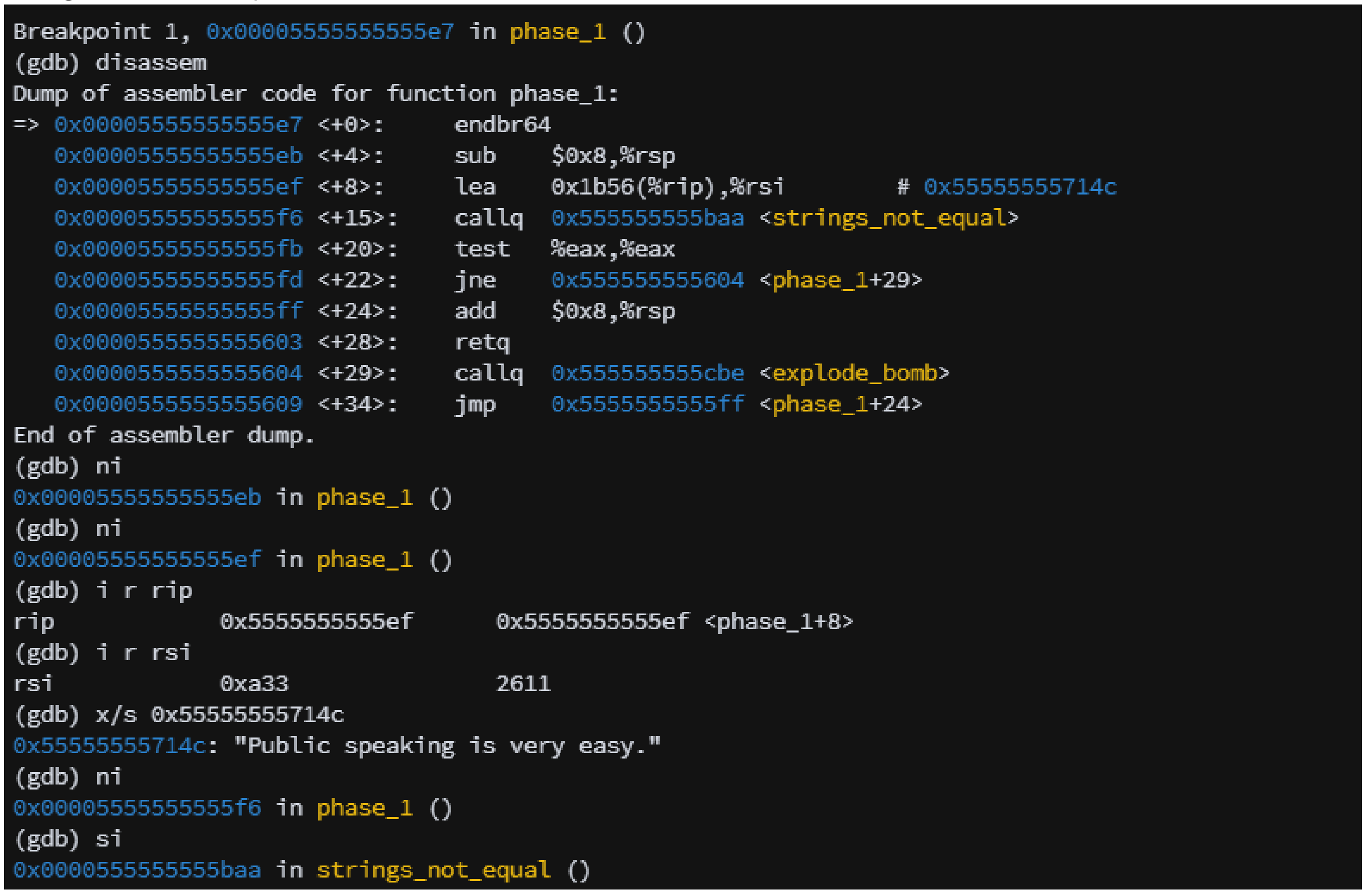
计算机系统第三次实验BombLab

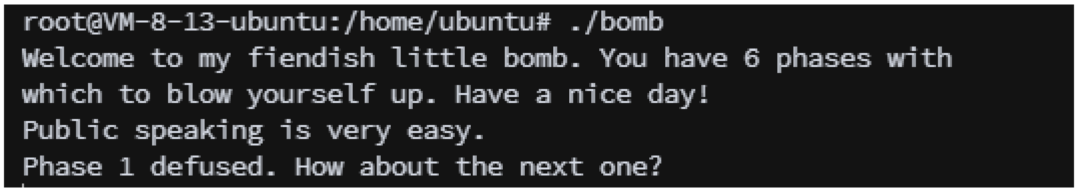
phase_1

使用gdb调试程序在phase_1设置断点



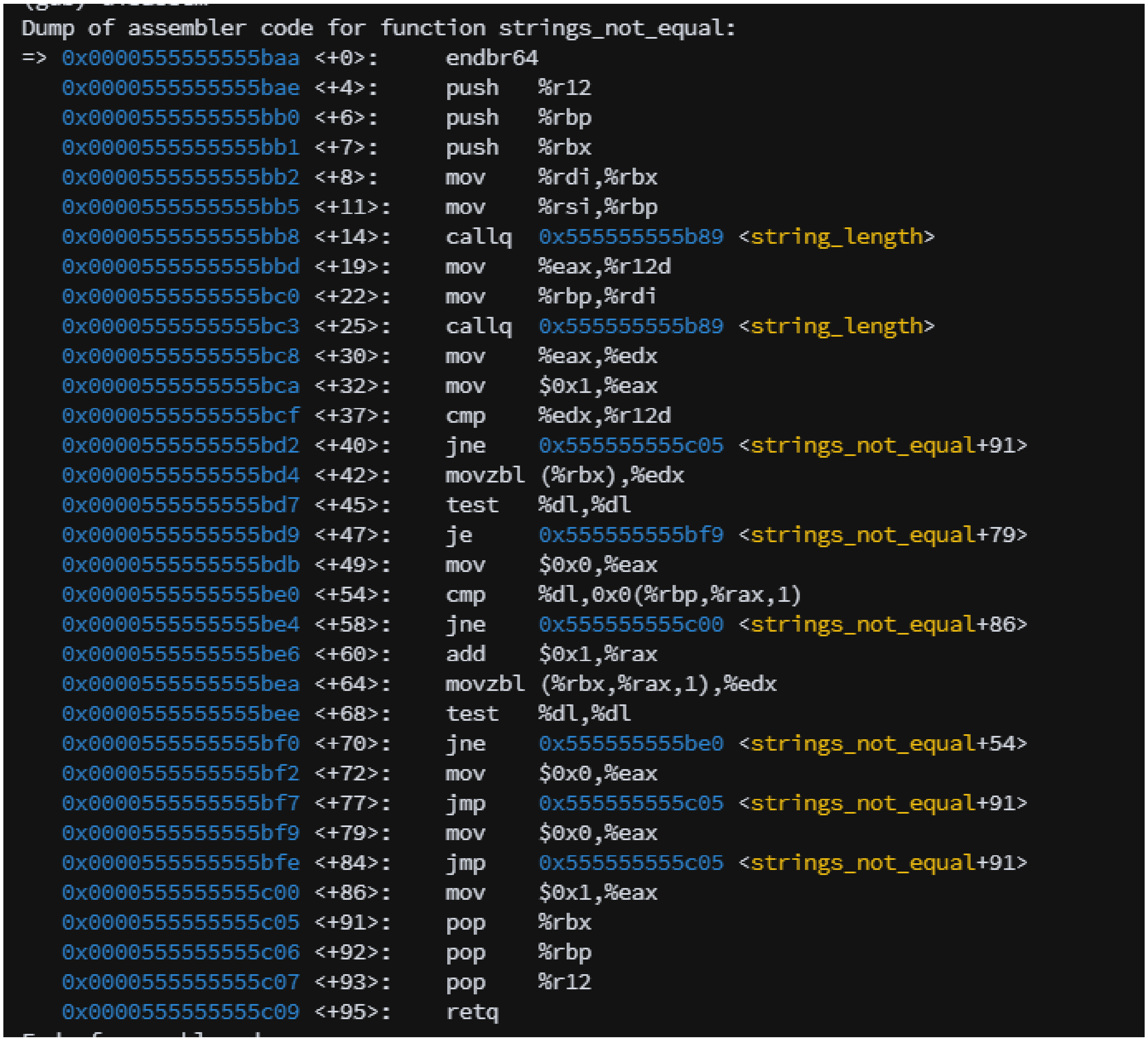
```
Breakpoint 1, 0x00005555555555e7 in phase_1 ()
(gdb) disassem
Dump of assembler code for function phase_1:
=> 0x00005555555555e7 <+0>:     endbr64
   0x00005555555555eb <+4>:     sub    $0x8,%rsp
   0x00005555555555ef <+8>:     lea    0x1b56(%rip),%rsi        # 0x55555555714c
   0x00005555555555f6 <+15>:    callq  0x555555555baa <strings_not_equal>
   0x00005555555555fb <+20>:    test   %eax,%eax
   0x00005555555555fd <+22>:    jne    0x555555555604 <phase_1+29>
   0x00005555555555ff <+24>:    add    $0x8,%rsp
   0x0000555555555603 <+28>:    retq
   0x0000555555555604 <+29>:    callq  0x555555555cbe <explode_bomb>
   0x0000555555555609 <+34>:    jmp    0x5555555555ff <phase_1+24>
End of assembler dump.
(gdb) ni
0x00005555555555eb in phase_1 ()
(gdb) ni
0x00005555555555ef in phase_1 ()
(gdb) i r rip
rip            0x5555555555ef      0x5555555555ef <phase_1+8>
(gdb) i r rsi
rsi            0xa33               2611
(gdb) x/s 0x55555555714c
0x55555555714c: "Public speaking is very easy."
(gdb) ni
0x00005555555555f6 in phase_1 ()
(gdb) si
0x0000555555555baa in strings_not_equal ()
```

查看**rsi**寄存器 发现是一串字符串也就是答案 验证程序如下



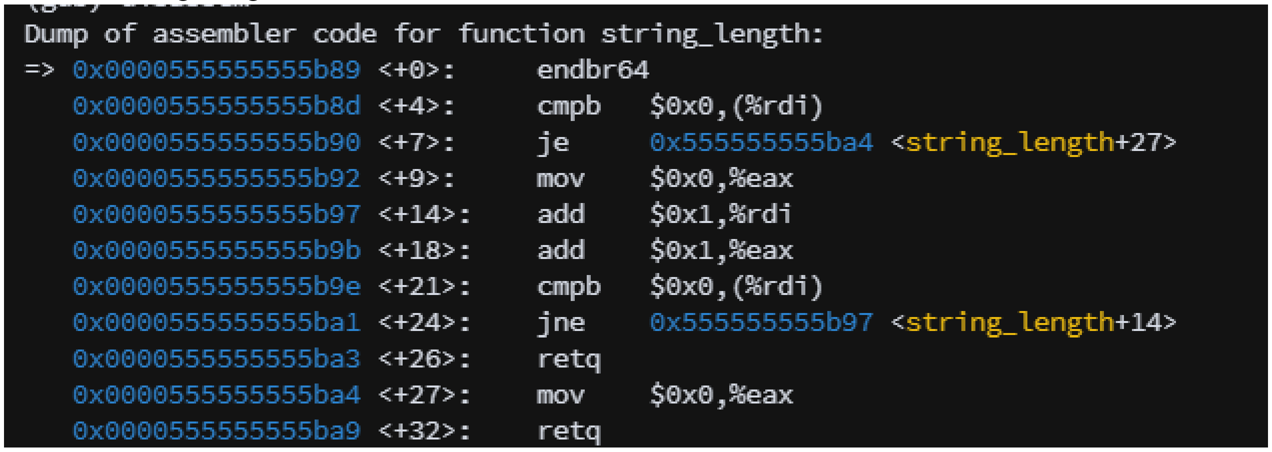
```
root@VM-8-13-ubuntu:/home/ubuntu# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
```

仔细分析一下phase程序，首先开辟栈空间然后 将答案字符串给 **rsi** 寄存器， 然后调用了string_not_equal 函数 step in 看一下函数的内容



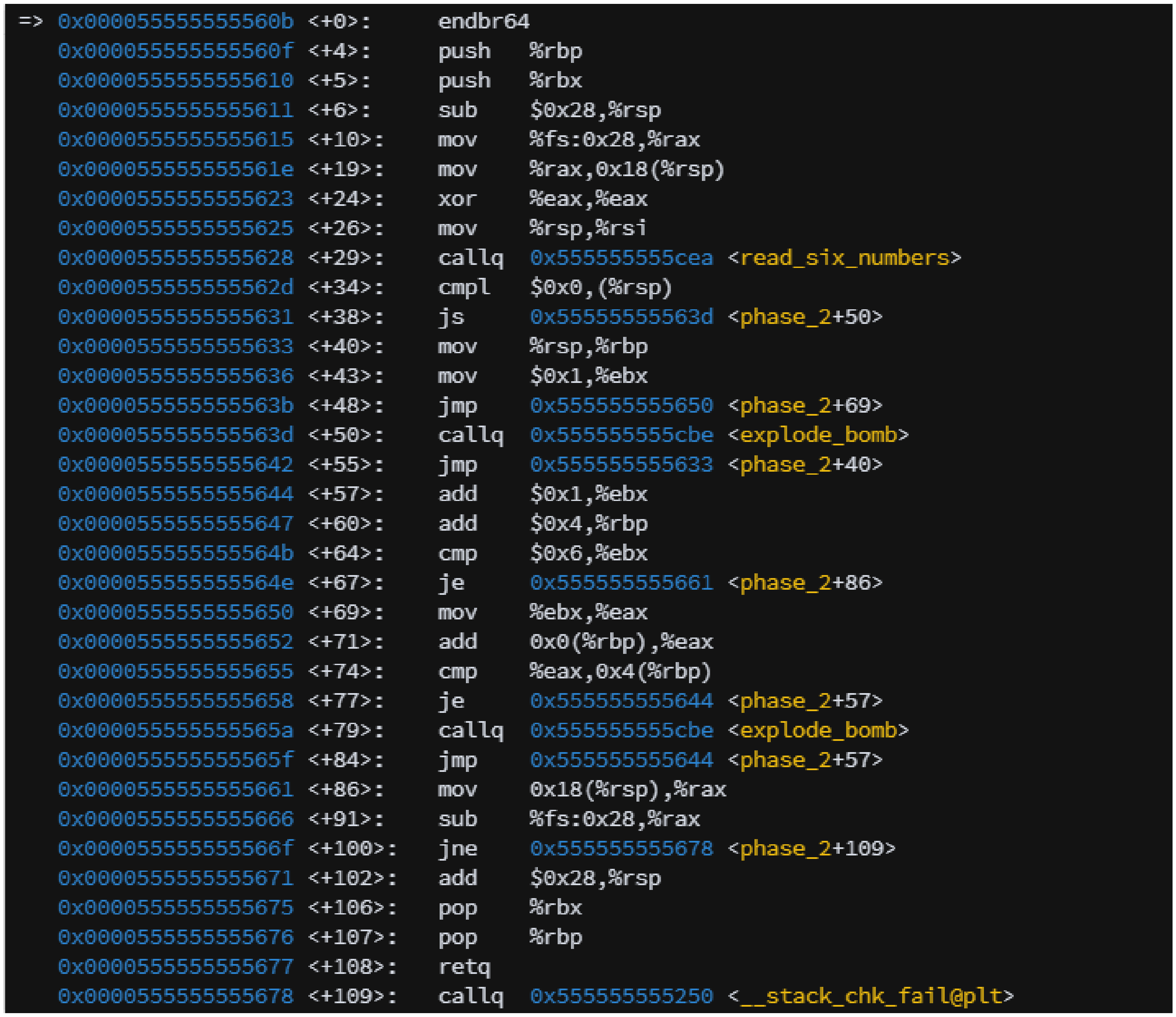
```
Dump of assembler code for function strings_not_equal:
=> 0x0000555555555baa <+0>:     endbr64
   0x0000555555555bae <+4>:     push   %r12
   0x0000555555555bb0 <+6>:     push   %rbp
   0x0000555555555bb1 <+7>:     push   %rbx
   0x0000555555555bb2 <+8>:     mov    %rdi,%rbx
   0x0000555555555bb5 <+11>:    mov    %rsi,%rbp
   0x0000555555555bb8 <+14>:    callq  0x555555555b89 <string_length>
   0x0000555555555bbd <+19>:    mov    %eax,%r12d
   0x0000555555555bc0 <+22>:    mov    %rbp,%rdi
   0x0000555555555bc3 <+25>:    callq  0x555555555b89 <string_length>
   0x0000555555555bc8 <+30>:    mov    %eax,%edx
   0x0000555555555bca <+32>:    mov    $0x1,%eax
   0x0000555555555bcf <+37>:    cmp    %edx,%r12d
   0x0000555555555bd2 <+40>:    jne    0x555555555c05 <strings_not_equal+91>
   0x0000555555555bd4 <+42>:    movzbl (%rbx),%edx
   0x0000555555555bd7 <+45>:    test   %dl,%dl
   0x0000555555555bd9 <+47>:    je     0x555555555bf9 <strings_not_equal+79>
   0x0000555555555bdb <+49>:    mov    $0x0,%eax
   0x0000555555555be0 <+54>:    cmp    %dl,0x0(%rbp,%rax,1)
   0x0000555555555be4 <+58>:    jne    0x555555555c00 <strings_not_equal+86>
   0x0000555555555be6 <+60>:    add    $0x1,%rax
   0x0000555555555bea <+64>:    movzbl (%rbx,%rax,1),%edx
   0x0000555555555bee <+68>:    test   %dl,%dl
   0x0000555555555bf0 <+70>:    jne    0x555555555be0 <strings_not_equal+54>
   0x0000555555555bf2 <+72>:    mov    $0x0,%eax
   0x0000555555555bf7 <+77>:    jmp    0x555555555c05 <strings_not_equal+91>
   0x0000555555555bf9 <+79>:    mov    $0x0,%eax
   0x0000555555555bfe <+84>:    jmp    0x555555555c05 <strings_not_equal+91>
   0x0000555555555c00 <+86>:    mov    $0x1,%eax
   0x0000555555555c05 <+91>:    pop    %rbx
   0x0000555555555c06 <+92>:    pop    %rbp
   0x0000555555555c07 <+93>:    pop    %r12
   0x0000555555555c09 <+95>:    retq
```

我们可以发现phase1往这个函数传了**rdi（第一个保存输入） rsi（第二个保存答案）** 两个参数分别存放在**rbx**和 **rbp**中
然后调用string_length函数比较长度



```
Dump of assembler code for function string_length:
=> 0x0000555555555b89 <+0>:     endbr64
   0x0000555555555b8d <+4>:     cmpb   $0x0,(%rdi)
   0x0000555555555b90 <+7>:     je     0x555555555ba4 <string_length+27>
   0x0000555555555b92 <+9>:     mov    $0x0,%eax
   0x0000555555555b97 <+14>:    add    $0x1,%rdi
   0x0000555555555b9b <+18>:    add    $0x1,%eax
   0x0000555555555b9e <+21>:    cmpb   $0x0,(%rdi)
   0x0000555555555ba1 <+24>:    jne    0x555555555b97 <string_length+14>
   0x0000555555555ba3 <+26>:    retq
   0x0000555555555ba4 <+27>:    mov    $0x0,%eax
   0x0000555555555ba9 <+32>:    retq
```

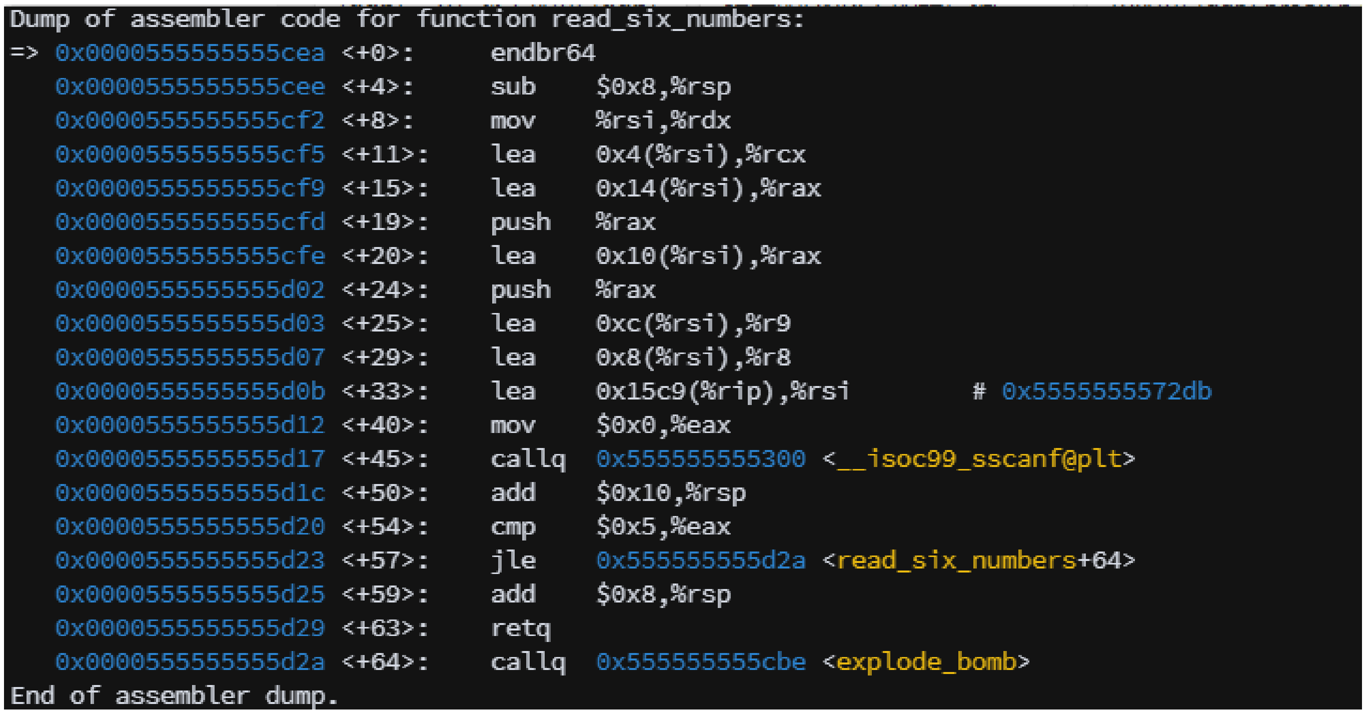
phase2



```
=> 0x000055555555560b <+0>:     endbr64
   0x000055555555560f <+4>:     push   %rbp
   0x0000555555555610 <+5>:     push   %rbx
   0x0000555555555611 <+6>:     sub    $0x28,%rsp
   0x0000555555555615 <+10>:    mov    %fs:0x28,%rax
   0x000055555555561e <+19>:    mov    %rax,0x18(%rsp)
   0x0000555555555623 <+24>:    xor    %eax,%eax
   0x0000555555555625 <+26>:    mov    %rsp,%rsi
   0x0000555555555628 <+29>:    callq  0x555555555cea <read_six_numbers>
   0x000055555555562d <+34>:    cmpl   $0x0,(%rsp)
   0x0000555555555631 <+38>:    js     0x55555555563d <phase_2+50>
   0x0000555555555633 <+40>:    mov    %rsp,%rbp
   0x0000555555555636 <+43>:    mov    $0x1,%ebx
   0x000055555555563b <+48>:    jmp    0x555555555650 <phase_2+69>
   0x000055555555563d <+50>:    callq  0x555555555cbe <explode_bomb>
   0x0000555555555642 <+55>:    jmp    0x555555555633 <phase_2+40>
   0x0000555555555644 <+57>:    add    $0x1,%ebx
   0x0000555555555647 <+60>:    add    $0x4,%rbp
   0x000055555555564b <+64>:    cmp    $0x6,%ebx
   0x000055555555564e <+67>:    je     0x555555555661 <phase_2+86>
   0x0000555555555650 <+69>:    mov    %ebx,%eax
   0x0000555555555652 <+71>:    add    0x0(%rbp),%eax
   0x0000555555555655 <+74>:    cmp    %eax,0x4(%rbp)
   0x0000555555555658 <+77>:    je     0x555555555644 <phase_2+57>
   0x000055555555565a <+79>:    callq  0x555555555cbe <explode_bomb>
   0x000055555555565f <+84>:    jmp    0x555555555644 <phase_2+57>
   0x0000555555555661 <+86>:    mov    0x18(%rsp),%rax
   0x0000555555555666 <+91>:    sub    %fs:0x28,%rax
   0x000055555555566f <+100>:   jne    0x555555555678 <phase_2+109>
   0x0000555555555671 <+102>:   add    $0x28,%rsp
   0x0000555555555675 <+106>:   pop    %rbx
   0x0000555555555676 <+107>:   pop    %rbp
   0x0000555555555677 <+108>:   retq
   0x0000555555555678 <+109>:   callq  0x555555555250 <__stack_chk_fail@plt>
```

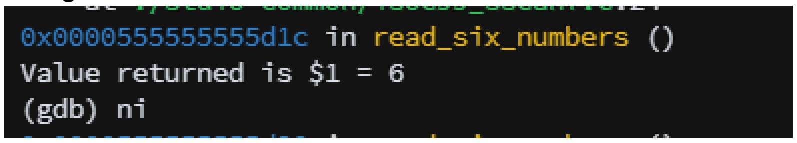
先查看*read_six_numbers*函数

这一部分查看蓝色注释的地址会发现 6个 %d 即提示要输入六个int整数



```
Dump of assembler code for function read_six_numbers:
=> 0x0000555555555cea <+0>:     endbr64
   0x0000555555555cee <+4>:     sub    $0x8,%rsp
   0x0000555555555cf2 <+8>:     mov    %rsi,%rdx
   0x0000555555555cf5 <+11>:    lea    0x4(%rsi),%rcx
   0x0000555555555cf9 <+15>:    lea    0x14(%rsi),%rax
   0x0000555555555cfd <+19>:    push   %rax
   0x0000555555555cfe <+20>:    lea    0x10(%rsi),%rax
   0x0000555555555d02 <+24>:    push   %rax
   0x0000555555555d03 <+25>:    lea    0xc(%rsi),%r9
   0x0000555555555d07 <+29>:    lea    0x8(%rsi),%r8
   0x0000555555555d0b <+33>:    lea    0x15c9(%rip),%rsi        # 0x5555555572db
   0x0000555555555d12 <+40>:    mov    $0x0,%eax
   0x0000555555555d17 <+45>:    callq  0x555555555300 <__isoc99_sscanf@plt>
   0x0000555555555d1c <+50>:    add    $0x10,%rsp
   0x0000555555555d20 <+54>:    cmp    $0x5,%eax
   0x0000555555555d23 <+57>:    jle    0x555555555d2a <read_six_numbers+64>
   0x0000555555555d25 <+59>:    add    $0x8,%rsp
   0x0000555555555d29 <+63>:    retq
   0x0000555555555d2a <+64>:    callq  0x555555555cbe <explode_bomb>
End of assembler dump.
```

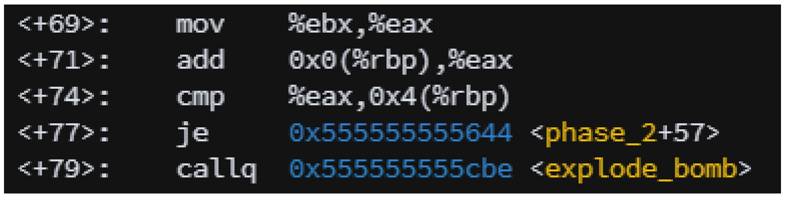
通过gdb查看发现会返回个数



```
0x0000555555555d1c in read_six_numbers ()
Value returned is $1 = 6
(gdb) ni
```

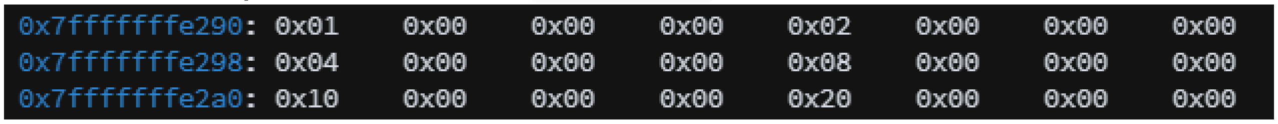
返回后到phase_2+69 也就是核心的一段

这是一个循环 ebx 保存累加的i 以一个数组为例 i从1开始



```
<+69>:    mov    %ebx,%eax
<+71>:    add    0x0(%rbp),%eax
<+74>:    cmp    %eax,0x4(%rbp)
<+77>:    je     0x555555555644 <phase_2+57>
<+79>:    callq  0x555555555cbe <explode_bomb>
```

这是第一次循环 **ebp** 保存了输入的数组 以`1 2 4 8 16 32`(并不是正确答案) 为例

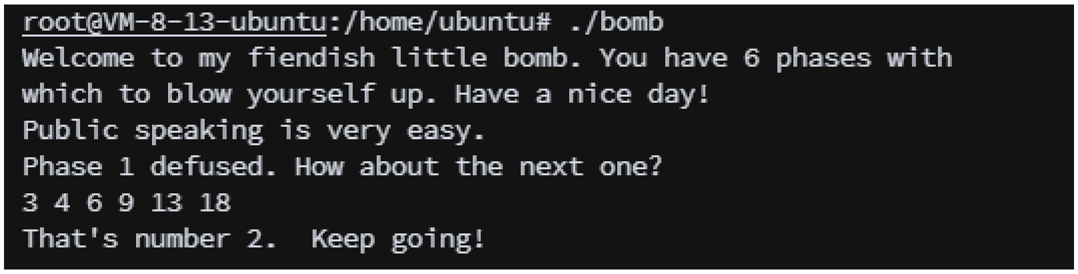


```
0x7fffffffe290: 0x01    0x00    0x00    0x00    0x02    0x00    0x00    0x00
0x7fffffffe298: 0x04    0x00    0x00    0x00    0x08    0x00    0x00    0x00
0x7fffffffe2a0: 0x10    0x00    0x00    0x00    0x20    0x00    0x00    0x00
```

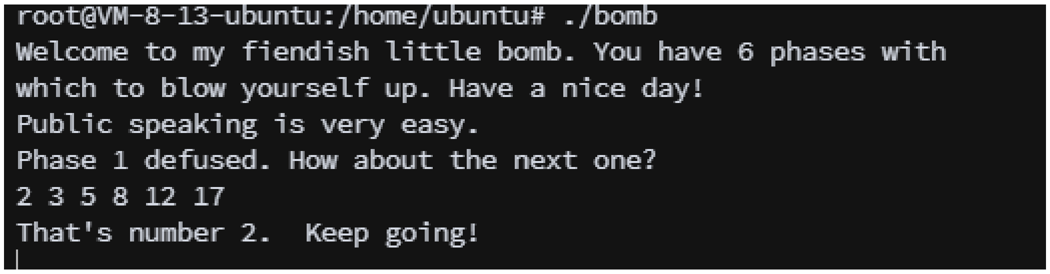
ebp每加4 也即到下一个输出的数

第一次循环就是 1+第一个数=第二个数也即 a[i]+i=a[i+1]

经实验有多种答案

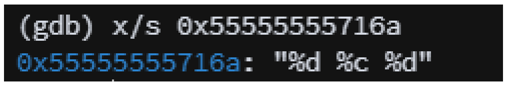


```
root@VM-8-13-ubuntu:/home/ubuntu# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
3 4 6 9 13 18
That's number 2.  Keep going!
```



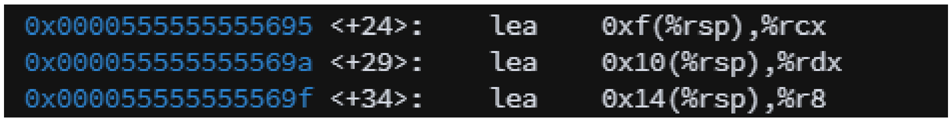
```
root@VM-8-13-ubuntu:/home/ubuntu# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
2 3 5 8 12 17
That's number 2.  Keep going!
```

Phase_3



```
(gdb) x/s 0x55555555716a
0x55555555716a: "%d %c %d"
```

提示我们输入三个一个为数字一个字符一个数字

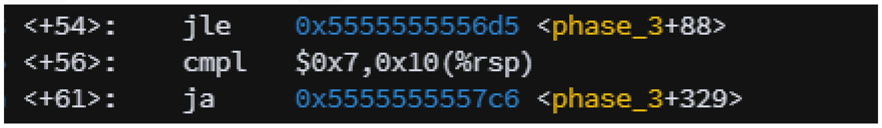


```
0x0000555555555695 <+24>:    lea    0xf(%rsp),%rcx
0x000055555555569a <+29>:    lea    0x10(%rsp),%rdx
0x000055555555569f <+34>:    lea    0x14(%rsp),%r8
```

根据寄存器的含义我们可以知道0xf 对应输入的 `char`

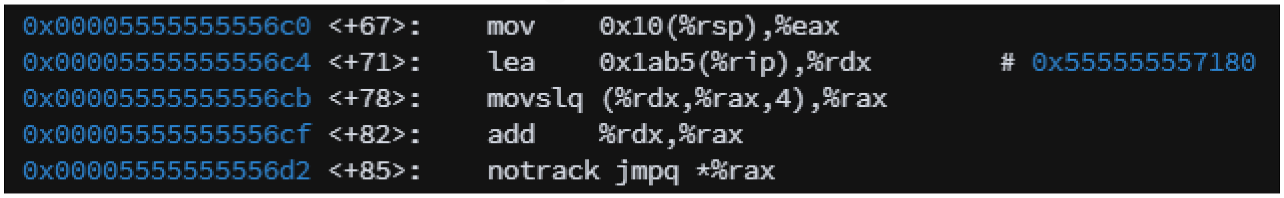
`0x10` 对应输入的第一个 `int`

`0x14` 对应输入的第二个 `int`



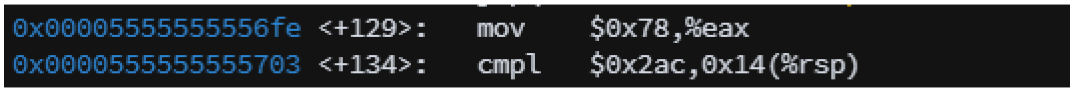
```
<+54>:    jle    0x5555555556d5 <phase_3+88>
<+56>:    cmpl   $0x7,0x10(%rsp)
<+61>:    ja     0x5555555557c6 <phase_3+329>
```

从这里我们可以知道输入的第一个int要小于 `7` ,不然会跳转到炸弹爆炸的位置



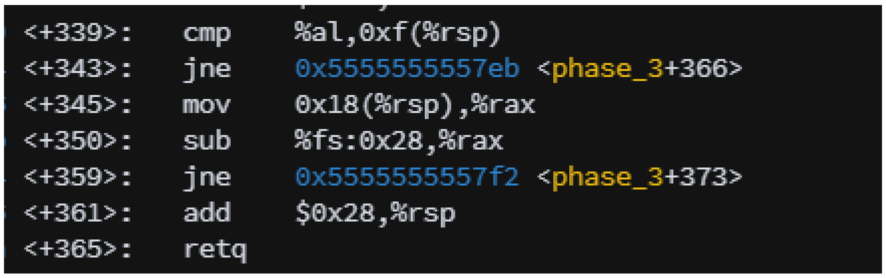
```
0x00005555555556c0 <+67>:    mov    0x10(%rsp),%eax
0x00005555555556c4 <+71>:    lea    0x1ab5(%rip),%rdx        # 0x555555557180
0x00005555555556cb <+78>:    movslq (%rdx,%rax,4),%rax
0x00005555555556cf <+82>:    add    %rdx,%rax
0x00005555555556d2 <+85>:    notrack jmpq *%rax
```

这里类似生成一个跳转表 我们可以根据输入的第一个 `int` 的数的大小在gdb查看*rax*存放的地址
例如我们第一个数输入的是1，就会跳转到下图的地址



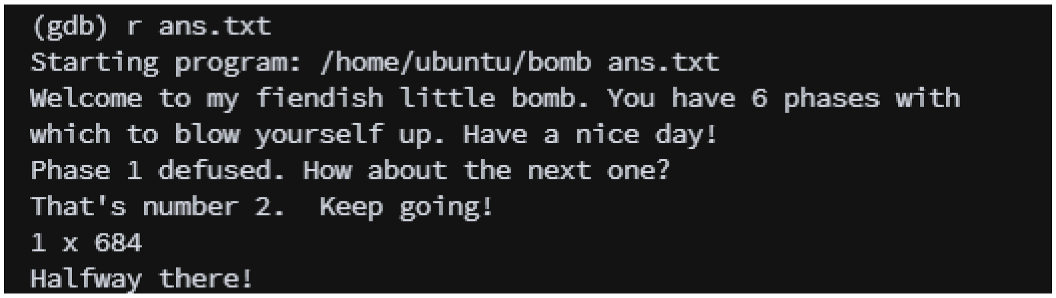
```
0x00005555555556fe <+129>:    mov    $0x78,%eax
0x0000555555555703 <+134>:    cmpl   $0x2ac,0x14(%rsp)
```

此时第二个 `int` 就需要跟2acH（684）相等
然后再次跳转



```
<+339>:   cmp    %al,0xf(%rsp)
<+343>:   jne    0x5555555557eb <phase_3+366>
<+345>:   mov    0x18(%rsp),%rax
<+350>:   sub    %fs:0x28,%rax
<+359>:   jne    0x5555555557f2 <phase_3+373>
<+361>:   add    $0x28,%rsp
<+365>:   retq
```

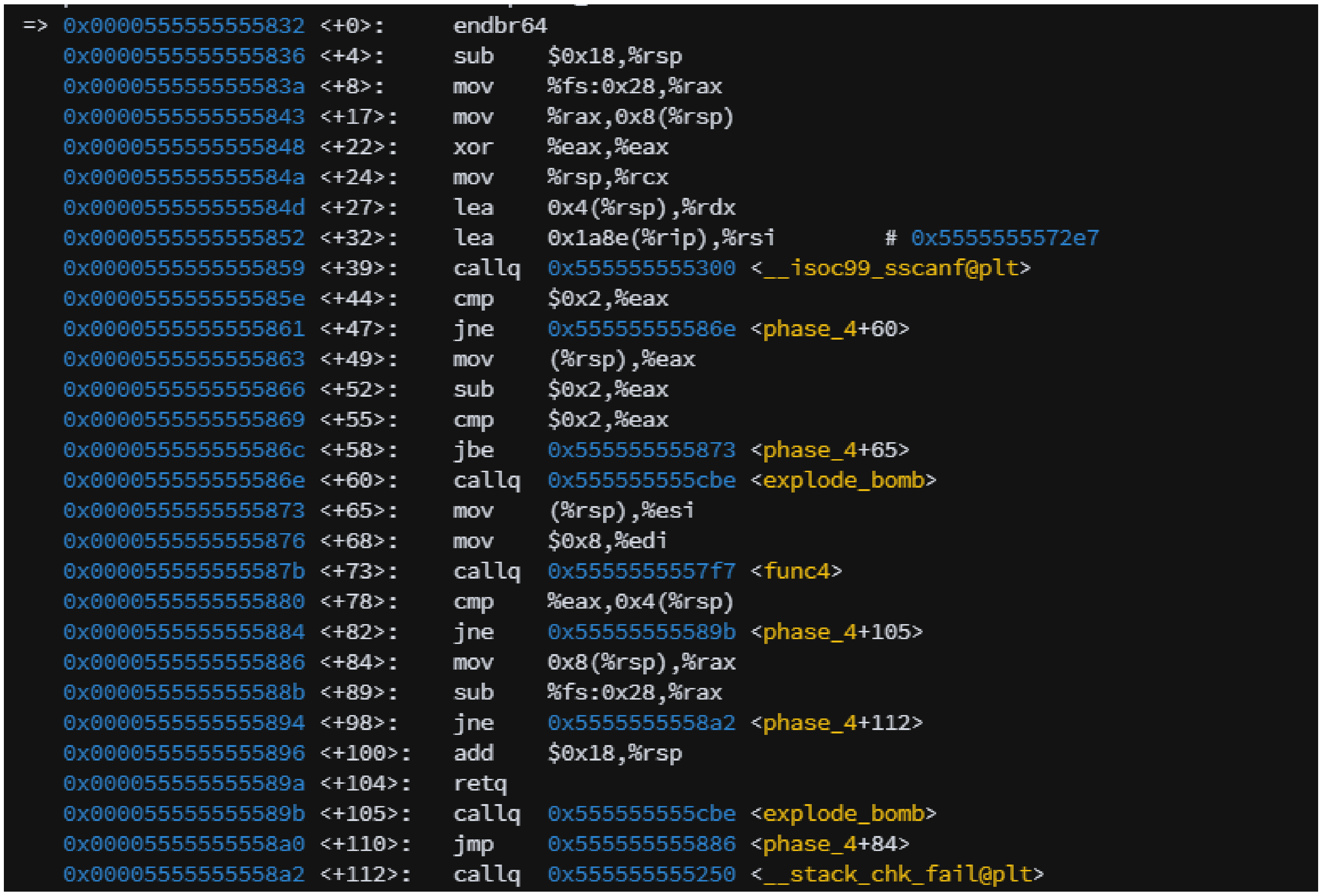
`al` 寄存器存储的是刚刚传给eax的0x78 对应的ASCII码就是 `x`
结合之前的对应
这题的完整答案就是 **1 x 684**
其他对应的答案也成立



```
(gdb) r ans.txt
Starting program: /home/ubuntu/bomb ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
1 x 684
Halfway there!
```

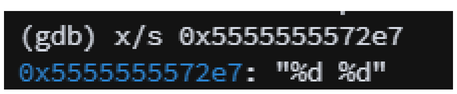
phase_4

查看汇编代码



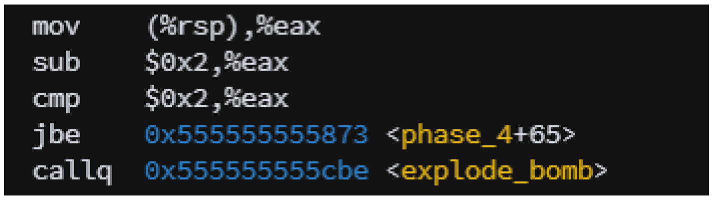
```
=> 0x0000555555555832 <+0>:     endbr64
   0x0000555555555836 <+4>:     sub    $0x18,%rsp
   0x000055555555583a <+8>:     mov    %fs:0x28,%rax
   0x0000555555555843 <+17>:    mov    %rax,0x8(%rsp)
   0x0000555555555848 <+22>:    xor    %eax,%eax
   0x000055555555584a <+24>:    mov    %rsp,%rcx
   0x000055555555584d <+27>:    lea    0x4(%rsp),%rdx
   0x0000555555555852 <+32>:    lea    0x1a8e(%rip),%rsi        # 0x5555555572e7
   0x0000555555555859 <+39>:    callq  0x555555555300 <__isoc99_sscanf@plt>
   0x000055555555585e <+44>:    cmp    $0x2,%eax
   0x0000555555555861 <+47>:    jne    0x55555555586e <phase_4+60>
   0x0000555555555863 <+49>:    mov    (%rsp),%eax
   0x0000555555555866 <+52>:    sub    $0x2,%eax
   0x0000555555555869 <+55>:    cmp    $0x2,%eax
   0x000055555555586c <+58>:    jbe    0x555555555873 <phase_4+65>
   0x000055555555586e <+60>:    callq  0x555555555cbe <explode_bomb>
   0x0000555555555873 <+65>:    mov    (%rsp),%esi
   0x0000555555555876 <+68>:    mov    $0x8,%edi
   0x000055555555587b <+73>:    callq  0x5555555557f7 <func4>
   0x0000555555555880 <+78>:    cmp    %eax,0x4(%rsp)
   0x0000555555555884 <+82>:    jne    0x55555555589b <phase_4+105>
   0x0000555555555886 <+84>:    mov    0x8(%rsp),%rax
   0x000055555555588b <+89>:    sub    %fs:0x28,%rax
   0x0000555555555894 <+98>:    jne    0x5555555558a2 <phase_4+112>
   0x0000555555555896 <+100>:   add    $0x18,%rsp
   0x000055555555589a <+104>:   retq
   0x000055555555589b <+105>:   callq  0x555555555cbe <explode_bomb>
   0x00005555555558a0 <+110>:   jmp    0x555555555886 <phase_4+84>
   0x00005555555558a2 <+112>:   callq  0x555555555250 <__stack_chk_fail@plt>
```

查看 提示输入两个整数



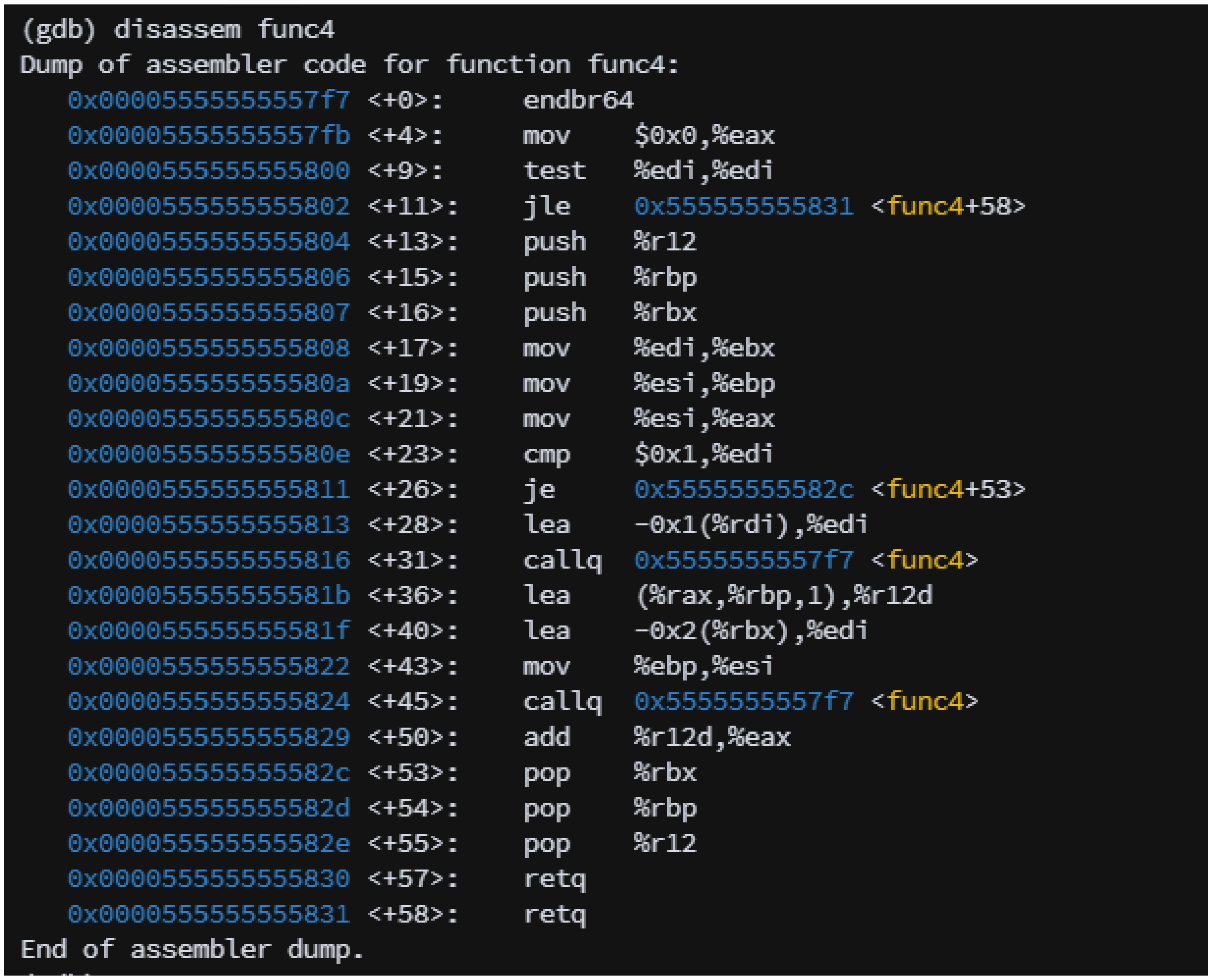
```
(gdb) x/s 0x5555555572e7
0x5555555572e7: "%d %d"
```

第二个参数需要小于4 否则会爆炸



```
 mov     (%rsp),%eax
 sub     $0x2,%eax
 cmp     $0x2,%eax
 jbe     0x555555555873 <phase_4+65>
 callq   0x555555555cbe <explode_bomb>
```

将第二个参数放入 `esi` `edi` 放 `8`
然后进入递归函数 `func4`

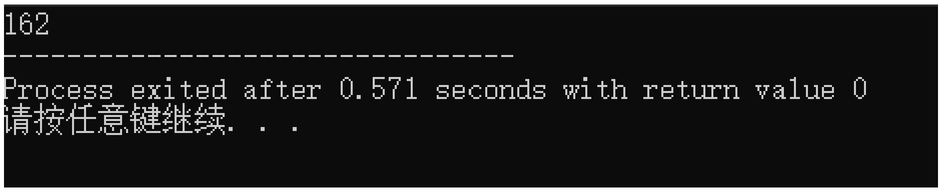


```
(gdb) disassem func4
Dump of assembler code for function func4:
   0x00005555555557f7 <+0>:     endbr64
   0x00005555555557fb <+4>:     mov    $0x0,%eax
   0x0000555555555800 <+9>:     test   %edi,%edi
   0x0000555555555802 <+11>:    jle    0x555555555831 <func4+58>
   0x0000555555555804 <+13>:    push   %r12
   0x0000555555555806 <+15>:    push   %rbp
   0x0000555555555807 <+16>:    push   %rbx
   0x0000555555555808 <+17>:    mov    %edi,%ebx
   0x000055555555580a <+19>:    mov    %esi,%ebp
   0x000055555555580c <+21>:    mov    %esi,%eax
   0x000055555555580e <+23>:    cmp    $0x1,%edi
   0x0000555555555811 <+26>:    je     0x55555555582c <func4+53>
   0x0000555555555813 <+28>:    lea    -0x1(%rdi),%edi
   0x0000555555555816 <+31>:    callq  0x5555555557f7 <func4>
   0x000055555555581b <+36>:    lea    (%rax,%rbp,1),%r12d
   0x000055555555581f <+40>:    lea    -0x2(%rbx),%edi
   0x0000555555555822 <+43>:    mov    %ebp,%esi
   0x0000555555555824 <+45>:    callq  0x5555555557f7 <func4>
   0x0000555555555829 <+50>:    add    %r12d,%eax
   0x000055555555582c <+53>:    pop    %rbx
   0x000055555555582d <+54>:    pop    %rbp
   0x000055555555582e <+55>:    pop    %r12
   0x0000555555555830 <+57>:    retq
   0x0000555555555831 <+58>:    retq
End of assembler dump.
```

递归函数对应的代码

```cpp
#include <iostream>
using namespace std;
int func4(int a, int b) {
    if (a == 0)
       return 0;
    if (a == 1)
        return b;
    return func4(a - 1, 3) + func4(a - 2, 3) + 3;
}
int main() {
    int a = 8;
    int b = 3;
    int re = func4(a, b);
    cout << re;
}
```

运行程序得到结果



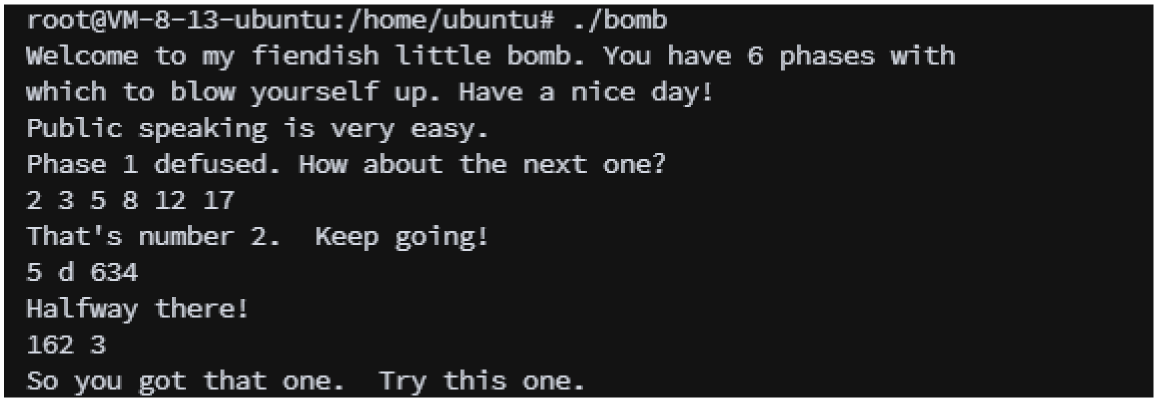
```
162
--------------------------------
Process exited after 0.571 seconds with return value 0
请按任意键继续. . .
```

从汇编语言

`cmp %eax,0x4(%rsp)`

可知第一个输入应该等于162

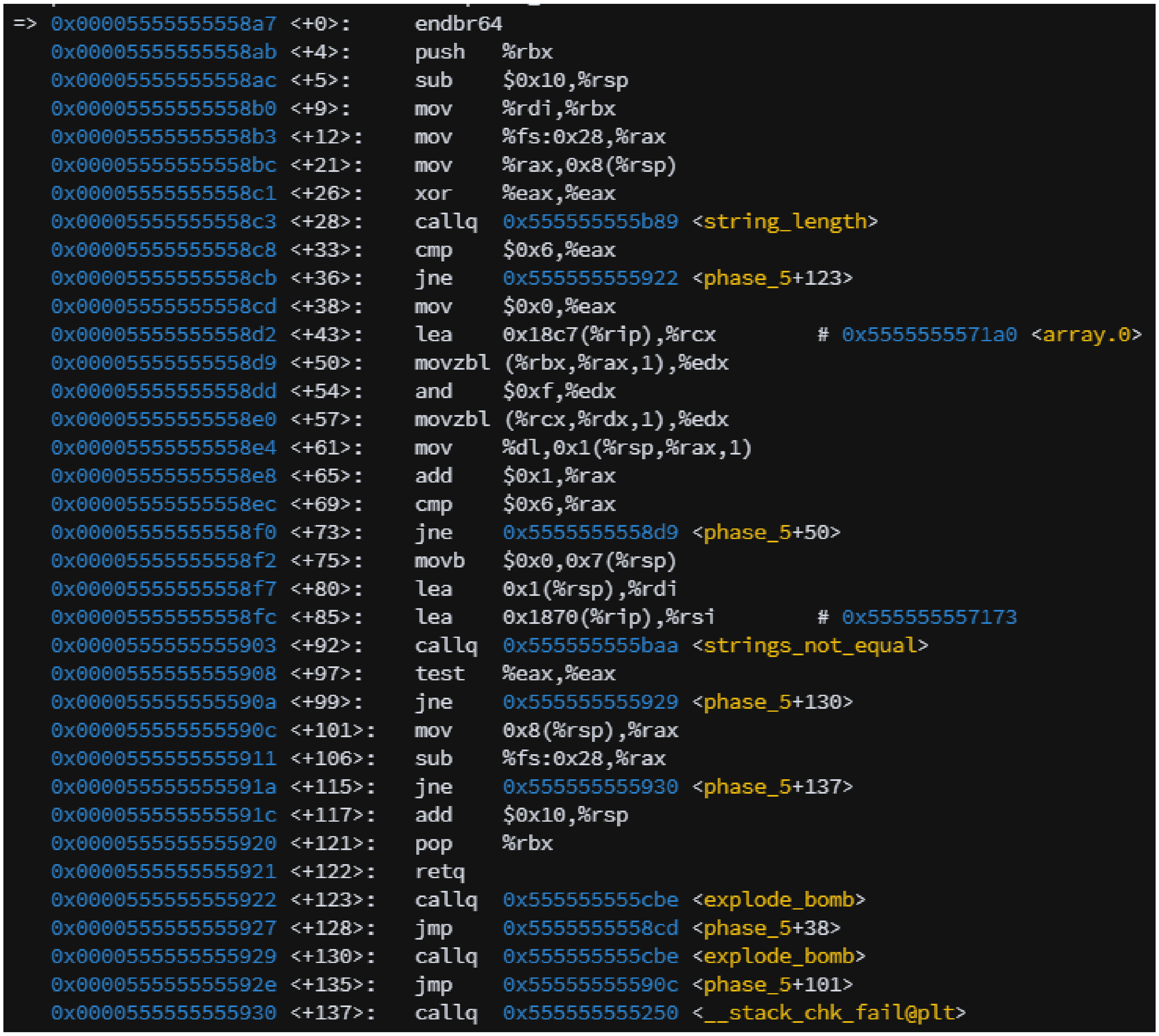
故本题的答案是162 3



```
root@VM-8-13-ubuntu:/home/ubuntu# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
2 3 5 8 12 17
That's number 2.  Keep going!
5 d 634
Halfway there!
162 3
So you got that one.  Try this one.
```

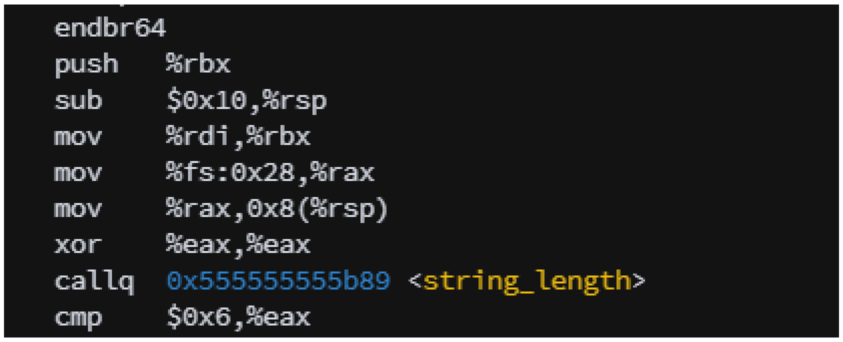
phase_5

汇编代码



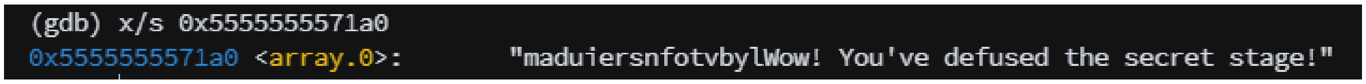
```
=> 0x00005555555558a7 <+0>:     endbr64
   0x00005555555558ab <+4>:     push   %rbx
   0x00005555555558ac <+5>:     sub    $0x10,%rsp
   0x00005555555558b0 <+9>:     mov    %rdi,%rbx
   0x00005555555558b3 <+12>:    mov    %fs:0x28,%rax
   0x00005555555558bc <+21>:    mov    %rax,0x8(%rsp)
   0x00005555555558c1 <+26>:    xor    %eax,%eax
   0x00005555555558c3 <+28>:    callq  0x555555555b89 <string_length>
   0x00005555555558c8 <+33>:    cmp    $0x6,%eax
   0x00005555555558cb <+36>:    jne    0x555555555922 <phase_5+123>
   0x00005555555558cd <+38>:    mov    $0x0,%eax
   0x00005555555558d2 <+43>:    lea    0x18c7(%rip),%rcx        # 0x5555555571a0 <array.0>
   0x00005555555558d9 <+50>:    movzbl (%rbx,%rax,1),%edx
   0x00005555555558dd <+54>:    and    $0xf,%edx
   0x00005555555558e0 <+57>:    movzbl (%rcx,%rdx,1),%edx
   0x00005555555558e4 <+61>:    mov    %dl,0x1(%rsp,%rax,1)
   0x00005555555558e8 <+65>:    add    $0x1,%rax
   0x00005555555558ec <+69>:    cmp    $0x6,%rax
   0x00005555555558f0 <+73>:    jne    0x5555555558d9 <phase_5+50>
   0x00005555555558f2 <+75>:    movb   $0x0,0x7(%rsp)
   0x00005555555558f7 <+80>:    lea    0x1(%rsp),%rdi
   0x00005555555558fc <+85>:    lea    0x1870(%rip),%rsi        # 0x555555557173
   0x0000555555555903 <+92>:    callq  0x555555555baa <strings_not_equal>
   0x0000555555555908 <+97>:    test   %eax,%eax
   0x000055555555590a <+99>:    jne    0x555555555929 <phase_5+130>
   0x000055555555590c <+101>:   mov    0x8(%rsp),%rax
   0x0000555555555911 <+106>:   sub    %fs:0x28,%rax
   0x000055555555591a <+115>:   jne    0x555555555930 <phase_5+137>
   0x000055555555591c <+117>:   add    $0x10,%rsp
   0x0000555555555920 <+121>:   pop    %rbx
   0x0000555555555921 <+122>:   retq
   0x0000555555555922 <+123>:   callq  0x555555555cbe <explode_bomb>
   0x0000555555555927 <+128>:   jmp    0x5555555558cd <phase_5+38>
   0x0000555555555929 <+130>:   callq  0x555555555cbe <explode_bomb>
   0x000055555555592e <+135>:   jmp    0x55555555590c <phase_5+101>
   0x0000555555555930 <+137>:   callq  0x555555555250 <__stack_chk_fail@plt>
```

从此处我们可知要输入长度为6的字符串



```
endbr64
push   %rbx
sub    $0x10,%rsp
mov    %rdi,%rbx
mov    %fs:0x28,%rax
mov    %rax,0x8(%rsp)
xor    %eax,%eax
callq  0x555555555b89 <string_length>
cmp    $0x6,%eax
```

查看地址

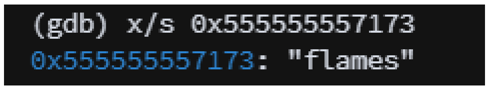


```
(gdb) x/s 0x5555555571a0
0x5555555571a0 <array.0>:       "maduiersnfotvbylWow! You've defused the secret stage!"
```

这段即为字符0-f对应的编码

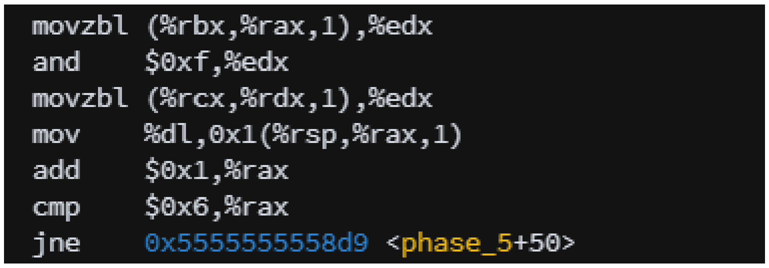
`maduiersnfotvbyl`

这是本题答案对应的字符



```
(gdb) x/s 0x555555557173
0x555555557173: "flames"
```

再根据这段循环



```
movzbl (%rbx,%rax,1),%edx
and    $0xf,%edx
movzbl (%rcx,%rdx,1),%edx
mov    %dl,0x1(%rsp,%rax,1)
add    $0x1,%rax
cmp    $0x6,%rax
jne    0x5555555558d9 <phase_5+50>
```

我们知道 `rax` 从0到5对应循环的次数

`edx = rax+rbx` 对应 rcx的偏移量 而**rax+rbx**则对应输入 `rbx` 的第几个字符

将字符的后四位保存下来 也即十六进制下的最后一个数字 保存到栈中 得到我们的偏移量

根据 " flame " 我们得出偏移量

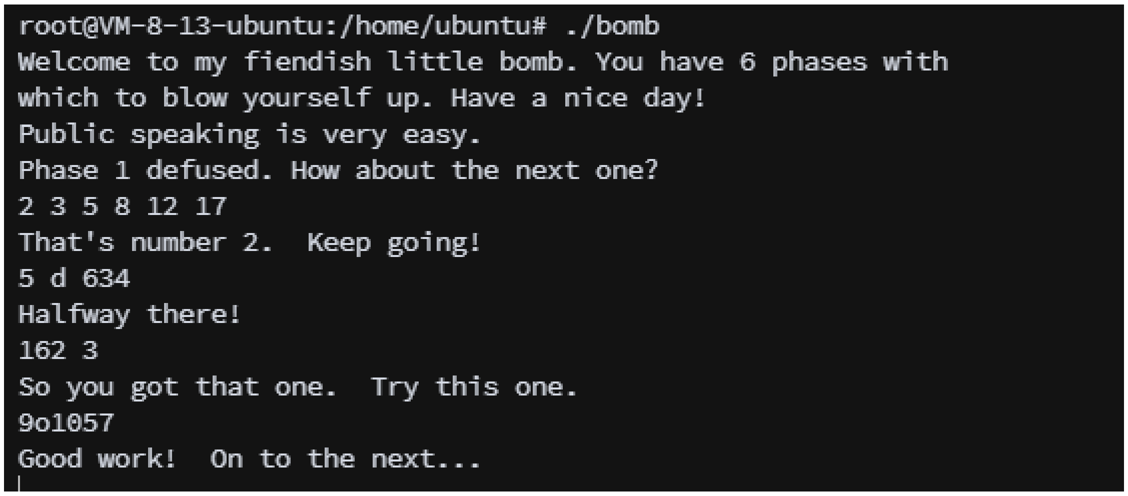
例如第一个f的偏移量为9

那么所有十六进制下最后一位为9的字符都符合条件

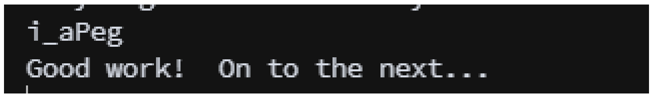
我们用其中一个即可

比如 `9o1057` `i_aPeg`

本题答案



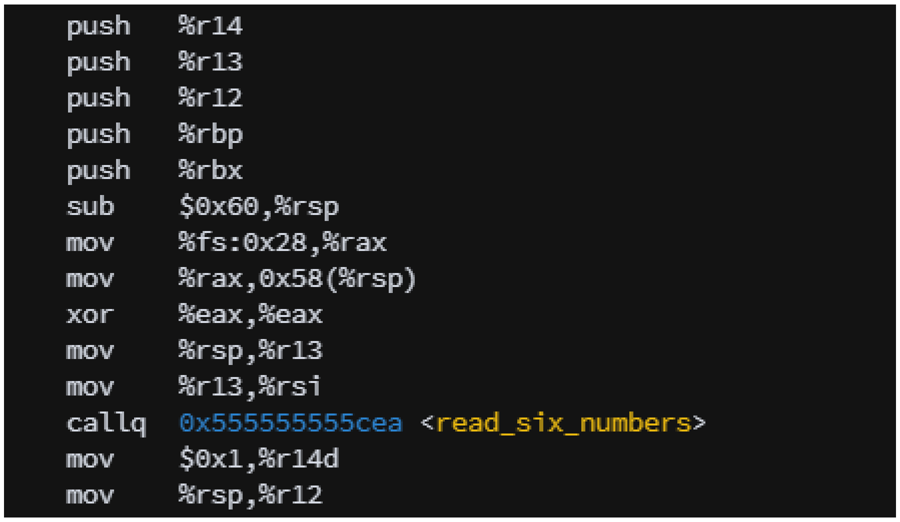
```
root@VM-8-13-ubuntu:/home/ubuntu# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
2 3 5 8 12 17
That's number 2.  Keep going!
5 d 634
Halfway there!
162 3
So you got that one.  Try this one.
9o1057
Good work!  On to the next...
```



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

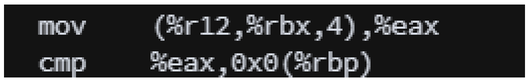
phase_6

首先读入6个数



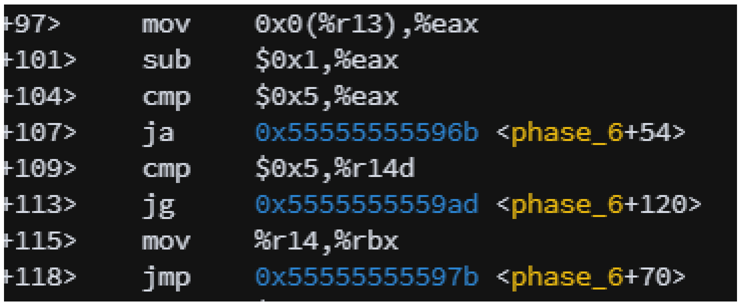
```
    push   %r14
    push   %r13
    push   %r12
    push   %rbp
    push   %rbx
    sub    $0x60,%rsp
    mov    %fs:0x28,%rax
    mov    %rax,0x58(%rsp)
    xor    %eax,%eax
    mov    %rsp,%r13
    mov    %r13,%rsi
    callq  0x555555555cea <read_six_numbers>
    mov    $0x1,%r14d
    mov    %rsp,%r12
```

输入的数中没有相等的数



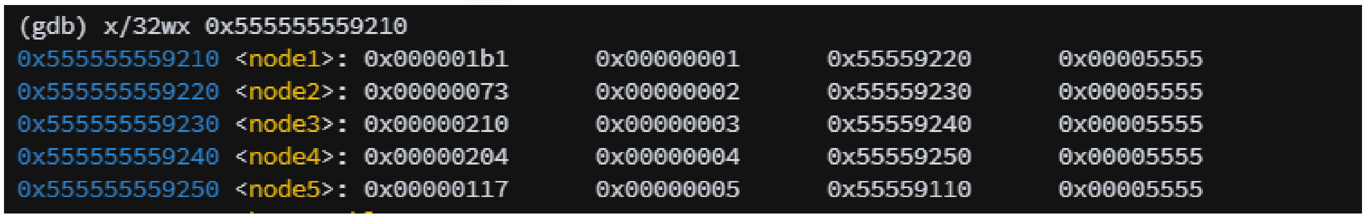
```
  mov     (%r12,%rbx,4),%eax
  cmp     %eax,0x0(%rbp)
```

输入的数大于0小于6

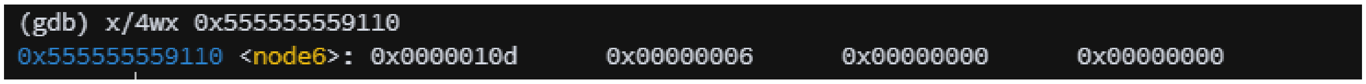


```
+97>      mov     0x0(%r13),%eax
+101>     sub     $0x1,%eax
+104>     cmp     $0x5,%eax
+107>     ja      0x55555555596b <phase_6+54>
+109>     cmp     $0x5,%r14d
+113>     jg      0x5555555559ad <phase_6+120>
+115>     mov     %r14,%rbx
+118>     jmp     0x55555555597b <phase_6+70>
```

我们查看链表的结点 发现前面有例如 `1b1` 的权值

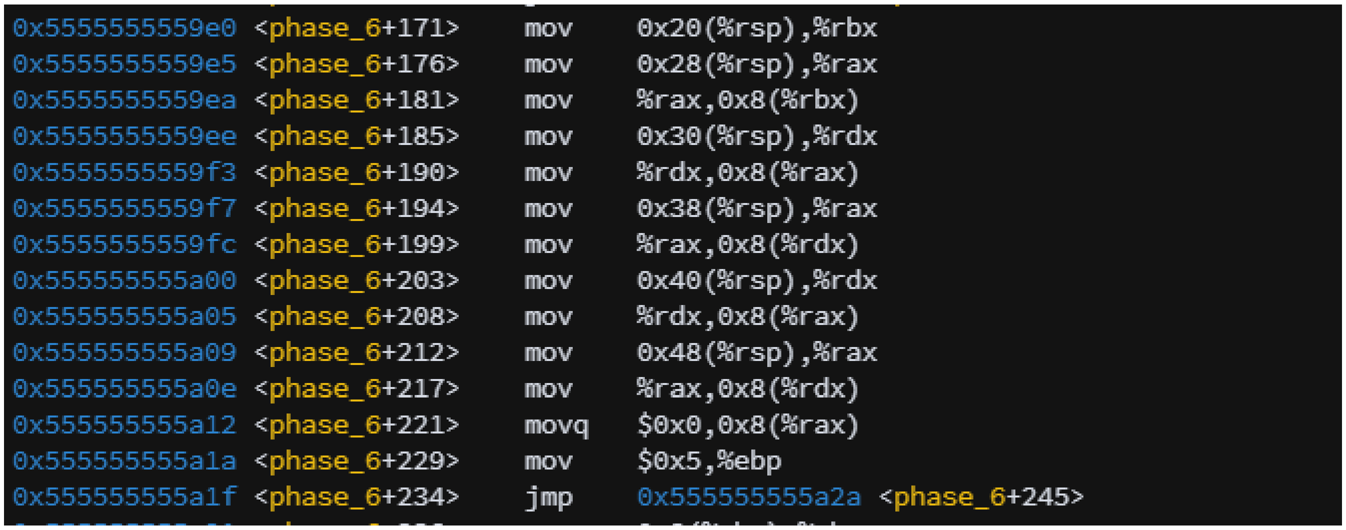


```
(gdb) x/32wx 0x555555559210
0x555555559210 <node1>: 0x000001b1      0x00000001      0x55559220      0x00005555
0x555555559220 <node2>: 0x00000073      0x00000002      0x55559230      0x00005555
0x555555559230 <node3>: 0x00000210      0x00000003      0x55559240      0x00005555
0x555555559240 <node4>: 0x00000204      0x00000004      0x55559250      0x00005555
0x555555559250 <node5>: 0x00000117      0x00000005      0x55559110      0x00005555
```



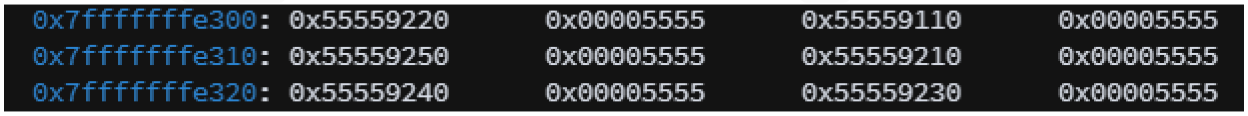
```
(gdb) x/4wx 0x555555559110
0x555555559110 <node6>: 0x0000010d      0x00000006      0x00000000      0x00000000
```

执行以修改跳转的地址



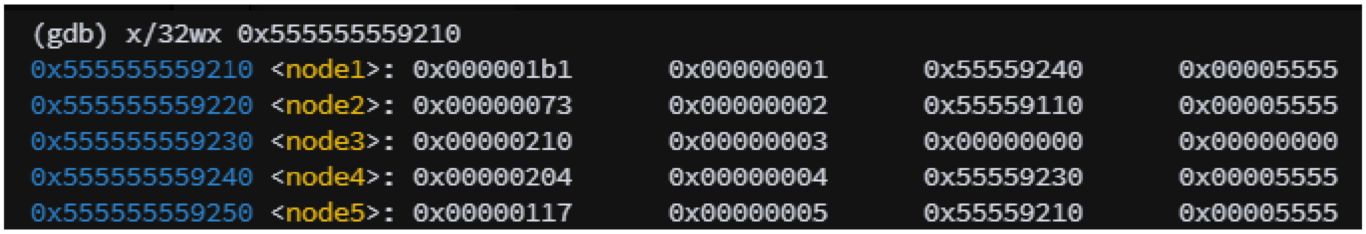
```
0x5555555559e0 <phase_6+171>    mov    0x20(%rsp),%rbx
0x5555555559e5 <phase_6+176>    mov    0x28(%rsp),%rax
0x5555555559ea <phase_6+181>    mov    %rax,0x8(%rbx)
0x5555555559ee <phase_6+185>    mov    0x30(%rsp),%rdx
0x5555555559f3 <phase_6+190>    mov    %rdx,0x8(%rax)
0x5555555559f7 <phase_6+194>    mov    0x38(%rsp),%rax
0x5555555559fc <phase_6+199>    mov    %rax,0x8(%rdx)
0x555555555a00 <phase_6+203>    mov    0x40(%rsp),%rdx
0x555555555a05 <phase_6+208>    mov    %rdx,0x8(%rax)
0x555555555a09 <phase_6+212>    mov    0x48(%rsp),%rax
0x555555555a0e <phase_6+217>    mov    %rax,0x8(%rdx)
0x555555555a12 <phase_6+221>    movq   $0x0,0x8(%rax)
0x555555555a1a <phase_6+229>    mov    $0x5,%ebp
0x555555555a1f <phase_6+234>    jmp    0x555555555a2a <phase_6+245>
```

在这个循环里 `20(rsp)` 后存放的是按照输入顺序连续的跳转地址 `rsp` 的地址为0x7fffffffe2e0



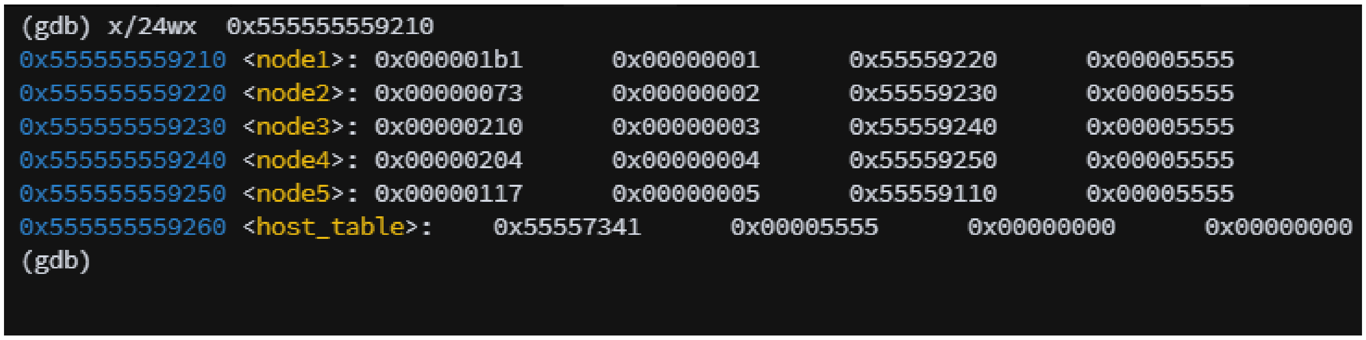
```
 0x7fffffffe300: 0x55559220      0x00005555      0x55559110      0x00005555
 0x7fffffffe310: 0x55559250      0x00005555      0x55559210      0x00005555
 0x7fffffffe320: 0x55559240      0x00005555      0x55559230      0x00005555
```

而 `rbx` 存放的是node的地址 比如0x555555559210 对应 `1` rbx+8对应这个结点的下一个结点的地址
我们只要修改这个地址便可以重构链表的顺序。



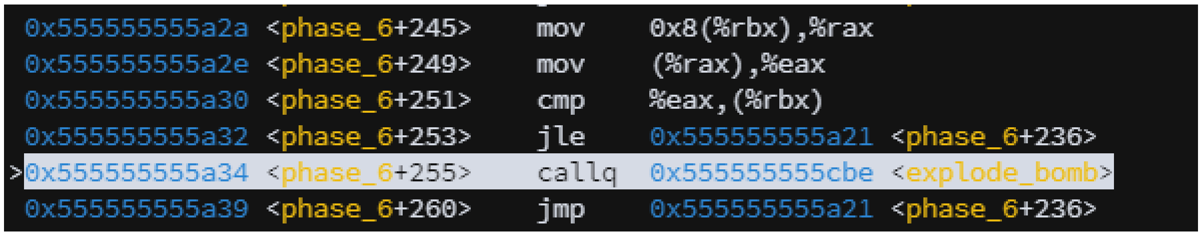
```
 (gdb) x/32wx 0x555555559210
 0x555555559210 <node1>: 0x000001b1      0x00000001      0x55559240      0x00005555
 0x555555559220 <node2>: 0x00000073      0x00000002      0x55559110      0x00005555
 0x555555559230 <node3>: 0x00000210      0x00000003      0x00000000      0x00000000
 0x555555559240 <node4>: 0x00000204      0x00000004      0x55559230      0x00005555
 0x555555559250 <node5>: 0x00000117      0x00000005      0x55559210      0x00005555
```

和原来的图对比很明显



```
(gdb) x/24wx  0x555555559210
0x555555559210 <node1>: 0x000001b1      0x00000001      0x55559220      0x00005555
0x555555559220 <node2>: 0x00000073      0x00000002      0x55559230      0x00005555
0x555555559230 <node3>: 0x00000210      0x00000003      0x55559240      0x00005555
0x555555559240 <node4>: 0x00000204      0x00000004      0x55559250      0x00005555
0x555555559250 <node5>: 0x00000117      0x00000005      0x55559110      0x00005555
0x555555559260 <host_table>:    0x55557341      0x00005555      0x00000000      0x00000000
(gdb)
```

这里eax是后一个权值 (rbx）是前一个权值，为了避免爆炸我们必须让整个链表的权值呈现升序

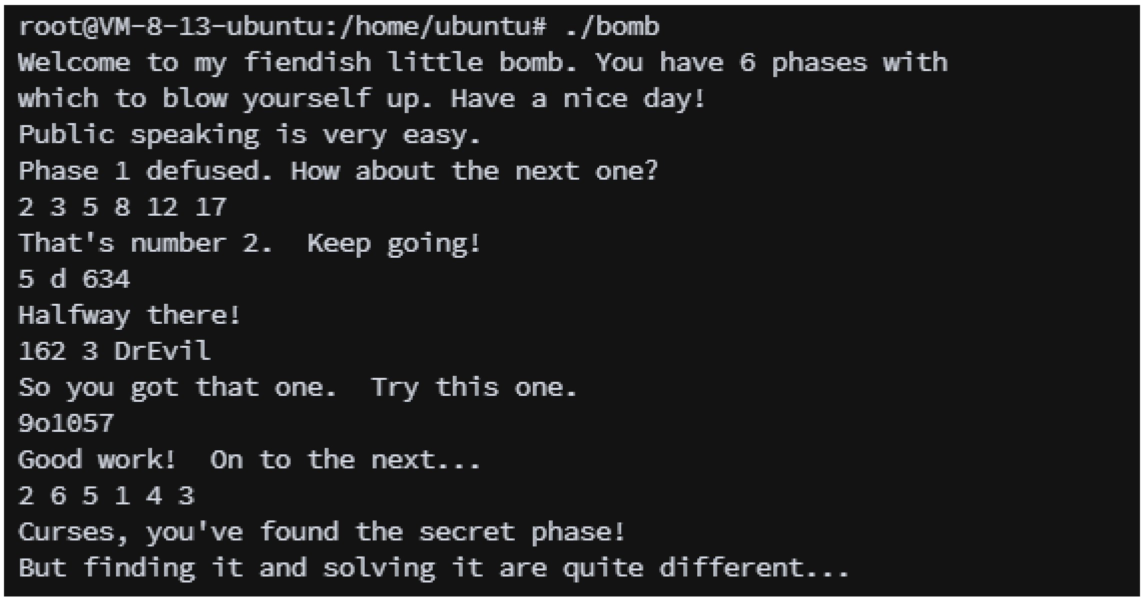


```
 0x555555555a2a <phase_6+245>    mov    0x8(%rbx),%rax
 0x555555555a2e <phase_6+249>    mov    (%rax),%eax
 0x555555555a30 <phase_6+251>    cmp    %eax,(%rbx)
 0x555555555a32 <phase_6+253>    jle    0x555555555a21 <phase_6+236>
>0x555555555a34 <phase_6+255>    callq  0x555555555cbe <explode_bomb>
 0x555555555a39 <phase_6+260>    jmp    0x555555555a21 <phase_6+236>
```

权值如下面的表

1	2	3	4	5	6
`1b1`	`73`	`210`	`204`	`117`	`10d`

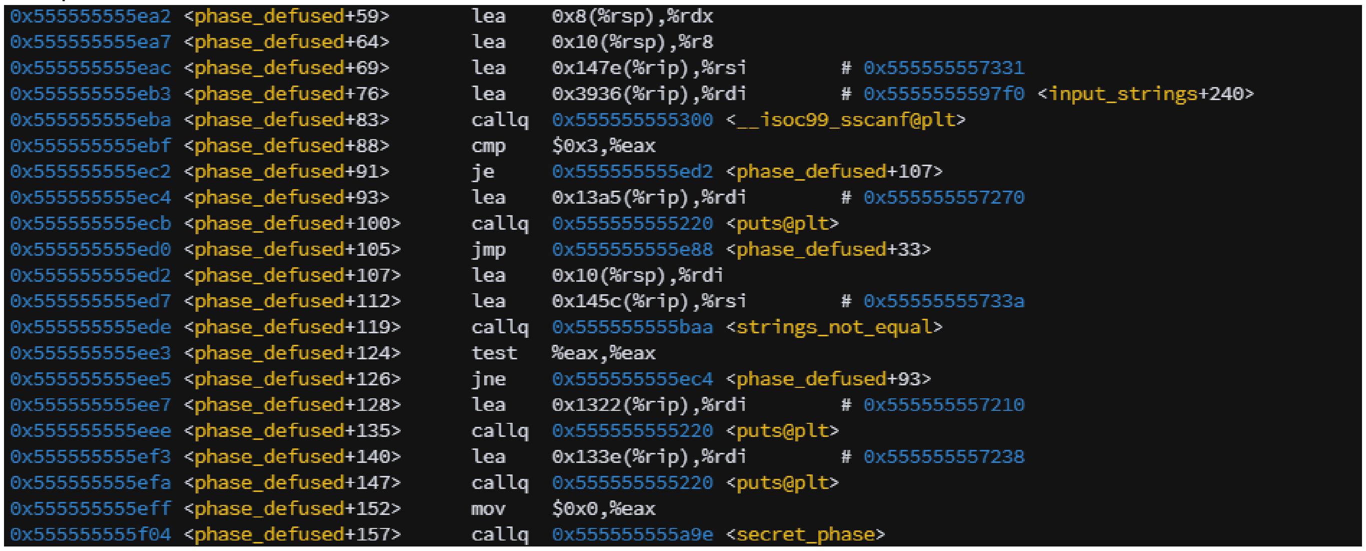
所以我们的正确顺序应该是
2 6 5 1 4 3



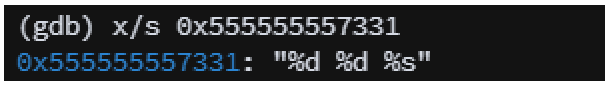
```
root@VM-8-13-ubuntu:/home/ubuntu# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
2 3 5 8 12 17
That's number 2.  Keep going!
5 d 634
Halfway there!
162 3 DrEvil
So you got that one.  Try this one.
9o1057
Good work!  On to the next...
2 6 5 1 4 3
Curses, you've found the secret phase!
But finding it and solving it are quite different...
```

secret_phase

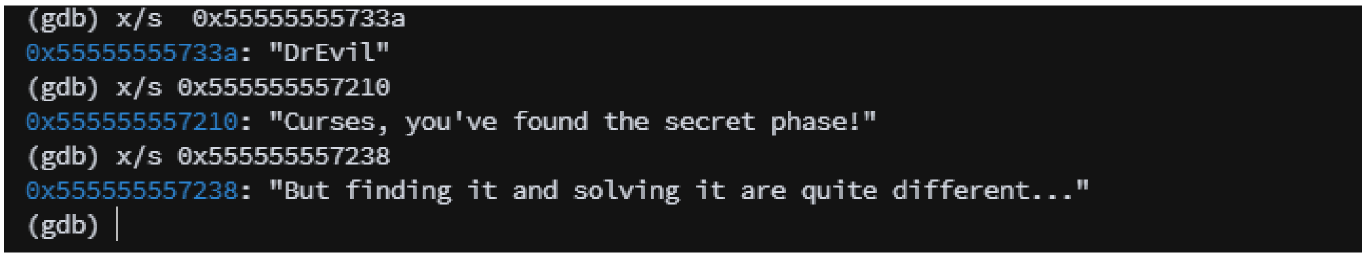
查看phase_defuse



```
0x555555555ea2 <phase_defused+59>       lea    0x8(%rsp),%rdx
0x555555555ea7 <phase_defused+64>       lea    0x10(%rsp),%r8
0x555555555eac <phase_defused+69>       lea    0x147e(%rip),%rsi        # 0x555555557331
0x555555555eb3 <phase_defused+76>       lea    0x3936(%rip),%rdi        # 0x5555555597f0 <input_strings+240>
0x555555555eba <phase_defused+83>       callq  0x555555555300 <__isoc99_sscanf@plt>
0x555555555ebf <phase_defused+88>       cmp    $0x3,%eax
0x555555555ec2 <phase_defused+91>       je     0x555555555ed2 <phase_defused+107>
0x555555555ec4 <phase_defused+93>       lea    0x13a5(%rip),%rdi        # 0x555555557270
0x555555555ecb <phase_defused+100>      callq  0x555555555220 <puts@plt>
0x555555555ed0 <phase_defused+105>      jmp    0x555555555e88 <phase_defused+33>
0x555555555ed2 <phase_defused+107>      lea    0x10(%rsp),%rdi
0x555555555ed7 <phase_defused+112>      lea    0x145c(%rip),%rsi        # 0x55555555733a
0x555555555ede <phase_defused+119>      callq  0x555555555baa <strings_not_equal>
0x555555555ee3 <phase_defused+124>      test   %eax,%eax
0x555555555ee5 <phase_defused+126>      jne    0x555555555ec4 <phase_defused+93>
0x555555555ee7 <phase_defused+128>      lea    0x1322(%rip),%rdi        # 0x555555557210
0x555555555eee <phase_defused+135>      callq  0x555555555220 <puts@plt>
0x555555555ef3 <phase_defused+140>      lea    0x133e(%rip),%rdi        # 0x555555557238
0x555555555efa <phase_defused+147>      callq  0x555555555220 <puts@plt>
0x555555555eff <phase_defused+152>      mov    $0x0,%eax
0x555555555f04 <phase_defused+157>      callq  0x555555555a9e <secret_phase>
```



```
(gdb) x/s 0x555555557331
0x555555557331: "%d %d %s"
```



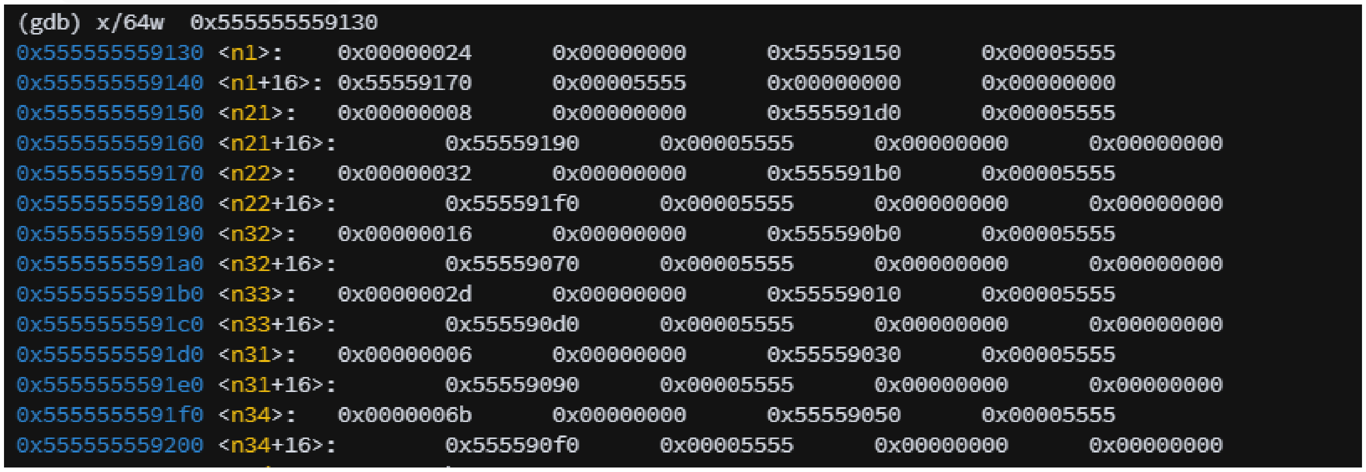
```
(gdb) x/s  0x55555555733a
0x55555555733a: "DrEvil"
(gdb) x/s 0x555555557210
0x555555557210: "Curses, you've found the secret phase!"
(gdb) x/s 0x555555557238
0x555555557238: "But finding it and solving it are quite different..."
(gdb) |
```

由这些信息我们可知当phase_4 多输入一个字符串 DeEvil便可进入Secret_phase

```
0x555555555a9e <secret_phase> endbr64

0x555555555aa2 <secret_phase+4> push %rbx
```

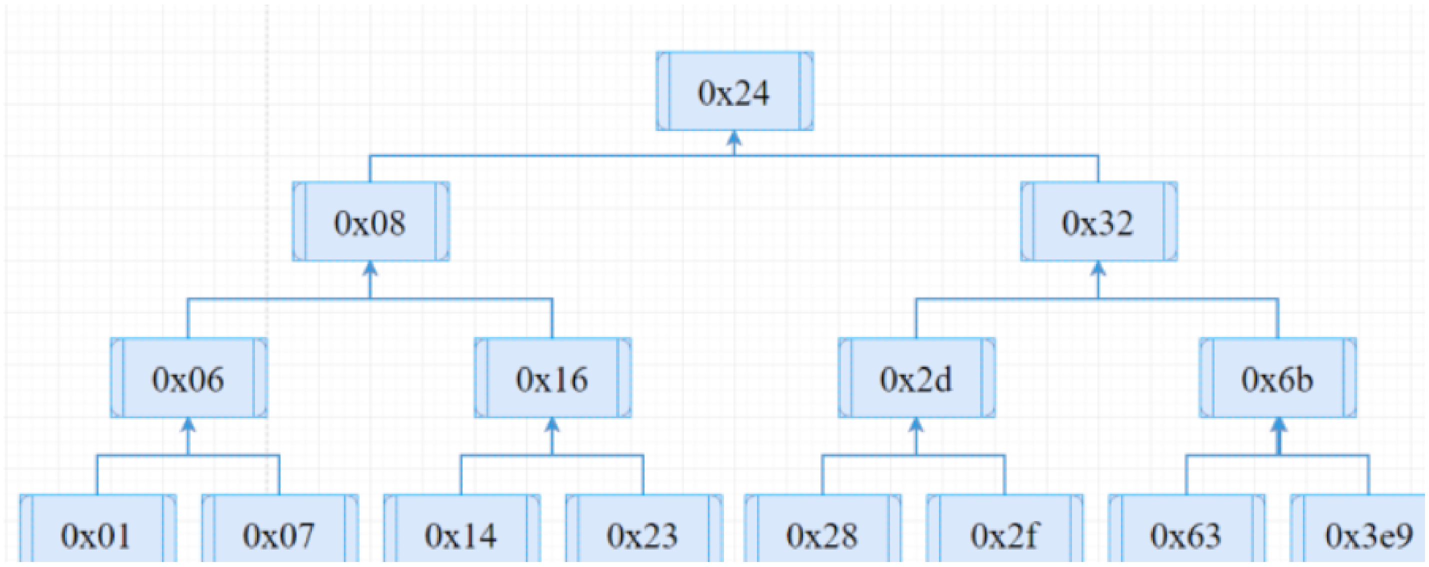
```
0x555555555aa3 <secret_phase+5> callq 0x555555555d2f <read_line>

0x555555555aa8 <secret_phase+10> mov %rax,%rdi

0x555555555aab <secret_phase+13> mov $0xa,%edx

0x555555555ab0 <secret_phase+18> mov $0x0,%esi

0x555555555ab5 <secret_phase+23> callq 0x5555555552e0 <strtol@plt>

0x555555555aba <secret_phase+28> mov %eax,%ebx

0x555555555abc <secret_phase+30> sub $0x1,%eax

0x555555555abf <secret_phase+33> cmp $0x3e8,%eax

0x555555555ac4 <secret_phase+38> ja 0x555555555aec <secret_phase+78>

0x555555555ac6 <secret_phase+40> mov %ebx,%esi

0x555555555ac8 <secret_phase+42> lea 0x3661(%rip),%rdi # 0x555555559130 <n1>

0x555555555acf <secret_phase+49> callq 0x555555555a5d <fun7>

0x555555555ad4 <secret_phase+54> cmp $0x1,%eax

0x555555555ad7 <secret_phase+57> jne 0x555555555af3 <secret_phase+85>

0x555555555ad9 <secret_phase+59> lea 0x16d0(%rip),%rdi # 0x5555555571b0

0x555555555ae0 <secret_phase+66> callq 0x555555555220 <puts@plt>

0x555555555ae5 <secret_phase+71> callq 0x555555555e67 <phase_defused>

0x555555555aea <secret_phase+76> pop %rbx

0x555555555aeb <secret_phase+77> retq

0x555555555aec <secret_phase+78> callq 0x555555555cbe <explode_bomb>

0x555555555af1 <secret_phase+83> jmp 0x555555555ac6 <secret_phase+40>

0x555555555af3 <secret_phase+85> callq 0x555555555cbe <explode_bomb>

0x555555555af8 <secret_phase+90> jmp 0x555555555ad9 <secret_phase+59>
```

查看 `0x555555559130 <n1>` 的地址我们会发现 建成了一颗二叉树

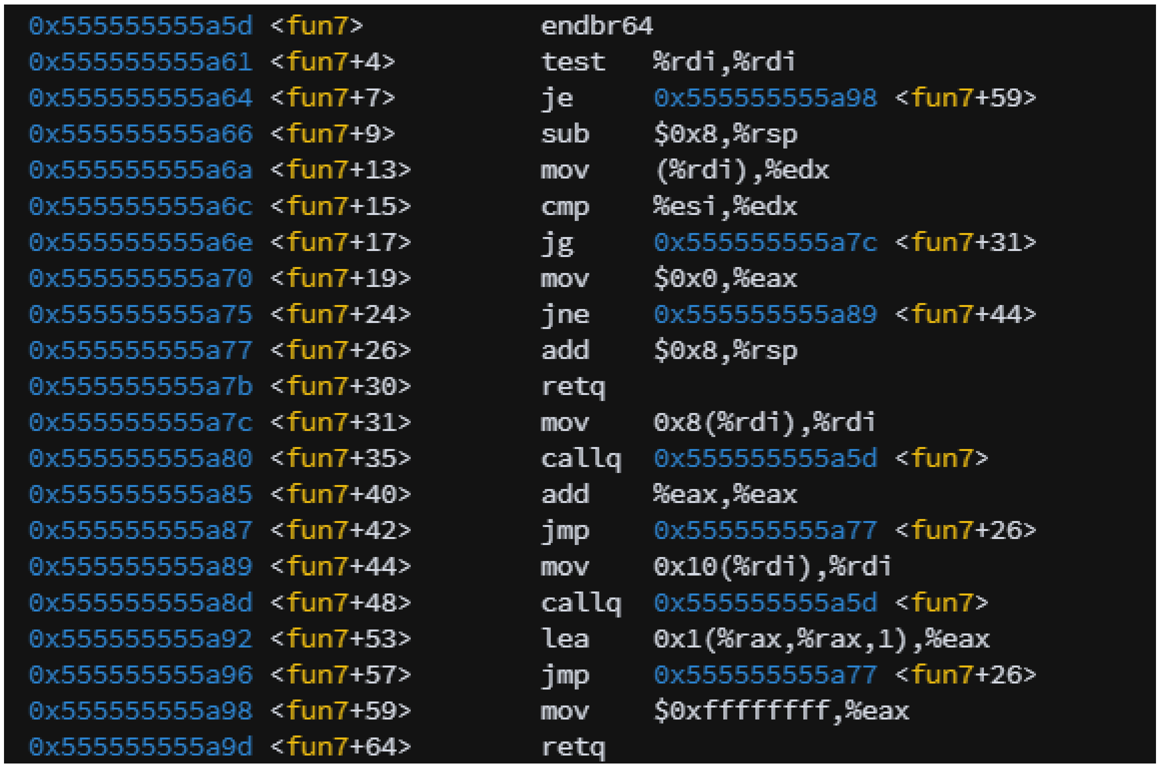


```
(gdb) x/64w  0x555555559130
0x555555559130 <n1>:    0x00000024      0x00000000      0x55559150      0x00005555
0x555555559140 <n1+16>: 0x55559170      0x00005555      0x00000000      0x00000000
0x555555559150 <n21>:   0x00000008      0x00000000      0x555591d0      0x00005555
0x555555559160 <n21+16>:        0x55559190      0x00005555      0x00000000      0x00000000
0x555555559170 <n22>:   0x00000032      0x00000000      0x555591b0      0x00005555
0x555555559180 <n22+16>:        0x555591f0      0x00005555      0x00000000      0x00000000
0x555555559190 <n32>:   0x00000016      0x00000000      0x555590b0      0x00005555
0x5555555591a0 <n32+16>:        0x55559070      0x00005555      0x00000000      0x00000000
0x5555555591b0 <n33>:   0x0000002d      0x00000000      0x55559010      0x00005555
0x5555555591c0 <n33+16>:        0x555590d0      0x00005555      0x00000000      0x00000000
0x5555555591d0 <n31>:   0x00000006      0x00000000      0x55559030      0x00005555
0x5555555591e0 <n31+16>:        0x55559090      0x00005555      0x00000000      0x00000000
0x5555555591f0 <n34>:   0x0000006b      0x00000000      0x55559050      0x00005555
0x555555559200 <n34+16>:        0x555590f0      0x00005555      0x00000000      0x00000000
```

二叉树的示意图如下

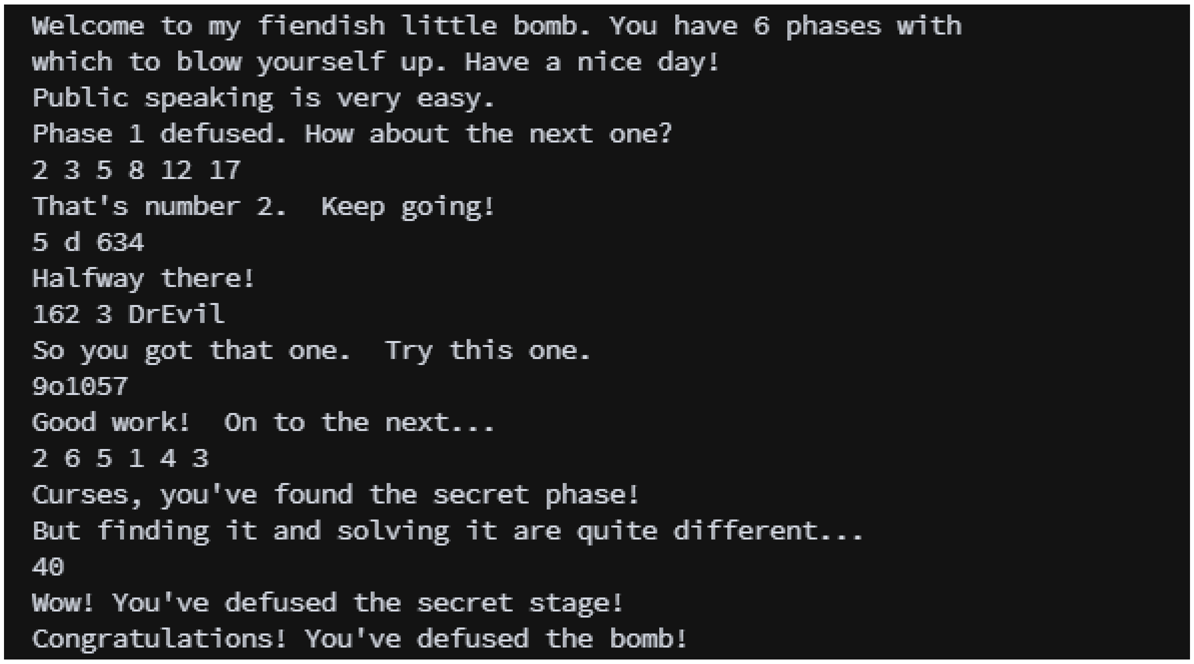


查看fun7函数 有下面的一行我们可以知道 当返回值为1是 程序便是正确的



```
0x555555555a5d <fun7>          endbr64
0x555555555a61 <fun7+4>        test   %rdi,%rdi
0x555555555a64 <fun7+7>        je     0x555555555a98 <fun7+59>
0x555555555a66 <fun7+9>        sub    $0x8,%rsp
0x555555555a6a <fun7+13>       mov    (%rdi),%edx
0x555555555a6c <fun7+15>       cmp    %esi,%edx
0x555555555a6e <fun7+17>       jg     0x555555555a7c <fun7+31>
0x555555555a70 <fun7+19>       mov    $0x0,%eax
0x555555555a75 <fun7+24>       jne    0x555555555a89 <fun7+44>
0x555555555a77 <fun7+26>       add    $0x8,%rsp
0x555555555a7b <fun7+30>       retq
0x555555555a7c <fun7+31>       mov    0x8(%rdi),%rdi
0x555555555a80 <fun7+35>       callq  0x555555555a5d <fun7>
0x555555555a85 <fun7+40>       add    %eax,%eax
0x555555555a87 <fun7+42>       jmp    0x555555555a77 <fun7+26>
0x555555555a89 <fun7+44>       mov    0x10(%rdi),%rdi
0x555555555a8d <fun7+48>       callq  0x555555555a5d <fun7>
0x555555555a92 <fun7+53>       lea    0x1(%rax,%rax,1),%eax
0x555555555a96 <fun7+57>       jmp    0x555555555a77 <fun7+26>
0x555555555a98 <fun7+59>       mov    $0xffffffff,%eax
0x555555555a9d <fun7+64>       retq
```

当这个结点的数（rdx）大于输入的数（esi) 则会跳转到 <fun+31> eax *2*
如果小于 eax=2 eax+1
如果等于 eax=0;
如若要使eax=1 那我们输入的数应该为0x28 也就是 40



```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
2 3 5 8 12 17
That's number 2.  Keep going!
5 d 634
Halfway there!
162 3 DrEvil
So you got that one.  Try this one.
9o1057
Good work!  On to the next...
2 6 5 1 4 3
Curses, you've found the secret phase!
But finding it and solving it are quite different...
40
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```

本题完整的答案为

```
Public speaking is very easy.
2 3 5 8 12 17
5 d 634
162 3 DrEvil
9o1057
```

```
2 6 5 1 4 3
40
```