>:   mov    $0x402378,%edi
   0x0000000000400e2d <+141>:   callq  0x400b10 <puts@plt>
=> 0x0000000000400e32 <+146>:   callq  0x40149e <read_line>
   0x0000000000400e37 <+151>:   mov    %rax,%rdi
   0x0000000000400e3a <+154>:   callq  0x400ee0 <phase_1>
   0x0000000000400e3f <+159>:   callq  0x4015c4 <phase_defused>
   0x0000000000400e44 <+164>:   mov    $0x4023a8,%edi
   0x0000000000400e49 <+169>:   callq  0x400b10 <puts@plt>
   0x0000000000400e4e <+174>:   callq  0x40149e <read_line>
   0x0000000000400e53 <+179>:   mov    %rax,%rdi
   0x0000000000400e56 <+182>:   callq  0x400efc <phase_2>
   0x0000000000400e5b <+187>:   callq  0x4015c4 <phase_defused>
   0x0000000000400e60 <+192>:   mov    $0x4022ed,%edi
   0x0000000000400e65 <+197>:   callq  0x400b10 <puts@plt>
   0x0000000000400e6a <+202>:   callq  0x40149e <read_line>
   0x0000000000400e6f <+207>:   mov    %rax,%rdi
   0x0000000000400e72 <+210>:   callq  0x400f43 <phase_3>
   0x0000000000400e77 <+215>:   callq  0x4015c4 <phase_defused>
   0x0000000000400e7c <+220>:   mov    $0x40230b,%edi
   0x0000000000400e81 <+225>:   callq  0x400b10 <puts@plt>
   0x0000000000400e86 <+230>:   callq  0x40149e <read_line>
   0x0000000000400e8b <+235>:   mov    %rax,%rdi
   0x0000000000400e8e <+238>:   callq  0x40100c <phase_4>
   0x0000000000400e93 <+243>:   callq  0x4015c4 <phase_defused>
   0x0000000000400e98 <+248>:   mov    $0x4023d8,%edi
   0x0000000000400e9d <+253>:   callq  0x400b10 <puts@plt>
   0x0000000000400ea2 <+258>:   callq  0x40149e <read_line>
   0x0000000000400ea7 <+263>:   mov    %rax,%rdi
   0x0000000000400eaa <+266>:   callq  0x401062 <phase_5>
   0x0000000000400eaf <+271>:   callq  0x4015c4 <phase_defused>
   0x0000000000400eb4 <+276>:   mov    $0x40231a,%edi
   0x0000000000400eb9 <+281>:   callq  0x400b10 <puts@plt>
   0x0000000000400ebe <+286>:   callq  0x40149e <read_line>
   0x0000000000400ec3 <+291>:   mov    %rax,%rdi
   0x0000000000400ec6 <+294>:   callq  0x4010f4 <phase_6>
---Type <return> to continue, or q <return> to quit---
```

4. 由上图可知，phase_1的地址为`0x400ee0`，对phase_1进行反汇编

```
disassemble 0x400ee0
```

```
Dump of assembler code for function phase_1:
   0x0000000000400ee0 <+0>:     sub    $0x8,%rsp
   0x0000000000400ee4 <+4>:     mov    $0x402400,%esi
   0x0000000000400ee9 <+9>:     callq  0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   %eax,%eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    callq  0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    $0x8,%rsp
   0x0000000000400efb <+27>:    retq
End of assembler dump.
```

以上是反汇编phase_1的步骤，此后不在赘述。

接下来对phase_1进行分析

- 为函数分配栈帧
- 将0x402400传入%esi，并将其作为参数调用函数strings_not_equal，
- 判断返回值，若ZF=0，则进行跳转
- 释放栈帧，返回结果

对strings_not_equal进行反汇编

```
Dump of assembler code for function strings_not_equal:
   0x0000000000401338 <+0>:     push   %r12
   0x000000000040133a <+2>:     push   %rbp
   0x000000000040133b <+3>:     push   %rbx
   0x000000000040133c <+4>:     mov    %rdi,%rbx
   0x000000000040133f <+7>:     mov    %rsi,%rbp
   0x0000000000401342 <+10>:    callq  0x40131b <string_length>
   0x0000000000401347 <+15>:    mov    %eax,%r12d
   0x000000000040134a <+18>:    mov    %rbp,%rdi
   0x000000000040134d <+21>:    callq  0x40131b <string_length>
   0x0000000000401352 <+26>:    mov    $0x1,%edx
   0x0000000000401357 <+31>:    cmp    %eax,%r12d
   0x000000000040135a <+34>:    jne    0x40139b <strings_not_equal+99>
   0x000000000040135c <+36>:    movzbl (%rbx),%eax
   0x000000000040135f <+39>:    test   %al,%al
   0x0000000000401361 <+41>:    je     0x401388 <strings_not_equal+80>
   0x0000000000401363 <+43>:    cmp    0x0(%rbp),%al
   0x0000000000401366 <+46>:    je     0x401372 <strings_not_equal+58>
   0x0000000000401368 <+48>:    jmp    0x40138f <strings_not_equal+87>
   0x000000000040136a <+50>:    cmp    0x0(%rbp),%al
   0x000000000040136d <+53>:    nopl   (%rax)
   0x0000000000401370 <+56>:    jne    0x401396 <strings_not_equal+94>
   0x0000000000401372 <+58>:    add    $0x1,%rbx
   0x0000000000401376 <+62>:    add    $0x1,%rbp
   0x000000000040137a <+66>:    movzbl (%rbx),%eax
   0x000000000040137d <+69>:    test   %al,%al
   0x000000000040137f <+71>:    jne    0x40136a <strings_not_equal+50>
   0x0000000000401381 <+73>:    mov    $0x0,%edx
   0x0000000000401386 <+78>:    jmp    0x40139b <strings_not_equal+99>
   0x0000000000401388 <+80>:    mov    $0x0,%edx
   0x000000000040138d <+85>:    jmp    0x40139b <strings_not_equal+99>
   0x000000000040138f <+87>:    mov    $0x1,%edx
   0x0000000000401394 <+92>:    jmp    0x40139b <strings_not_equal+99>
   0x0000000000401396 <+94>:    mov    $0x1,%edx
   0x000000000040139b <+99>:    mov    %edx,%eax
   0x000000000040139d <+101>:   pop    %rbx
   0x000000000040139e <+102>:   pop    %rbp
   0x000000000040139f <+103>:   pop    %r12
   0x00000000004013a1 <+105>:   retq
End of assembler dump.
```

通过阅读代码可知，该函数比较输入字符串与已知字符串，若两字符串相同，则返回0，否则返回1。

所以在phase_1，我们输入的字符串与给定字符串相同时，炸弹被拆除。

获得给定字符串即位置`0x402400`的内容

```
x/s 0x402400
```

```
(gdb) x/s 0x402400
0x402400:       "Border relations with Canada have never been better."
(gdb)
```

则在第一阶段输入`Border relations with Canada have never been better.`

```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
```

phase_1完成

phase_2

对phase_2进行反汇编

```
Dump of assembler code for function phase_2:
   0x0000000000400efc <+0>:     push   %rbp
   0x0000000000400efd <+1>:     push   %rbx
   0x0000000000400efe <+2>:     sub    $0x28,%rsp
   0x0000000000400f02 <+6>:     mov    %rsp,%rsi
   0x0000000000400f05 <+9>:     callq  0x40145c <read_six_numbers>
   0x0000000000400f0a <+14>:    cmpl   $0x1,(%rsp)
   0x0000000000400f0e <+18>:    je     0x400f30 <phase_2+52>
   0x0000000000400f10 <+20>:    callq  0x40143a <explode_bomb>
   0x0000000000400f15 <+25>:    jmp    0x400f30 <phase_2+52>
   0x0000000000400f17 <+27>:    mov    -0x4(%rbx),%eax
   0x0000000000400f1a <+30>:    add    %eax,%eax
   0x0000000000400f1c <+32>:    cmp    %eax,(%rbx)
   0x0000000000400f1e <+34>:    je     0x400f25 <phase_2+41>
   0x0000000000400f20 <+36>:    callq  0x40143a <explode_bomb>
   0x0000000000400f25 <+41>:    add    $0x4,%rbx
   0x0000000000400f29 <+45>:    cmp    %rbp,%rbx
   0x0000000000400f2c <+48>:    jne    0x400f17 <phase_2+27>
   0x0000000000400f2e <+50>:    jmp    0x400f3c <phase_2+64>
   0x0000000000400f30 <+52>:    lea    0x4(%rsp),%rbx
   0x0000000000400f35 <+57>:    lea    0x18(%rsp),%rbp
   0x0000000000400f3a <+62>:    jmp    0x400f17 <phase_2+27>
   0x0000000000400f3c <+64>:    add    $0x28,%rsp
   0x0000000000400f40 <+68>:    pop    %rbx
   0x0000000000400f41 <+69>:    pop    %rbp
   0x0000000000400f42 <+70>:    retq
End of assembler dump.
```

- 保存被调用者的寄存器的值
- 为函数分配栈帧
- 将栈顶指针传入`%rsi`，并将其作为参数调用`read_six_numbers`
- 比较0x1与栈顶中的值，若相等则跳转至`0x400f30`，若不相等则炸弹被引爆
- `0x400f30`中指令将`%rsp + 0x4`传入`%rbx`
- 将`%rsp+0x18`传入`%rbp`
- 跳转至`0x400f17`，将`(%rbx-0x4)`传入`%eax`
- 将`%eax`中的值翻倍后与`(%rbx)`进行比较，若相等则跳转至`0x400f25`，若不相等，则炸弹被引爆
- `0x400f25`中指令执行`%rbx <- %rbx + 0x4`
- 比较`%rbp`,`%rbx`，若相等则释放栈帧，返回结果，若不相等，则跳转至`0x400f25`

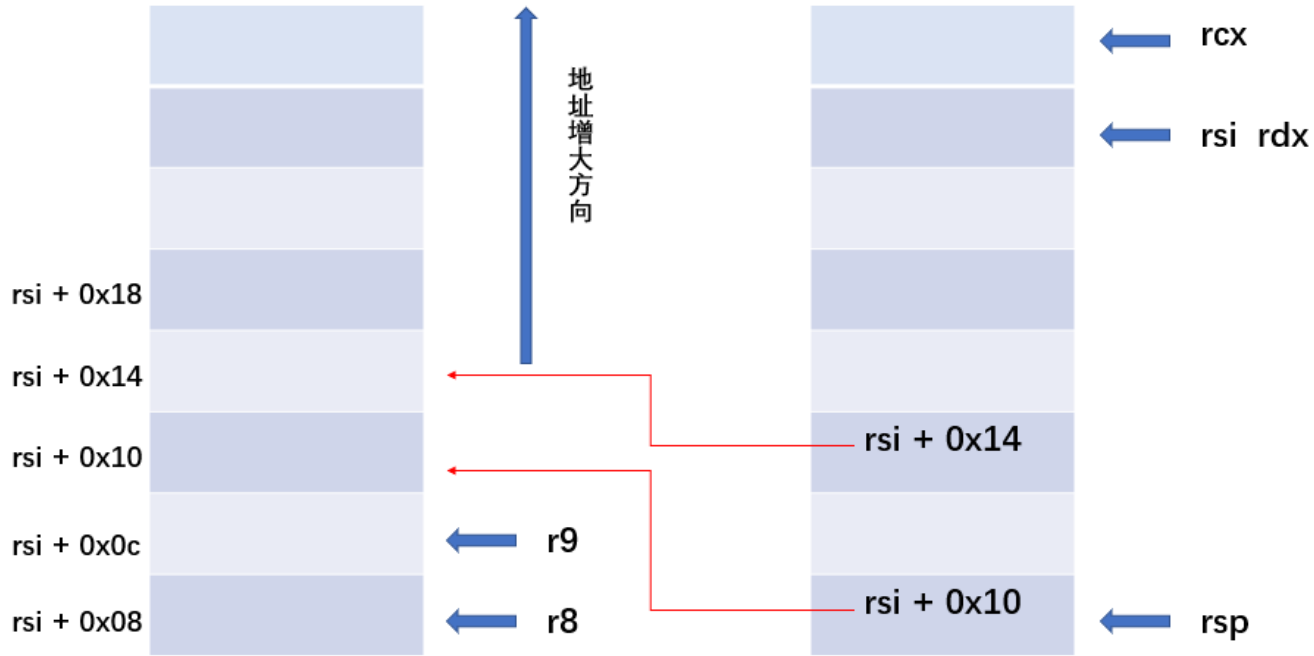
读完该程序以后，可以发现程序似乎是在循环比较栈中的数字，由于`%rbp`中的值为`%rsp + 0x18`，可知循环需要进行6次。

即程序依次比较栈中的数字，栈顶数字为1，剩余5个数字依次为前一个数字的二倍，则6个数字分别为`1 2 4 8 16 32`。

为了验证判断，对`read_six_numbers`进行反汇编

```
Dump of assembler code for function read_six_numbers:
   0x000000000040145c <+0>:     sub    $0x18,%rsp
   0x0000000000401460 <+4>:     mov    %rsi,%rdx
   0x0000000000401463 <+7>:     lea    0x4(%rsi),%rcx
   0x0000000000401467 <+11>:    lea    0x14(%rsi),%rax
   0x000000000040146b <+15>:    mov    %rax,0x8(%rsp)
   0x0000000000401470 <+20>:    lea    0x10(%rsi),%rax
   0x0000000000401474 <+24>:    mov    %rax,(%rsp)
   0x0000000000401478 <+28>:    lea    0xc(%rsi),%r9
   0x000000000040147c <+32>:    lea    0x8(%rsi),%r8
   0x0000000000401480 <+36>:    mov    $0x4025c3,%esi
   0x0000000000401485 <+41>:    mov    $0x0,%eax
   0x000000000040148a <+46>:    callq  0x400bf0 <__isoc99_sscanf@plt>
   0x000000000040148f <+51>:    cmp    $0x5,%eax
   0x0000000000401492 <+54>:    jg     0x401499 <read_six_numbers+61>
   0x0000000000401494 <+56>:    callq  0x40143a <explode_bomb>
   0x0000000000401499 <+61>:    add    $0x18,%rsp
   0x000000000040149d <+65>:    retq
End of assembler dump.
```

在执行`x401480`中的指令之前，寄存器指向的内容如图所示



则6个数字存储的位置按照输入顺序分别为

`%rsi %rsi+0x4 %rsi+0x8 %rsi+0xc %rsi+0x10 %rsi+0x14`

当其返回时，调用者的`%rsp`恰好和被调用函数中`%rsi`相等，则判断正确

```
1 2 4 8 16 32
That's number 2.  Keep going!
```

phase_2完成

phase_3

对phase_3进行反汇编

```
Dump of assembler code for function phase_3:
   0x0000000000400f43 <+0>:     sub    $0x18,%rsp
   0x0000000000400f47 <+4>:     lea    0xc(%rsp),%rcx
   0x0000000000400f4c <+9>:     lea    0x8(%rsp),%rdx
   0x0000000000400f51 <+14>:    mov    $0x4025cf,%esi
   0x0000000000400f56 <+19>:    mov    $0x0,%eax
   0x0000000000400f5b <+24>:    callq  0x400bf0 <__isoc99_sscanf@plt>
   0x0000000000400f60 <+29>:    cmp    $0x1,%eax
   0x0000000000400f63 <+32>:    jg     0x400f6a <phase_3+39>
   0x0000000000400f65 <+34>:    callq  0x40143a <explode_bomb>
   0x0000000000400f6a <+39>:    cmpl   $0x7,0x8(%rsp)
   0x0000000000400f6f <+44>:    ja     0x400fad <phase_3+106>
   0x0000000000400f71 <+46>:    mov    0x8(%rsp),%eax
   0x0000000000400f75 <+50>:    jmpq   *0x402470(,%rax,8)
   0x0000000000400f7c <+57>:    mov    $0xcf,%eax
   0x0000000000400f81 <+62>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f83 <+64>:    mov    $0x2c3,%eax
   0x0000000000400f88 <+69>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f8a <+71>:    mov    $0x100,%eax
   0x0000000000400f8f <+76>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f91 <+78>:    mov    $0x185,%eax
   0x0000000000400f96 <+83>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f98 <+85>:    mov    $0xce,%eax
   0x0000000000400f9d <+90>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f9f <+92>:    mov    $0x2aa,%eax
   0x0000000000400fa4 <+97>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400fa6 <+99>:    mov    $0x147,%eax
   0x0000000000400fab <+104>:   jmp    0x400fbe <phase_3+123>
   0x0000000000400fad <+106>:   callq  0x40143a <explode_bomb>
   0x0000000000400fb2 <+111>:   mov    $0x0,%eax
   0x0000000000400fb7 <+116>:   jmp    0x400fbe <phase_3+123>
   0x0000000000400fb9 <+118>:   mov    $0x137,%eax
   0x0000000000400fbe <+123>:   cmp    0xc(%rsp),%eax
   0x0000000000400fc2 <+127>:   je     0x400fc9 <phase_3+134>
   0x0000000000400fc4 <+129>:   callq  0x40143a <explode_bomb>
   0x0000000000400fc9 <+134>:   add    $0x18,%rsp
   0x0000000000400fcd <+138>:   retq
End of assembler dump.
```

- 为函数分配栈帧
- `%rcx <- (%rsp + 0x8),%rdx <- (%rsp + 0xc)`
- 读入两个整数
- 若整数的数量小于2，则炸弹被引爆

- 将第一个数`(%rsp + 0x8)`与`0x7`比较，若前者大于后者，则炸弹被引爆
- 已知`%rax`为输入的返回值，即第一个数，则程序跳转至`(8 * %rax + 0x402470)`

  已知第一个数字的范围为0-7，则计算结果分别为

  `0x402470 0x402478 0x402480 0x402488 0x402490 0x402498 0x4024a0 0x4024a8`

  而这8个地址中存储的内容为

```
(gdb) x/x 0x402470
0x402470:       0x00400f7c
(gdb) x/x 0x402478
0x402478:       0x00400fb9
(gdb) x/x 0x402480
0x402480:       0x00400f83
(gdb) x/x 0x402488
0x402488:       0x00400f8a
(gdb) x/x 0x402490
0x402490:       0x00400f91
(gdb) x/x 0x402498
0x402498:       0x00400f98
(gdb) x/x 0x4024a0
0x4024a0:       0x00400f9f
(gdb) x/x 0x4024a8
0x4024a8:       0x00400fa6
```

- 分别考虑以上8中情况，发现程序将输入的第二个数与`%eax`中的值进行比较，而8中情况对应8个不同的值，分别为

  `0xcf 0x137 0x2c3 0x100 0x185 0xce 0x2aa 0x147`

则答案分别为

`0 207 1 311 2 707 3 256`

`4 389 5 206 6 682 7 327`

```
7 327
Halfway there!
```

phase_3完成

phase_4

对phase_4进行反汇编

```
Dump of assembler code for function phase_4:
   0x000000000040100c <+0>:     sub    $0x18,%rsp
   0x0000000000401010 <+4>:     lea    0xc(%rsp),%rcx
   0x0000000000401015 <+9>:     lea    0x8(%rsp),%rdx
   0x000000000040101a <+14>:    mov    $0x4025cf,%esi
   0x000000000040101f <+19>:    mov    $0x0,%eax
   0x0000000000401024 <+24>:    callq  0x400bf0 <__isoc99_sscanf@plt>
   0x0000000000401029 <+29>:    cmp    $0x2,%eax
   0x000000000040102c <+32>:    jne    0x401035 <phase_4+41>
   0x000000000040102e <+34>:    cmpl   $0xe,0x8(%rsp)
   0x0000000000401033 <+39>:    jbe    0x40103a <phase_4+46>
   0x0000000000401035 <+41>:    callq  0x40143a <explode_bomb>
   0x000000000040103a <+46>:    mov    $0xe,%edx
   0x000000000040103f <+51>:    mov    $0x0,%esi
   0x0000000000401044 <+56>:    mov    0x8(%rsp),%edi
   0x0000000000401048 <+60>:    callq  0x400fce <func4>
   0x000000000040104d <+65>:    test   %eax,%eax
   0x000000000040104f <+67>:    jne    0x401058 <phase_4+76>
   0x0000000000401051 <+69>:    cmpl   $0x0,0xc(%rsp)
   0x0000000000401056 <+74>:    je     0x40105d <phase_4+81>
   0x0000000000401058 <+76>:    callq  0x40143a <explode_bomb>
   0x000000000040105d <+81>:    add    $0x18,%rsp
   0x0000000000401061 <+85>:    retq
End of assembler dump.
```

- 前一部分与phase_3几乎完全相同，输入两个数将其存储在`%rsp + 0x8`，`%rsp + 0xc`中，并判断输入数量是否有效。
- 若第一个数大于`0xe`，则炸弹被引爆
- `%edx = 0xe`,`%esi = 0x0`,`%edi = (%rsp + 0x8)`
- 调用`func_4`
- 若返回值`%eax`不为0，则炸弹被引爆
- 当第二个数不为0时，炸弹被引爆

对`func_4`进行反汇编

```
Dump of assembler code for function func4:
   0x0000000000400fce <+0>:     sub    $0x8,%rsp
   0x0000000000400fd2 <+4>:     mov    %edx,%eax
   0x0000000000400fd4 <+6>:     sub    %esi,%eax
   0x0000000000400fd6 <+8>:     mov    %eax,%ecx
   0x0000000000400fd8 <+10>:    shr    $0x1f,%ecx
   0x0000000000400fdb <+13>:    add    %ecx,%eax
   0x0000000000400fdd <+15>:    sar    %eax
   0x0000000000400fdf <+17>:    lea    (%rax,%rsi,1),%ecx
   0x0000000000400fe2 <+20>:    cmp    %edi,%ecx
   0x0000000000400fe4 <+22>:    jle    0x400ff2 <func4+36>
   0x0000000000400fe6 <+24>:    lea    -0x1(%rcx),%edx
   0x0000000000400fe9 <+27>:    callq  0x400fce <func4>
   0x0000000000400fee <+32>:    add    %eax,%eax
   0x0000000000400ff0 <+34>:    jmp    0x401007 <func4+57>
   0x0000000000400ff2 <+36>:    mov    $0x0,%eax
   0x0000000000400ff7 <+41>:    cmp    %edi,%ecx
   0x0000000000400ff9 <+43>:    jge    0x401007 <func4+57>
   0x0000000000400ffb <+45>:    lea    0x1(%rcx),%esi
   0x0000000000400ffe <+48>:    callq  0x400fce <func4>
   0x0000000000401003 <+53>:    lea    0x1(%rax,%rax,1),%eax
   0x0000000000401007 <+57>:    add    $0x8,%rsp
   0x000000000040100b <+61>:    retq
End of assembler dump.
```

通过观察可知，该函数是一个递归函数

```
int func_4(int edx, int esi, int edi)
{
    eax = edx;
    eax -= esi;
    eax = eax + (eax >> 31);
    eax >>= 1;
    ecx = eax + esi;
    if(edi <= ecx)
    {
        eax = 0;
        if(edi >= ecx)
        {
            return eax;
        }
        else
        {
            esi = ecx + 1;
            eax = func_4(edx, esi, edi);
        }
    }
    else
    {
        edx = ecx - 1;
        eax = func_4(edx, esi, edi);
    }
    return eax;
}
```

我们不需要考虑递归的所有情况，只考虑直接返回0即可，即edx=7时，即输入的第一个数为7。

则答案为7 0

```
7 0
So you got that one.  Try this one.
```

phase_4完成

phase_5

对phase_5进行反汇编

```
Dump of assembler code for function phase_5:
   0x0000000000401062 <+0>:     push   %rbx
   0x0000000000401063 <+1>:     sub    $0x20,%rsp
   0x0000000000401067 <+5>:     mov    %rdi,%rbx
   0x000000000040106a <+8>:     mov    %fs:0x28,%rax
   0x0000000000401073 <+17>:    mov    %rax,0x18(%rsp)
   0x0000000000401078 <+22>:    xor    %eax,%eax
   0x000000000040107a <+24>:    callq  0x40131b <string_length>
   0x000000000040107f <+29>:    cmp    $0x6,%eax
   0x0000000000401082 <+32>:    je     0x4010d2 <phase_5+112>
   0x0000000000401084 <+34>:    callq  0x40143a <explode_bomb>
   0x0000000000401089 <+39>:    jmp    0x4010d2 <phase_5+112>
   0x000000000040108b <+41>:    movzbl (%rbx,%rax,1),%ecx
   0x000000000040108f <+45>:    mov    %cl,(%rsp)
   0x0000000000401092 <+48>:    mov    (%rsp),%rdx
   0x0000000000401096 <+52>:    and    $0xf,%edx
   0x0000000000401099 <+55>:    movzbl 0x4024b0(%rdx),%edx
   0x00000000004010a0 <+62>:    mov    %dl,0x10(%rsp,%rax,1)
   0x00000000004010a4 <+66>:    add    $0x1,%rax
   0x00000000004010a8 <+70>:    cmp    $0x6,%rax
   0x00000000004010ac <+74>:    jne    0x40108b <phase_5+41>
   0x00000000004010ae <+76>:    movb   $0x0,0x16(%rsp)
   0x00000000004010b3 <+81>:    mov    $0x40245e,%esi
   0x00000000004010b8 <+86>:    lea    0x10(%rsp),%rdi
   0x00000000004010bd <+91>:    callq  0x401338 <strings_not_equal>
   0x00000000004010c2 <+96>:    test   %eax,%eax
   0x00000000004010c4 <+98>:    je     0x4010d9 <phase_5+119>
   0x00000000004010c6 <+100>:   callq  0x40143a <explode_bomb>
   0x00000000004010cb <+105>:   nopl   0x0(%rax,%rax,1)
   0x00000000004010d0 <+110>:   jmp    0x4010d9 <phase_5+119>
   0x00000000004010d2 <+112>:   mov    $0x0,%eax
   0x00000000004010d7 <+117>:   jmp    0x40108b <phase_5+41>
   0x00000000004010d9 <+119>:   mov    0x18(%rsp),%rax
   0x00000000004010de <+124>:   xor    %fs:0x28,%rax
   0x00000000004010e7 <+133>:   je     0x4010ee <phase_5+140>
   0x00000000004010e9 <+135>:   callq  0x400b30 <__stack_chk_fail@plt>
   0x00000000004010ee <+140>:   add    $0x20,%rsp
   0x00000000004010f2 <+144>:   pop    %rbx
   0x00000000004010f3 <+145>:   retq
End of assembler dump.
```

- 为调用者保存%rbx
- 为函数分配栈帧
- %rbx <- %rdi
- (%rsp + 0x18) <- (fs:0x28)，FS:0x28是在Linux上存储一个特殊的哨兵堆栈保护值，该指令的目的是保护堆栈
- 将%eax清零
- 判断字符串长度，若长度不为6，则炸弹被引爆，否则跳转至0x4010d2
- 将%eax清零
- 跳转至0x40108b
- %ecx <- (%rbx + %rax)
- (%rsp) <- %cl即将%ecx的低8位传送至栈顶指针指向的位置
- 将栈顶的值的低4位传送至%edx
- 查看0x4024b0的内容

  ```
  (gdb) x/s 0x4024b0
  0x4024b0 <array.3449>:  "maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"
  ```

  则%edx中的值为(0x4024b0 + %rdx)
- 将%ld传送至%rsp + %rax + 0x10，即将上一步得到的内容存储至栈中
- %rax中的值加1
- 判断%rax与6是否相等，即循环需要进行6次
- 将%rsp + 0x16中的内容清零
- 查看0x40245e中的内容

  ```
  (gdb) x/s 0x40245e
  0x40245e:       "flyers"
  ```

  将字符串的首地址传入%esi
- 调用string_not_equal，比较字符串，若相同则返回0，否则炸弹被引爆
- 剩余部分为程序结束所执行的内容

已知最后传入string_not_equal的字符串应该为flyers才能拆除炸弹。

通过阅读代码可知，在循环部分，程序将输入字符的低4位作为索引，来获得0x4024b0中存储的字符串中的对应字符。然后将对应字符按照倒序入栈，即存储位置依次增大。

当循环结束时，从%rsp + 0x10开始依次存储6个字符，此6个字符组成的字符串即为flyers。

flyers的6个字符依次对应maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?的9,15,14,5,6,7位。

那么我们输入的6个字符的低4为应依次为1001,1111,1110,0101,0110,0111。

则输入的6个字符应为ionuvw。

```
ionuvw
Good work!  On to the next...
```

phase_5完成

phase_6

对phase_6进行反汇编

```
Dump of assembler code for function phase_6:
   0x00000000004010f4 <+0>:     push   %r14
   0x00000000004010f6 <+2>:     push   %r13
   0x00000000004010f8 <+4>:     push   %r12
   0x00000000004010fa <+6>:     push   %rbp
   0x00000000004010fb <+7>:     push   %rbx
   0x00000000004010fc <+8>:     sub    $0x50,%rsp
   0x0000000000401100 <+12>:    mov    %rsp,%r13
   0x0000000000401103 <+15>:    mov    %rsp,%rsi
   0x0000000000401106 <+18>:    callq  0x40145c <read_six_numbers>
   0x000000000040110b <+23>:    mov    %rsp,%r14
   0x000000000040110e <+26>:    mov    $0x0,%r12d
   0x0000000000401114 <+32>:    mov    %r13,%rbp
   0x0000000000401117 <+35>:    mov    0x0(%r13),%eax
   0x000000000040111b <+39>:    sub    $0x1,%eax
   0x000000000040111e <+42>:    cmp    $0x5,%eax
   0x0000000000401121 <+45>:    jbe    0x401128 <phase_6+52>
   0x0000000000401123 <+47>:    callq  0x40143a <explode_bomb>
   0x0000000000401128 <+52>:    add    $0x1,%r12d
   0x000000000040112c <+56>:    cmp    $0x6,%r12d
   0x0000000000401130 <+60>:    je     0x401153 <phase_6+95>
   0x0000000000401132 <+62>:    mov    %r12d,%ebx
   0x0000000000401135 <+65>:    movslq %ebx,%rax
   0x0000000000401138 <+68>:    mov    (%rsp,%rax,4),%eax
   0x000000000040113b <+71>:    cmp    %eax,0x0(%rbp)
   0x000000000040113e <+74>:    jne    0x401145 <phase_6+81>
   0x0000000000401140 <+76>:    callq  0x40143a <explode_bomb>
   0x0000000000401145 <+81>:    add    $0x1,%ebx
   0x0000000000401148 <+84>:    cmp    $0x5,%ebx
   0x000000000040114b <+87>:    jle    0x401135 <phase_6+65>
   0x000000000040114d <+89>:    add    $0x4,%r13
   0x0000000000401151 <+93>:    jmp    0x401114 <phase_6+32>
   0x0000000000401153 <+95>:    lea    0x18(%rsp),%rsi
   0x0000000000401158 <+100>:   mov    %r14,%rax
   0x000000000040115b <+103>:   mov    $0x7,%ecx
   0x0000000000401160 <+108>:   mov    %ecx,%edx
   0x0000000000401162 <+110>:   sub    (%rax),%edx
   0x0000000000401164 <+112>:   mov    %edx,(%rax)
   0x0000000000401166 <+114>:   add    $0x4,%rax
   0x000000000040116a <+118>:   cmp    %rsi,%rax
```

```
   0x000000000040116d <+121>:   jne    0x401160 <phase_6+108>
   0x000000000040116f <+123>:   mov    $0x0,%esi
   0x0000000000401174 <+128>:   jmp    0x401197 <phase_6+163>
   0x0000000000401176 <+130>:   mov    0x8(%rdx),%rdx
   0x000000000040117a <+134>:   add    $0x1,%eax
   0x000000000040117d <+137>:   cmp    %ecx,%eax
   0x000000000040117f <+139>:   jne    0x401176 <phase_6+130>
   0x0000000000401181 <+141>:   jmp    0x401188 <phase_6+148>
   0x0000000000401183 <+143>:   mov    $0x6032d0,%edx
   0x0000000000401188 <+148>:   mov    %rdx,0x20(%rsp,%rsi,2)
   0x000000000040118d <+153>:   add    $0x4,%rsi
   0x0000000000401191 <+157>:   cmp    $0x18,%rsi
   0x0000000000401195 <+161>:   je     0x4011ab <phase_6+183>
   0x0000000000401197 <+163>:   mov    (%rsp,%rsi,1),%ecx
   0x000000000040119a <+166>:   cmp    $0x1,%ecx
   0x000000000040119d <+169>:   jle    0x401183 <phase_6+143>
   0x000000000040119f <+171>:   mov    $0x1,%eax
   0x00000000004011a4 <+176>:   mov    $0x6032d0,%edx
   0x00000000004011a9 <+181>:   jmp    0x401176 <phase_6+130>
   0x00000000004011ab <+183>:   mov    0x20(%rsp),%rbx
   0x00000000004011b0 <+188>:   lea    0x28(%rsp),%rax
   0x00000000004011b5 <+193>:   lea    0x50(%rsp),%rsi
   0x00000000004011ba <+198>:   mov    %rbx,%rcx
   0x00000000004011bd <+201>:   mov    (%rax),%rdx
   0x00000000004011c0 <+204>:   mov    %rdx,0x8(%rcx)
   0x00000000004011c4 <+208>:   add    $0x8,%rax
   0x00000000004011c8 <+212>:   cmp    %rsi,%rax
   0x00000000004011cb <+215>:   je     0x4011d2 <phase_6+222>
   0x00000000004011cd <+217>:   mov    %rdx,%rcx
   0x00000000004011d0 <+220>:   jmp    0x4011bd <phase_6+201>
   0x00000000004011d2 <+222>:   movq   $0x0,0x8(%rdx)
   0x00000000004011da <+230>:   mov    $0x5,%ebp
   0x00000000004011df <+235>:   mov    0x8(%rbx),%rax
   0x00000000004011e3 <+239>:   mov    (%rax),%eax
   0x00000000004011e5 <+241>:   cmp    %eax,(%rbx)
   0x00000000004011e7 <+243>:   jge    0x4011ee <phase_6+250>
   0x00000000004011e9 <+245>:   callq  0x40143a <explode_bomb>
   0x00000000004011ee <+250>:   mov    0x8(%rbx),%rbx
   0x00000000004011f2 <+254>:   sub    $0x1,%ebp
   0x00000000004011f5 <+257>:   jne    0x4011df <phase_6+235>
   0x00000000004011f7 <+259>:   add    $0x50,%rsp
   0x00000000004011fb <+263>:   pop    %rbx
   0x00000000004011fc <+264>:   pop    %rbp
   0x00000000004011fd <+265>:   pop    %r12
   0x00000000004011ff <+267>:   pop    %r13
   0x0000000000401201 <+269>:   pop    %r14
   0x0000000000401203 <+271>:   retq
End of assembler dump.
```

- 为调用者保存寄存器中的内容
- 为函数分配栈帧
- `%r13 <- %rsp`,`%rsi <- %rsp`

- 调用`read_six_numbers`，其已经在phase_2出现过，当其返回时，6个数字存放的位置依次为`%rsp`,`%rsp + 0x4`,`%rsp + 0x8`,`%rsp + 0xc`,`%rsp + 0x10`,`%rsp + 0x14`
- `%r14 <- %rsp`,`%r12d <- 0x0`
- `%rbp <- %r13`
- 将第一个数字传入`%eax`，若该数字大于6，则炸弹被引爆
- 将进行6次循环
- 首先比较其余5个数字是否与第一个相同，若相同，则炸弹被引爆
- 循环跳转至`0x401114`，即继续判断后面的数字是否大于6，继续判断数组中的数是否有两两相同的情况
- 将`%rsi`作为数组遍历结束的标志位，我们可以推测这又是一个循环
- 假设数组中的数为`nums[i]`，首先 `%edx <- 7 - nums[i]`
- 然后`nums[i] <- %edx`
- 由之前的条件可知，经过计算后，数组中的数不相同且都大于0，小于7
- `%esi <- 0x0`
- 跳转至`0x401197`
- 将数组中的元素传送至`%ecx`
- 判断其是否等于1，若等于则跳转至`0x401183`
- 根据下面的代码可知，最多循环6次，则查看`0x6032d0`以及其后面的内容

```
(gdb) x/24x 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
```

  查阅资料可知，存储内容为长度为6的链表，第三个变量值为`next`指针

- 若数组中的元素为1，则栈中`%rsp + 2 * %rsi + 0x20`存储的是第一个节点
- 若`nums[i] > 1`，则指向第一个节点的指针向后移动`nums[i] - 1`次，将该节点存入位置`%rsp + 2 * %rsi + 0x20`，已知数组中元素各不相同，则栈中存放的节点也各不相同
- `%rbx <- (%rsp + 0x20)`
- `%rax <- %rsp + 0x28`
- `%rsi <- %rsp + 0x50`
- 阅读代码可知下面的循环按照节点在栈中的顺序将重排链表
- `(%rdx + 0x8) <- 0x0`

- `%ebp <- 0x5`
- `%rax <- (%rbx + 0x8)`即该寄存器中存储第二个节点的地址值
- `%eax <- (%rax)`该寄存器存储第二个节点的值
- 如果第一个节点的值小于第二个节点，则炸弹被引爆
- 进行5次循环，即当前一个节点的值大于后一个节点时，炸弹才能被拆除。
- 程序结束

我们已经知道了每个节点的值，按照原顺序，依次为`0x14c`,`0xa8`,`0x39c`,`0x2b3`,`0x1dd`,`0x1bb`，当其顺序满足`0x39c`,`0x2b3`,`0x1dd`,`0x1bb`,`0x14c`,`0xa8`时，炸弹被拆除。

即链表重排的顺序为`node[3] -> node[4] -> node[5] -> node[6] -> node[1] -> node[2]`，已知输入的6个数字参与重排链表之前要进行运算`7 - nums[i]`，则答案为`4 3 2 1 6 5`。

```
4 3 2 1 6 5
Congratulations! You've defused the bomb!
```

phase_6完成

secret_phase

在阅读`bomb.c`时，我们在最后发现这样一句话

```
111     /* Wow, they got it!  But isn't something... missing?  Perhaps
112      * something they overlooked?  Mua ha ha ha ha! */
```

尝试反汇编`phase_defused`

```
Dump of assembler code for function phase_defused:
   0x00000000004015c4 <+0>:     sub    $0x78,%rsp
   0x00000000004015c8 <+4>:     mov    %fs:0x28,%rax
   0x00000000004015d1 <+13>:    mov    %rax,0x68(%rsp)
   0x00000000004015d6 <+18>:    xor    %eax,%eax
   0x00000000004015d8 <+20>:    cmpl   $0x6,0x202181(%rip)        # 0x603760 <num_input_strings>
   0x00000000004015df <+27>:    jne    0x40163f <phase_defused+123>
   0x00000000004015e1 <+29>:    lea    0x10(%rsp),%r8
   0x00000000004015e6 <+34>:    lea    0xc(%rsp),%rcx
   0x00000000004015eb <+39>:    lea    0x8(%rsp),%rdx
   0x00000000004015f0 <+44>:    mov    $0x402619,%esi
   0x00000000004015f5 <+49>:    mov    $0x603870,%edi
   0x00000000004015fa <+54>:    callq  0x400bf0 <__isoc99_sscanf@plt>
   0x00000000004015ff <+59>:    cmp    $0x3,%eax
   0x0000000000401602 <+62>:    jne    0x401635 <phase_defused+113>
   0x0000000000401604 <+64>:    mov    $0x402622,%esi
   0x0000000000401609 <+69>:    lea    0x10(%rsp),%rdi
   0x000000000040160e <+74>:    callq  0x401338 <strings_not_equal>
   0x0000000000401613 <+79>:    test   %eax,%eax
   0x0000000000401615 <+81>:    jne    0x401635 <phase_defused+113>
   0x0000000000401617 <+83>:    mov    $0x4024f8,%edi
   0x000000000040161c <+88>:    callq  0x400b10 <puts@plt>
   0x0000000000401621 <+93>:    mov    $0x402520,%edi
   0x0000000000401626 <+98>:    callq  0x400b10 <puts@plt>
   0x000000000040162b <+103>:   mov    $0x0,%eax
   0x0000000000401630 <+108>:   callq  0x401242 <secret_phase>
   0x0000000000401635 <+113>:   mov    $0x402558,%edi
   0x000000000040163a <+118>:   callq  0x400b10 <puts@plt>
   0x000000000040163f <+123>:   mov    0x68(%rsp),%rax
   0x0000000000401644 <+128>:   xor    %fs:0x28,%rax
   0x000000000040164d <+137>:   je     0x401654 <phase_defused+144>
   0x000000000040164f <+139>:   callq  0x400b30 <__stack_chk_fail@plt>
   0x0000000000401654 <+144>:   add    $0x78,%rsp
   0x0000000000401658 <+148>:   retq
End of assembler dump.
```

查看0x402619中的内容

```
(gdb) x/s 0x402619
0x402619:       "%d %d %s"
```

查看0x603870中的内容，显然内容为空

```
(gdb) x/s 0x603870
0x603870 <input_strings+240>:    ""
```

查看0x402622中的内容

```
(gdb) x/s 0x402622
0x402622:       "DrEvil"
```

阅读代码可知，当输入满足两个数字和一个字符串DrEvil时，可以进入secret_phase。

但我们知道phase_3,phase_4的答案都是两个数字，所以在哪个phase输入是未知的。

已知在之前的phase中，输入的内容的地址都会存入%rsi ，尝试在每个phase打印%rsi中的内容

```
Breakpoint 1, 0x000000000040100c in phase_4 ()
(gdb) print /x $rdi
$1 = 0x603870
```

发现phase_4输入的字符串的地址与触发secret_phase需要比较的字符串的地址相同，所以在phase_4输入7 0 DrEvil可以触发secret_phase。

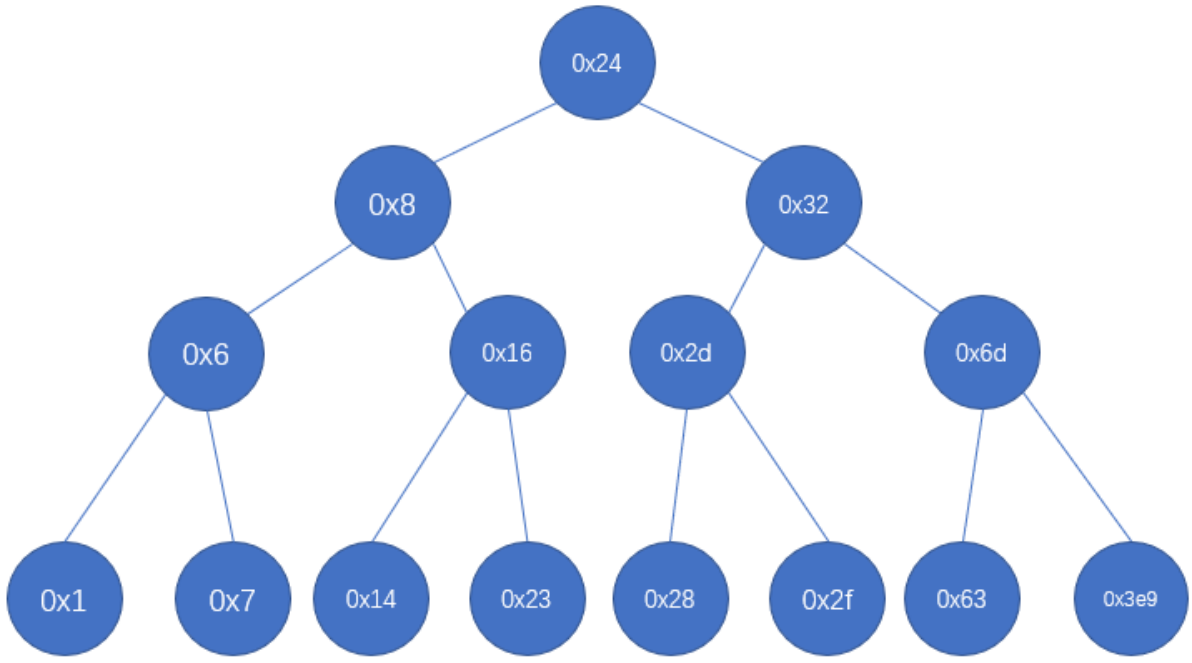
反汇编secret_phase

```
Dump of assembler code for function secret_phase:
   0x0000000000401242 <+0>:     push   %rbx
   0x0000000000401243 <+1>:     callq  0x40149e <read_line>
   0x0000000000401248 <+6>:     mov    $0xa,%edx
   0x000000000040124d <+11>:    mov    $0x0,%esi
   0x0000000000401252 <+16>:    mov    %rax,%rdi
   0x0000000000401255 <+19>:    callq  0x400bd0 <strtol@plt>
   0x000000000040125a <+24>:    mov    %rax,%rbx
   0x000000000040125d <+27>:    lea    -0x1(%rax),%eax
   0x0000000000401260 <+30>:    cmp    $0x3e8,%eax
   0x0000000000401265 <+35>:    jbe    0x40126c <secret_phase+42>
   0x0000000000401267 <+37>:    callq  0x40143a <explode_bomb>
   0x000000000040126c <+42>:    mov    %ebx,%esi
   0x000000000040126e <+44>:    mov    $0x6030f0,%edi
   0x0000000000401273 <+49>:    callq  0x401204 <fun7>
   0x0000000000401278 <+54>:    cmp    $0x2,%eax
   0x000000000040127b <+57>:    je     0x401282 <secret_phase+64>
   0x000000000040127d <+59>:    callq  0x40143a <explode_bomb>
   0x0000000000401282 <+64>:    mov    $0x402438,%edi
   0x0000000000401287 <+69>:    callq  0x400b10 <puts@plt>
   0x000000000040128c <+74>:    callq  0x4015c4 <phase_defused>
   0x0000000000401291 <+79>:    pop    %rbx
   0x0000000000401292 <+80>:    retq
End of assembler dump.
```

- 保存调用者的寄存器中的内容
- `%edx <- 0xa`,`%esi <- 0x0`,`%rdi <- %rax`
- 调用`strtol`将字符串转换成对应的整数
- 该整数若小于1或大于1001时，炸弹被引爆
- `%esi <- %ebx`
- 查看`0x6030f0`中存储的内容

```
(gdb) x/150x 0x6030f0
0x6030f0 <n1>:  0x00000024      0x00000000      0x00603110      0x00000000
0x603100 <n1+16>:       0x00603130      0x00000000      0x00000000      0x00000000
0x603110 <n21>: 0x00000008      0x00000000      0x00603190      0x00000000
0x603120 <n21+16>:      0x00603150      0x00000000      0x00000000      0x00000000
0x603130 <n22>: 0x00000032      0x00000000      0x00603170      0x00000000
0x603140 <n22+16>:      0x006031b0      0x00000000      0x00000000      0x00000000
0x603150 <n32>: 0x00000016      0x00000000      0x00603270      0x00000000
0x603160 <n32+16>:      0x00603230      0x00000000      0x00000000      0x00000000
0x603170 <n33>: 0x0000002d      0x00000000      0x006031d0      0x00000000
0x603180 <n33+16>:      0x00603290      0x00000000      0x00000000      0x00000000
0x603190 <n31>: 0x00000006      0x00000000      0x006031f0      0x00000000
0x6031a0 <n31+16>:      0x00603250      0x00000000      0x00000000      0x00000000
0x6031b0 <n34>: 0x0000006b      0x00000000      0x00603210      0x00000000
0x6031c0 <n34+16>:      0x006032b0      0x00000000      0x00000000      0x00000000
0x6031d0 <n45>: 0x00000028      0x00000000      0x00000000      0x00000000
0x6031e0 <n45+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x6031f0 <n41>: 0x00000001      0x00000000      0x00000000      0x00000000
0x603200 <n41+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x603210 <n47>: 0x00000063      0x00000000      0x00000000      0x00000000
0x603220 <n47+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x603230 <n44>: 0x00000023      0x00000000      0x00000000      0x00000000
0x603240 <n44+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x603250 <n42>: 0x00000007      0x00000000      0x00000000      0x00000000
0x603260 <n42+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x603270 <n43>: 0x00000014      0x00000000      0x00000000      0x00000000
0x603280 <n43+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x603290 <n46>: 0x0000002f      0x00000000      0x00000000      0x00000000
0x6032a0 <n46+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x6032b0 <n48>: 0x000003e9      0x00000000      0x00000000      0x00000000
```

内容按照树结构存储



观察可以发现，其不仅是一棵树，还是二叉搜索树

- 调用fun7，当返回值不为2时，炸弹被引爆
- 程序结束

反汇编fun7

```
Dump of assembler code for function fun7:
   0x0000000000401204 <+0>:     sub    $0x8,%rsp
   0x0000000000401208 <+4>:     test   %rdi,%rdi
   0x000000000040120b <+7>:     je     0x401238 <fun7+52>
   0x000000000040120d <+9>:     mov    (%rdi),%edx
   0x000000000040120f <+11>:    cmp    %esi,%edx
   0x0000000000401211 <+13>:    jle    0x401220 <fun7+28>
   0x0000000000401213 <+15>:    mov    0x8(%rdi),%rdi
   0x0000000000401217 <+19>:    callq  0x401204 <fun7>
   0x000000000040121c <+24>:    add    %eax,%eax
   0x000000000040121e <+26>:    jmp    0x40123d <fun7+57>
   0x0000000000401220 <+28>:    mov    $0x0,%eax
   0x0000000000401225 <+33>:    cmp    %esi,%edx
   0x0000000000401227 <+35>:    je     0x40123d <fun7+57>
   0x0000000000401229 <+37>:    mov    0x10(%rdi),%rdi
   0x000000000040122d <+41>:    callq  0x401204 <fun7>
   0x0000000000401232 <+46>:    lea    0x1(%rax,%rax,1),%eax
   0x0000000000401236 <+50>:    jmp    0x40123d <fun7+57>
   0x0000000000401238 <+52>:    mov    $0xffffffff,%eax
   0x000000000040123d <+57>:    add    $0x8,%rsp
   0x0000000000401241 <+61>:    retq
End of assembler dump.
```

观察可以发现，fun7是一个递归函数

```
/*
struct BTree
{
    int val;
    struct BTree* left, *right;
}
*/

int fun7(BTree* rdi, int esi)
{
    if(!rdi)
    {
        return -1;
    }
    else
    {
        if(rdi -> val <= esi)
        {
            if(rdi -> val == esi)
            {
                return 0;
            }
            else
            {
                return 1 + 2 * fun7(rdi -> right, esi);
            }
        }
```

```
        else
        {
            return 2 * fun7(rdi -> left, esi);
        }
    }
}
```

我们希望返回值为2，则上一次返回值一定为1，即`fun7(rdi -> left, esi)`返回1，推断可知`%esi`中的值为`0x16`。

```
But finding it and solving it are quite different...
22
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```

secret_phase完成

结论分析与体会

本次实验极大的提高了我对汇编程序的阅读理解能力，并且掌握了简单的gdb调试步骤。