中国海洋大学 计算机科学与技术系

实验报告

姓名：邓燚　　**年级**：2022　　**专业**：工程管理

科目：计算机系统原理　　**题目**：Bomb Lab

实验时间：2023/12/15

实验成绩：　　**实验教师**：宿浩

一、实验目的：

1. 学习并熟练使用 `gdb` 调试器和 `objdump`
2. 理解汇编语言代码的行为或作用
3. 提高阅读和理解汇编代码的能力

二、实验要求：

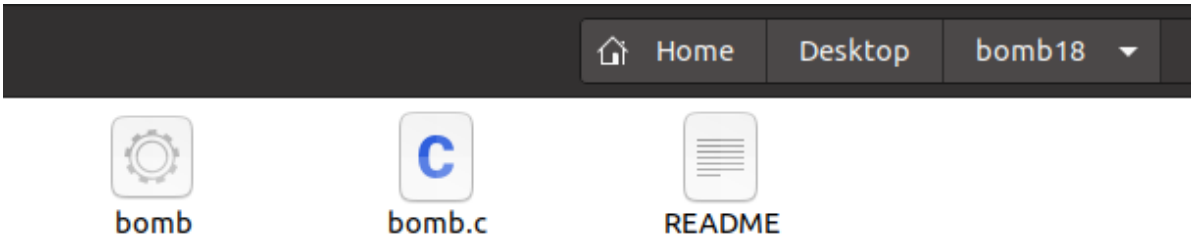
实验共包括七个阶段，每个阶段考察机器级语言程序的不同方面，难度递增

- **阶段一**：字符串比较
- **阶段二**：循环
- **阶段三**：条件/分支，含switch语句
- **阶段四**：递归调用和栈
- **阶段五**：指针
- **阶段六**：链表/指针/结构
- **隐藏阶段**：阶段四之后附加特定字符串后出现

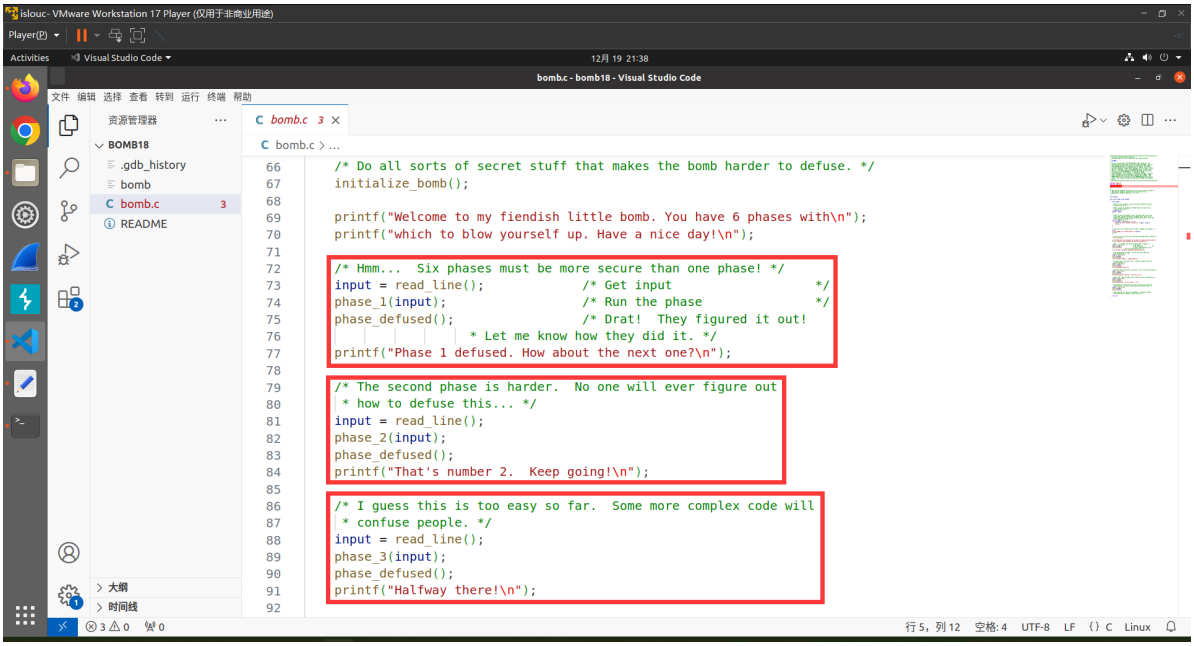
三、实验内容：

· 拆弹前准备

打开解压好的 `bomb18` 文件夹，其中有我们本次实验的目标文件 `bomb` 和 `bomb.c` 源文件



用 `Visual Studio Code` 打开之后发现



主函数 `main` 下有六个主要部分构成，分别对应着六个阶段（phase），

每个部分都由

输入函数

```
input = read_line();
```

炸弹函数

```
phase_x(input);
```

拆除成功函数

```
phase_defused();
```

提示语

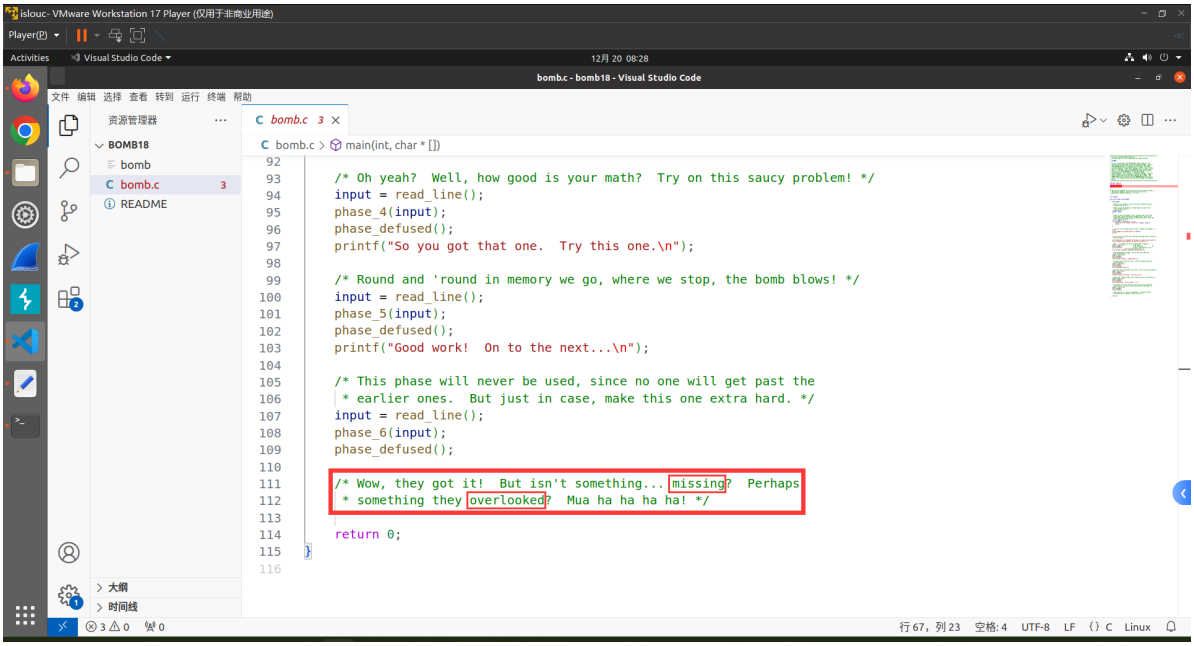
```
printf("xxxxxxxxxx");
```

显然我们应该关注的是炸弹函数phase_x（x=1,2,...,6）

但是我们注意到最后的注释部分：

```
/* Wow, they got it!  But isn't something... missing?  Perhaps

 * something they overlooked?  Mua ha ha ha ha! */
```

指的应该就是隐藏关卡

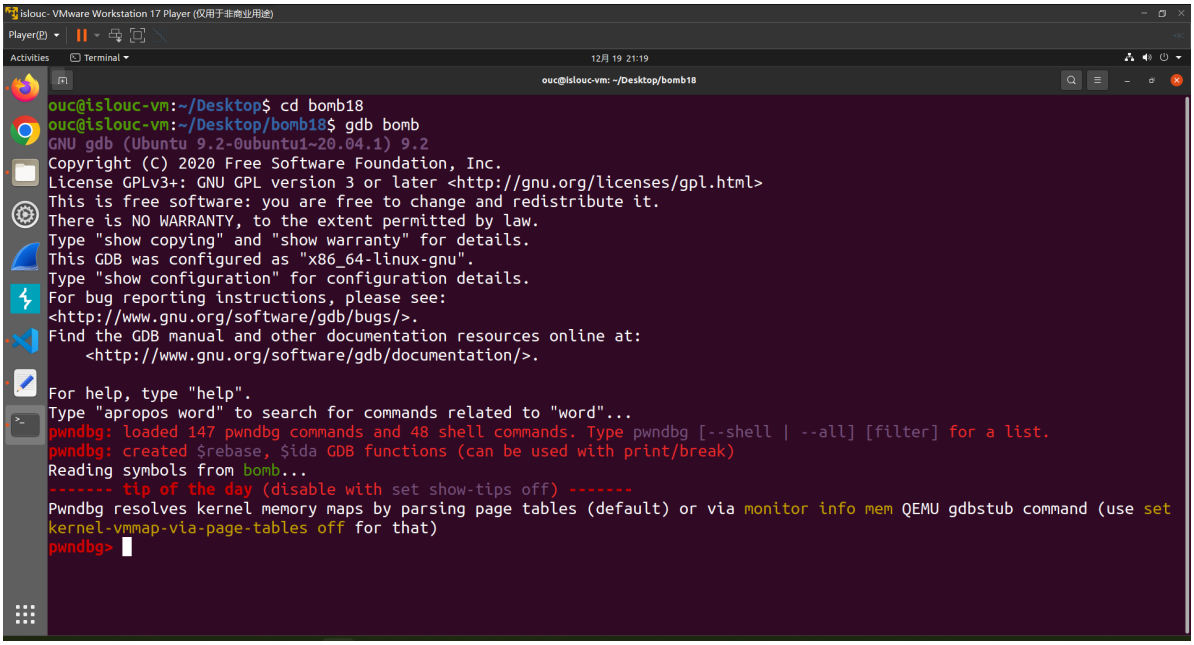


最后了解完结构就可以开始拆弹了~

· 启动GDB

打开终端，`cd` 至 `bomb` 目录下，并输入下面命令启动GDB

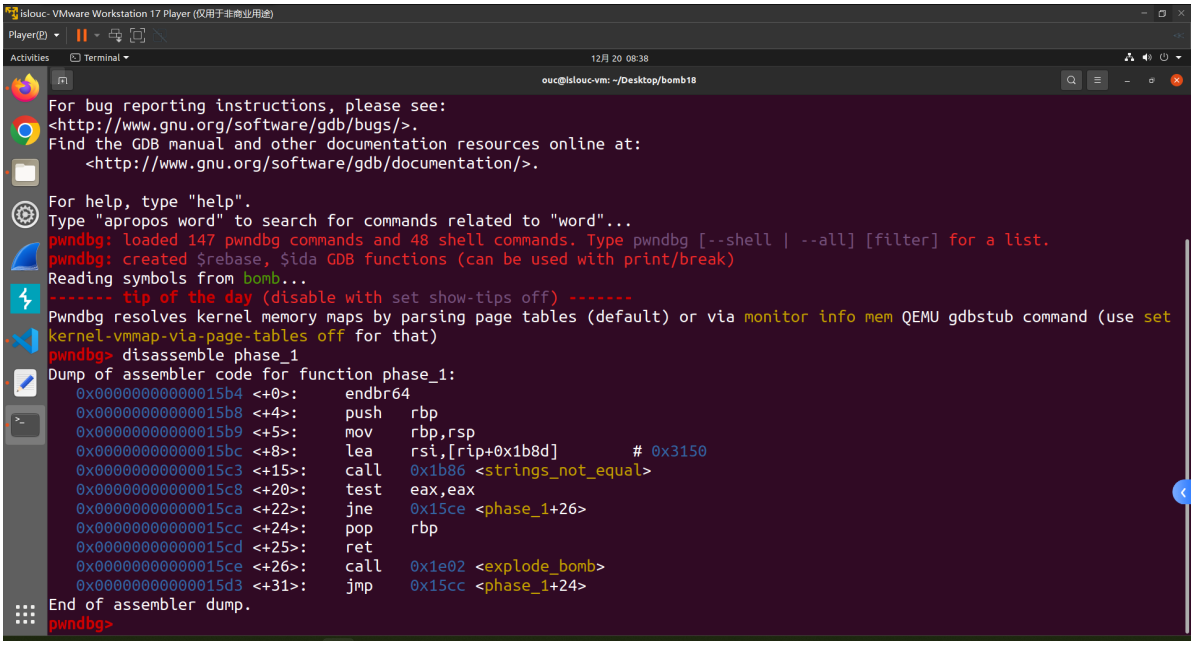
```
gdb bomb
```



· phase_1

先进行反汇编

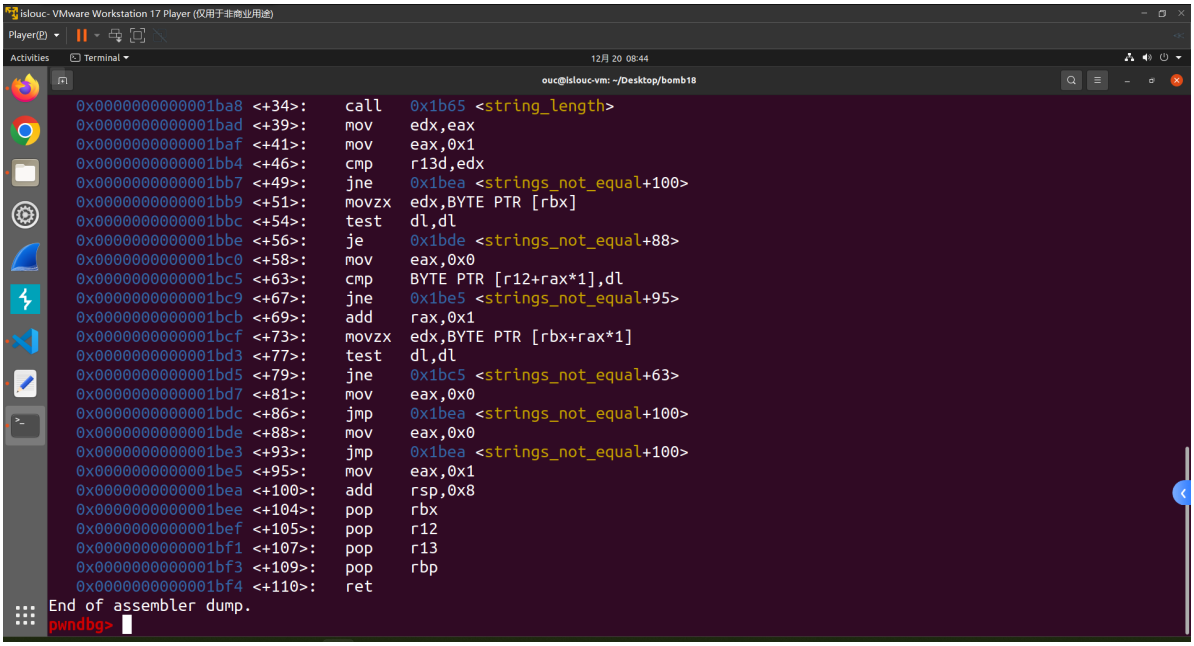
```
(gdb) disassemble phase_1
```




```
Dump of assembler code for function phase_1
   0x00000000000015b4 <+0>:     endbr64
   0x00000000000015b8 <+4>:     push   rbp
   0x00000000000015b9 <+5>:     mov    rbp,rsp
   0x00000000000015bc <+8>:     lea    rsi,[rip+0x1b8d]        # 0x3150
   0x00000000000015c3 <+15>:    call   0x1b86 <strings_not_equal>
   0x00000000000015c8 <+20>:    test   eax,eax
   0x00000000000015ca <+22>:    jne    0x15ce <phase_1+26>
   0x00000000000015cc <+24>:    pop    rbp
   0x00000000000015cd <+25>:    ret
   0x00000000000015ce <+26>:    call   0x1e02 <explode_bomb>
   0x00000000000015d3 <+31>:    jmp    0x15cc <phase_1+24>
End of assembler dump.
```

显然里面调用了一个 `strings_not_equal` 函数，我们把它也反汇编出来

```
(gdb) disassemble strings_not_equal
```




```
Dump of assembler code for function strings_not_equal:
   0x0000000000001b86 <+0>:     endbr64
```

```
   0x0000000000001b8a <+4>:     push   rbp
   0x0000000000001b8b <+5>:     mov    rbp,rsp
   0x0000000000001b8e <+8>:     push   r13
   0x0000000000001b90 <+10>:    push   r12
   0x0000000000001b92 <+12>:    push   rbx
   0x0000000000001b93 <+13>:    sub    rsp,0x8
   0x0000000000001b97 <+17>:    mov    rbx,rdi
   0x0000000000001b9a <+20>:    mov    r12,rsi
   0x0000000000001b9d <+23>:    call   0x1b65 <string_length>
   0x0000000000001ba2 <+28>:    mov    r13d,eax
   0x0000000000001ba5 <+31>:    mov    rdi,r12
   0x0000000000001ba8 <+34>:    call   0x1b65 <string_length>
   0x0000000000001bad <+39>:    mov    edx,eax
   0x0000000000001baf <+41>:    mov    eax,0x1
   0x0000000000001bb4 <+46>:    cmp    r13d,edx
   0x0000000000001bb7 <+49>:    jne    0x1bea <strings_not_equal+100>
   0x0000000000001bb9 <+51>:    movzx  edx,BYTE PTR [rbx]
   0x0000000000001bbc <+54>:    test   dl,dl
   0x0000000000001bbe <+56>:    je     0x1bde <strings_not_equal+88>
   0x0000000000001bc0 <+58>:    mov    eax,0x0
   0x0000000000001bc5 <+63>:    cmp    BYTE PTR [r12+rax*1],dl
   0x0000000000001bc9 <+67>:    jne    0x1be5 <strings_not_equal+95>
   0x0000000000001bcb <+69>:    add    rax,0x1
   0x0000000000001bcf <+73>:    movzx  edx,BYTE PTR [rbx+rax*1]
   0x0000000000001bd3 <+77>:    test   dl,dl
   0x0000000000001bd5 <+79>:    jne    0x1bc5 <strings_not_equal+63>
   0x0000000000001bd7 <+81>:    mov    eax,0x0
   0x0000000000001bdc <+86>:    jmp    0x1bea <strings_not_equal+100>
   0x0000000000001bde <+88>:    mov    eax,0x0
   0x0000000000001be3 <+93>:    jmp    0x1bea <strings_not_equal+100>
   0x0000000000001be5 <+95>:    mov    eax,0x1
   0x0000000000001bea <+100>:   add    rsp,0x8
   0x0000000000001bee <+104>:   pop    rbx
   0x0000000000001bef <+105>:   pop    r12
   0x0000000000001bf1 <+107>:   pop    r13
   0x0000000000001bf3 <+109>:   pop    rbp
   0x0000000000001bf4 <+110>:   ret
End of assembler dump.
```

同理，`string_length`函数

```
(gdb) disassemble string_length
```

```
Dump of assembler code for function string_length:
   0x0000000000001b65 <+0>:     endbr64
   0x0000000000001b69 <+4>:     cmp    BYTE PTR [rdi],0x0
   0x0000000000001b6c <+7>:     je     0x1b80 <string_length+27>
   0x0000000000001b6e <+9>:     mov    eax,0x0
   0x0000000000001b73 <+14>:    add    rdi,0x1
   0x0000000000001b77 <+18>:    add    eax,0x1
   0x0000000000001b7a <+21>:    cmp    BYTE PTR [rdi],0x0
   0x0000000000001b7d <+24>:    jne    0x1b73 <string_length+14>
   0x0000000000001b7f <+26>:    ret
   0x0000000000001b80 <+27>:    mov    eax,0x0
   0x0000000000001b85 <+32>:    ret
End of assembler dump.
```

至此，光是通过函数名推理我们也能得知需要输入一个字符串

分析一下发现，输入的字符串由 `strings_not_equal` 函数进入 `string_length` 函数，先判断字符串长度是否相等，相等则 `eax` 存0返回，不相等则 `eax` 存1返回

同时观察到 `phase_1` 中描述

```
0x00000000000015c8 <+20>:   test   eax,eax
0x00000000000015ca <+22>:   jne    0x15ce <phase_1+26>
0x00000000000015cc <+24>:   pop    rbp
0x00000000000015cd <+25>:   ret
0x00000000000015ce <+26>:   call   0x1e02 <explode_bomb>
```

当 `eax` != 0 时跳转到引爆函数，显然只有 `eax` == 0 且字符串相等时才能成功拆除

所以应该找到这个要比较的字符串，我们注意到 `phase_1` 中有个地址 `0x3150`

```
0x00000000000015bc <+8>:    lea    rsi,[rip+0x1b8d]        # 0x3150
```

查看这个内存地址中的值

```
(gdb) x/s 0x3150
```

```
pwndbg> x/s 0x3150
0x3150: "When a problem comes along, you must zip it!"
```

显然这个字符串就是第一关的答案

> `When a problem comes along, you must zip it!`

· phase_2

先为 `phase_2` 设置断点

```
(gdb) b phase_2
```

将第一关的答案先存在 `ans.txt` 中，用 `r` 命令运行

```
(gdb) r ans.txt
```

随便输入一个字符串后反汇编 `phase_2`

```
(gdb) disassemble phase_2
```

```
Dump of assembler code for function phase_2:
=> 0x00005555555555d5 <+0>:     endbr64
   0x00005555555555d9 <+4>:     push   rbp
   0x00005555555555da <+5>:     mov    rbp,rsp
   0x00005555555555dd <+8>:     push   r12
   0x00005555555555df <+10>:    push   rbx
   0x00005555555555e0 <+11>:    sub    rsp,0x20
   0x00005555555555e4 <+15>:    mov    rax,QWORD PTR fs:0x28
   0x00005555555555ed <+24>:    mov    QWORD PTR [rbp-0x18],rax
   0x00005555555555f1 <+28>:    xor    eax,eax
   0x00005555555555f3 <+30>:    lea    rsi,[rbp-0x30]
   0x00005555555555f7 <+34>:    call   0x555555555e42 <read_six_numbers>
   0x00005555555555fc <+39>:    cmp    DWORD PTR [rbp-0x30],0x0
```

```
   0x0000555555555600 <+43>:    js     0x55555555560d <phase_2+56>
   0x0000555555555602 <+45>:    lea    r12,[rbp-0x30]
   0x0000555555555606 <+49>:    mov    ebx,0x1
   0x000055555555560b <+54>:    jmp    0x555555555625 <phase_2+80>
   0x000055555555560d <+56>:    call   0x555555555e02 <explode_bomb>
   0x0000555555555612 <+61>:    jmp    0x555555555602 <phase_2+45>
   0x0000555555555614 <+63>:    call   0x555555555e02 <explode_bomb>
   0x0000555555555619 <+68>:    add    ebx,0x1
   0x000055555555561c <+71>:    add    r12,0x4
   0x0000555555555620 <+75>:    cmp    ebx,0x6
   0x0000555555555623 <+78>:    je     0x555555555634 <phase_2+95>
   0x0000555555555625 <+80>:    mov    eax,ebx
   0x0000555555555627 <+82>:    add    eax,DWORD PTR [r12]
   0x000055555555562b <+86>:    cmp    DWORD PTR [r12+0x4],eax
   0x0000555555555630 <+91>:    je     0x555555555619 <phase_2+68>
   0x0000555555555632 <+93>:    jmp    0x555555555614 <phase_2+63>
   0x0000555555555634 <+95>:    mov    rax,QWORD PTR [rbp-0x18]
   0x0000555555555638 <+99>:    xor    rax,QWORD PTR fs:0x28
   0x0000555555555641 <+108>:   jne    0x55555555564c <phase_2+119>
   0x0000555555555643 <+110>:   add    rsp,0x20
   0x0000555555555647 <+114>:   pop    rbx
   0x0000555555555648 <+115>:   pop    r12
   0x000055555555564a <+117>:   pop    rbp
   0x000055555555564b <+118>:   ret
   0x000055555555564c <+119>:   call   0x555555555220 <__stack_chk_fail@plt>
End of assembler dump.
```

同理，反汇编出 `read_six_numbers` 函数

```
Dump of assembler code for function read_six_numbers:
   0x0000555555555e42 <+0>:     endbr64
   0x0000555555555e46 <+4>:     push   rbp
   0x0000555555555e47 <+5>:     mov    rbp,rsp
   0x0000555555555e4a <+8>:     mov    rdx,rsi
   0x0000555555555e4d <+11>:    lea    rcx,[rsi+0x4]
   0x0000555555555e51 <+15>:    lea    rax,[rsi+0x14]
   0x0000555555555e55 <+19>:    push   rax
   0x0000555555555e56 <+20>:    lea    rax,[rsi+0x10]
   0x0000555555555e5a <+24>:    push   rax
   0x0000555555555e5b <+25>:    lea    r9,[rsi+0xc]
   0x0000555555555e5f <+29>:    lea    r8,[rsi+0x8]
   0x0000555555555e63 <+33>:    lea    rsi,[rip+0x152f]        # 0x555555557399
   0x0000555555555e6a <+40>:    mov    eax,0x0
   0x0000555555555e6f <+45>:    call   0x5555555552c0 <__isoc99_sscanf@plt>
   0x0000555555555e74 <+50>:    add    rsp,0x10
   0x0000555555555e78 <+54>:    cmp    eax,0x5
   0x0000555555555e7b <+57>:    jle    0x555555555e7f <read_six_numbers+61>
   0x0000555555555e7d <+59>:    leave
   0x0000555555555e7e <+60>:    ret
   0x0000555555555e7f <+61>:    call   0x555555555e02 <explode_bomb>
End of assembler dump.
```

显然要求我们输入6个数字，查看 `read_six_numbers` 中的某个内存地址的值，果然应证了我们的猜想

```
0x0000555555555e63 <+33>:    lea    rsi,[rip+0x152f]        # 0x555555557399
```

```
(gdb) x/s 0x555555557399
```

```
pwndbg> x/s 0x555555557399
0x555555557399: "%d %d %d %d %d %d"
```

阅读分析程序不难发现，输入的第一个数要求不能为负数，且后一个数与前一个数的差依次为 `1`、`2`、`3`、`4`、`5`

```
0x00005555555555fc <+39>:   cmp    DWORD PTR [rbp-0x30],0x0
0x0000555555555600 <+43>:   js     0x55555555560d <phase_2+56>
0x000055555555560d <+56>:   call   0x555555555e02 <explode_bomb>
```

```
0x0000555555555627 <+82>:   add    eax,DWORD PTR [r12]
0x000055555555562b <+86>:   cmp    DWORD PTR [r12+0x4],eax
0x0000555555555630 <+91>:   je     0x555555555619 <phase_2+68>
```

显然有很多解，如：

> 1. `0 1 3 6 10 15`
> 2. `1 2 4 7 11 16`
> 3. `2 3 5 8 12 17`
> 4. ...

· phase_3

同理，反汇编 `phase_3` 函数

```
Dump of assembler code for function phase_3:
=> 0x0000555555555651 <+0>:     endbr64
   0x0000555555555655 <+4>:     push   rbp
   0x0000555555555656 <+5>:     mov    rbp,rsp
   0x0000555555555659 <+8>:     sub    rsp,0x20
   0x000055555555565d <+12>:    mov    rax,QWORD PTR fs:0x28
   0x0000555555555666 <+21>:    mov    QWORD PTR [rbp-0x8],rax
   0x000055555555566a <+25>:    xor    eax,eax
   0x000055555555566c <+27>:    lea    rcx,[rbp-0x11]
   0x0000555555555670 <+31>:    lea    rdx,[rbp-0x10]
   0x0000555555555674 <+35>:    lea    r8,[rbp-0xc]
   0x0000555555555678 <+39>:    lea    rsi,[rip+0x1b27]        # 0x5555555571a6
   0x000055555555567f <+46>:    call   0x5555555552c0 <__isoc99_sscanf@plt>
   0x0000555555555684 <+51>:    cmp    eax,0x2
   0x0000555555555687 <+54>:    jle    0x5555555556a7 <phase_3+86>
   0x0000555555555689 <+56>:    cmp    DWORD PTR [rbp-0x10],0x7
   0x000055555555568d <+60>:    ja     0x555555555790 <phase_3+319>
   0x0000555555555693 <+66>:    mov    eax,DWORD PTR [rbp-0x10]
   0x0000555555555696 <+69>:    lea    rdx,[rip+0x1b23]        # 0x5555555571c0
   0x000055555555569d <+76>:    movsxd rax,DWORD PTR [rdx+rax*4]
   0x00005555555556a1 <+80>:    add    rax,rdx
   0x00005555555556a4 <+83>:    notrack jmp rax
   0x00005555555556a7 <+86>:    call   0x555555555e02 <explode_bomb>
   0x00005555555556ac <+91>:    jmp    0x555555555689 <phase_3+56>
   0x00005555555556ae <+93>:    mov    eax,0x6a
   0x00005555555556b3 <+98>:    cmp    DWORD PTR [rbp-0xc],0x104
   0x00005555555556ba <+105>:   je     0x55555555579a <phase_3+329>
```

```
   0x00005555555556c0 <+111>:   call   0x555555555e02 <explode_bomb>
   0x00005555555556c5 <+116>:   mov    eax,0x6a
   0x00005555555556ca <+121>:   jmp    0x55555555579a <phase_3+329>
   0x00005555555556cf <+126>:   mov    eax,0x79
   0x00005555555556d4 <+131>:   cmp    DWORD PTR [rbp-0xc],0x59
   0x00005555555556d8 <+135>:   je     0x55555555579a <phase_3+329>
   0x00005555555556de <+141>:   call   0x555555555e02 <explode_bomb>
   0x00005555555556e3 <+146>:   mov    eax,0x79
   0x00005555555556e8 <+151>:   jmp    0x55555555579a <phase_3+329>
   0x00005555555556ed <+156>:   mov    eax,0x7a
   0x00005555555556f2 <+161>:   cmp    DWORD PTR [rbp-0xc],0x9a
   0x00005555555556f9 <+168>:   je     0x55555555579a <phase_3+329>
   0x00005555555556ff <+174>:   call   0x555555555e02 <explode_bomb>
   0x0000555555555704 <+179>:   mov    eax,0x7a
   0x0000555555555709 <+184>:   jmp    0x55555555579a <phase_3+329>
   0x000055555555570e <+189>:   mov    eax,0x69
   0x0000555555555713 <+194>:   cmp    DWORD PTR [rbp-0xc],0x230
   0x000055555555571a <+201>:   je     0x55555555579a <phase_3+329>
   0x000055555555571c <+203>:   call   0x555555555e02 <explode_bomb>
   0x0000555555555721 <+208>:   mov    eax,0x69
   0x0000555555555726 <+213>:   jmp    0x55555555579a <phase_3+329>
   0x0000555555555728 <+215>:   mov    eax,0x72
   0x000055555555572d <+220>:   cmp    DWORD PTR [rbp-0xc],0x398
   0x0000555555555734 <+227>:   je     0x55555555579a <phase_3+329>
   0x0000555555555736 <+229>:   call   0x555555555e02 <explode_bomb>
   0x000055555555573b <+234>:   mov    eax,0x72
   0x0000555555555740 <+239>:   jmp    0x55555555579a <phase_3+329>
   0x0000555555555742 <+241>:   mov    eax,0x71
   0x0000555555555747 <+246>:   cmp    DWORD PTR [rbp-0xc],0xe2
   0x000055555555574e <+253>:   je     0x55555555579a <phase_3+329>
   0x0000555555555750 <+255>:   call   0x555555555e02 <explode_bomb>
   0x0000555555555755 <+260>:   mov    eax,0x71
   0x000055555555575a <+265>:   jmp    0x55555555579a <phase_3+329>
   0x000055555555575c <+267>:   mov    eax,0x6f
   0x0000555555555761 <+272>:   cmp    DWORD PTR [rbp-0xc],0x207
   0x0000555555555768 <+279>:   je     0x55555555579a <phase_3+329>
   0x000055555555576a <+281>:   call   0x555555555e02 <explode_bomb>
   0x000055555555576f <+286>:   mov    eax,0x6f
   0x0000555555555774 <+291>:   jmp    0x55555555579a <phase_3+329>
   0x0000555555555776 <+293>:   mov    eax,0x77
   0x000055555555577b <+298>:   cmp    DWORD PTR [rbp-0xc],0xe1
   0x0000555555555782 <+305>:   je     0x55555555579a <phase_3+329>
   0x0000555555555784 <+307>:   call   0x555555555e02 <explode_bomb>
   0x0000555555555789 <+312>:   mov    eax,0x77
   0x000055555555578e <+317>:   jmp    0x55555555579a <phase_3+329>
   0x0000555555555790 <+319>:   call   0x555555555e02 <explode_bomb>
   0x0000555555555795 <+324>:   mov    eax,0x6c
   0x000055555555579a <+329>:   cmp    BYTE PTR [rbp-0x11],al
   0x000055555555579d <+332>:   jne    0x5555555557b0 <phase_3+351>
   0x000055555555579f <+334>:   mov    rax,QWORD PTR [rbp-0x8]
   0x00005555555557a3 <+338>:   xor    rax,QWORD PTR fs:0x28
   0x00005555555557ac <+347>:   jne    0x5555555557b7 <phase_3+358>
   0x00005555555557ae <+349>:   leave
   0x00005555555557af <+350>:   ret
   0x00005555555557b0 <+351>:   call   0x555555555e02 <explode_bomb>
   0x00005555555557b5 <+356>:   jmp    0x55555555579f <phase_3+334>
```

```
   0x00005555555557b7 <+358>:   call   0x555555555220 <__stack_chk_fail@plt>
End of assembler dump.
```

查看内存地址

```
0x0000555555555678 <+39>:   lea    rsi,[rip+0x1b27]        # 0x5555555571a6
```

```
(gdb) x/s 0x5555555571a6
```

```
pwndbg> x/s 0x5555555571a6
0x5555555571a6: "%d %c %d"
```

显然需要输入两个整数和一个字符

```
0x000055555555566c <+27>:   lea    rcx,[rbp-0x11]
0x0000555555555670 <+31>:   lea    rdx,[rbp-0x10]
0x0000555555555674 <+35>:   lea    r8,[rbp-0xc]
```

阅读程序知 `%d` `%c` `%d` 分别存到了 `[rbp-0x10]` `[rbp-0x11]` `[rbp-0xc]`

所以我们只需要从程序中找到这三个应该满足的条件

- **[rbp-0x10]**

  ```
  0x0000555555555689 <+56>:   cmp    DWORD PTR [rbp-0x10],0x7
  0x000055555555568d <+60>:   ja     0x555555555790 <phase_3+319>
  0x0000555555555790 <+319>:  call   0x555555555e02 <explode_bomb>
  ```

  第一个数字需满足小于等于 `7`

- **[rbp-0x11]**

  ```
  0x000055555555579a <+329>:  cmp    BYTE PTR [rbp-0x11],al
  0x000055555555579d <+332>:  jne    0x5555555557b0 <phase_3+351>
  0x00005555555557b0 <+351>:  call   0x555555555e02 <explode_bomb>
  ```

  字符的ASCII码需等于前面出现的 `eax`

- **[rbp-0xc]**

  ```
  0x00005555555556ae <+93>:   mov    eax,0x6a
  0x00005555555556b3 <+98>:   cmp    DWORD PTR [rbp-0xc],0x104
  0x00005555555556ba <+105>:  je     0x55555555579a <phase_3+329>
  0x00005555555556c0 <+111>:  call   0x555555555e02 <explode_bomb>
  0x00005555555556c5 <+116>:  mov    eax,0x6a
  0x00005555555556ca <+121>:  jmp    0x55555555579a <phase_3+329>

  0x00005555555556cf <+126>:  mov    eax,0x79
  0x00005555555556d4 <+131>:  cmp    DWORD PTR [rbp-0xc],0x59
  0x00005555555556d8 <+135>:  je     0x55555555579a <phase_3+329>
  0x00005555555556de <+141>:  call   0x555555555e02 <explode_bomb>
  0x00005555555556e3 <+146>:  mov    eax,0x79
  0x00005555555556e8 <+151>:  jmp    0x55555555579a <phase_3+329>

  0x00005555555556ed <+156>:  mov    eax,0x7a
  0x00005555555556f2 <+161>:  cmp    DWORD PTR [rbp-0xc],0x9a
  0x00005555555556f9 <+168>:  je     0x55555555579a <phase_3+329>
  ```

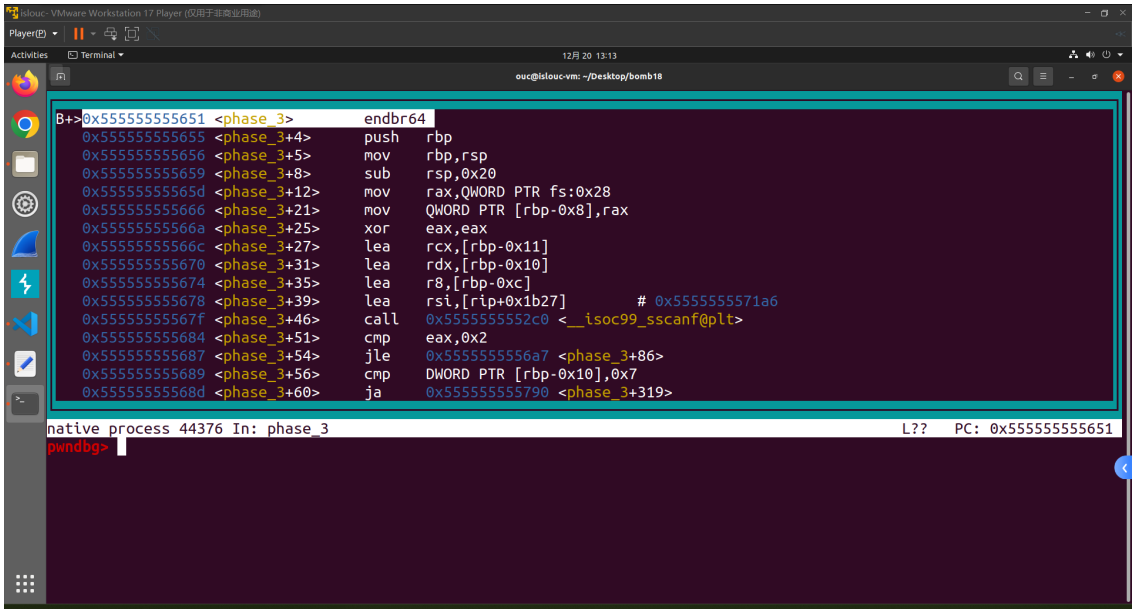
```
0x00005555555556ff <+174>:  call   0x555555555e02 <explode_bomb>
0x0000555555555704 <+179>:  mov    eax,0x7a
0x0000555555555709 <+184>:  jmp    0x55555555579a <phase_3+329>

0x000055555555570e <+189>:  mov    eax,0x69
0x0000555555555713 <+194>:  cmp    DWORD PTR [rbp-0xc],0x230
0x000055555555571a <+201>:  je     0x55555555579a <phase_3+329>
0x000055555555571c <+203>:  call   0x555555555e02 <explode_bomb>
0x0000555555555721 <+208>:  mov    eax,0x69
0x0000555555555726 <+213>:  jmp    0x55555555579a <phase_3+329>

0x0000555555555728 <+215>:  mov    eax,0x72
0x000055555555572d <+220>:  cmp    DWORD PTR [rbp-0xc],0x398
0x0000555555555734 <+227>:  je     0x55555555579a <phase_3+329>
0x0000555555555736 <+229>:  call   0x555555555e02 <explode_bomb>
0x000055555555573b <+234>:  mov    eax,0x72
0x0000555555555740 <+239>:  jmp    0x55555555579a <phase_3+329>

0x0000555555555742 <+241>:  mov    eax,0x71
0x0000555555555747 <+246>:  cmp    DWORD PTR [rbp-0xc],0xe2
0x000055555555574e <+253>:  je     0x55555555579a <phase_3+329>
0x0000555555555750 <+255>:  call   0x555555555e02 <explode_bomb>
0x0000555555555755 <+260>:  mov    eax,0x71
0x000055555555575a <+265>:  jmp    0x55555555579a <phase_3+329>

0x000055555555575c <+267>:  mov    eax,0x6f
0x0000555555555761 <+272>:  cmp    DWORD PTR [rbp-0xc],0x207
0x0000555555555768 <+279>:  je     0x55555555579a <phase_3+329>
0x000055555555576a <+281>:  call   0x555555555e02 <explode_bomb>
0x000055555555576f <+286>:  mov    eax,0x6f
0x0000555555555774 <+291>:  jmp    0x55555555579a <phase_3+329>

0x0000555555555776 <+293>:  mov    eax,0x77
0x000055555555577b <+298>:  cmp    DWORD PTR [rbp-0xc],0xe1
0x0000555555555782 <+305>:  je     0x55555555579a <phase_3+329>
0x0000555555555784 <+307>:  call   0x555555555e02 <explode_bomb>
0x0000555555555789 <+312>:  mov    eax,0x77
0x000055555555578e <+317>:  jmp    0x55555555579a <phase_3+329>

0x0000555555555790 <+319>:  call   0x555555555e02 <explode_bomb>
0x000055555555579a <+329>:  cmp    BYTE PTR [rbp-0x11],al
```

这里共有8种分支，分别对应着 `0` ~ `7` ，让我们分别输入 `0` ~ `7` 进行调试

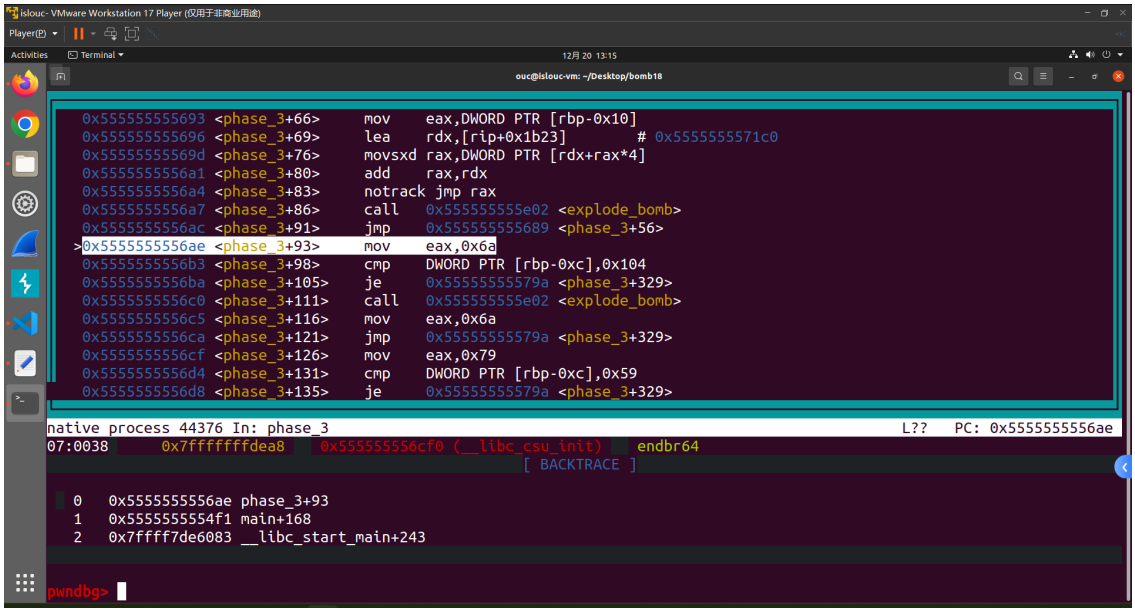
设置好断点后，依次输入 `0` `1` `2` `3` `4` `5` `6` `7` 作为第一个数字进行测试

显示反汇编窗口

```
(gdb) layout asm
```



通过 `next` 命令不断按顺序执行每一行的操作，发现最终跳转到了……




```
0x5555555556ae <phase_3+93>     mov     eax,0x6a
0x5555555556b3 <phase_3+98>     cmp     DWORD PTR [rbp-0xc],0x104
```

显然， `eax` 对应 `0x6a` ， `[rbp-0xc]` 对应 `0x104`

而 `0x6a` 转换成十进制 `106` ，其作为ASCII码对应的字符为 `j`

`0x104` 对应的数字则为 `260`

同理可得其他 `1` ~ `7` 的情况对应的分支

最后 `Ctrl` + `X` + `A` 退出反编译窗口

故答案有这8种情况：

```
0 j 260
1 y 89
2 z 154
3 i 560
4 r 920
```

```
5 q 226
6 o 519
7 w 225
```

· phase_4

反汇编 `phase_4` 函数

```
Dump of assembler code for function phase_4:
   0x00000000000017fa <+0>:     endbr64
   0x00000000000017fe <+4>:     push   rbp
   0x00000000000017ff <+5>:     mov    rbp,rsp
   0x0000000000001802 <+8>:     sub    rsp,0x10
   0x0000000000001806 <+12>:    mov    rax,QWORD PTR fs:0x28
   0x000000000000180f <+21>:    mov    QWORD PTR [rbp-0x8],rax
   0x0000000000001813 <+25>:    xor    eax,eax
   0x0000000000001815 <+27>:    lea    rcx,[rbp-0xc]
   0x0000000000001819 <+31>:    lea    rdx,[rbp-0x10]
   0x000000000000181d <+35>:    lea    rsi,[rip+0x1b81]        # 0x33a5
   0x0000000000001824 <+42>:    call   0x12c0 <__isoc99_sscanf@plt>
   0x0000000000001829 <+47>:    cmp    eax,0x2
   0x000000000000182c <+50>:    jne    0x1834 <phase_4+58>
   0x000000000000182e <+52>:    cmp    DWORD PTR [rbp-0x10],0xe
   0x0000000000001832 <+56>:    jbe    0x1839 <phase_4+63>
   0x0000000000001834 <+58>:    call   0x1e02 <explode_bomb>
   0x0000000000001839 <+63>:    mov    edx,0xe
   0x000000000000183e <+68>:    mov    esi,0x0
   0x0000000000001843 <+73>:    mov    edi,DWORD PTR [rbp-0x10]
   0x0000000000001846 <+76>:    call   0x17bc <func4>
   0x000000000000184b <+81>:    cmp    eax,0x6
   0x000000000000184e <+84>:    jne    0x1856 <phase_4+92>
   0x0000000000001850 <+86>:    cmp    DWORD PTR [rbp-0xc],0x6
   0x0000000000001854 <+90>:    je     0x185b <phase_4+97>
   0x0000000000001856 <+92>:    call   0x1e02 <explode_bomb>
   0x000000000000185b <+97>:    mov    rax,QWORD PTR [rbp-0x8]
   0x000000000000185f <+101>:   xor    rax,QWORD PTR fs:0x28
   0x0000000000001868 <+110>:   jne    0x186c <phase_4+114>
   0x000000000000186a <+112>:   leave
   0x000000000000186b <+113>:   ret
   0x000000000000186c <+114>:   call   0x1220 <__stack_chk_fail@plt>
End of assembler dump.
```

反汇编 `func4` 函数

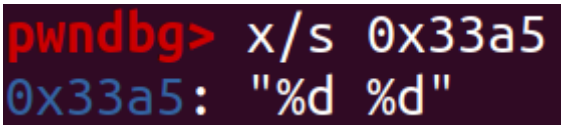
```
Dump of assembler code for function func4:
   0x00000000000017bc <+0>:     endbr64
   0x00000000000017c0 <+4>:     push   rbp
   0x00000000000017c1 <+5>:     mov    rbp,rsp
   0x00000000000017c4 <+8>:     mov    eax,edx
   0x00000000000017c6 <+10>:    sub    eax,esi
   0x00000000000017c8 <+12>:    mov    ecx,eax
   0x00000000000017ca <+14>:    shr    ecx,0x1f
   0x00000000000017cd <+17>:    add    ecx,eax
   0x00000000000017cf <+19>:    sar    ecx,1
   0x00000000000017d1 <+21>:    add    ecx,esi
```

```
   0x00000000000017d3 <+23>:    cmp    ecx,edi
   0x00000000000017d5 <+25>:    jg     0x17e0 <func4+36>
   0x00000000000017d7 <+27>:    mov    eax,0x0
   0x00000000000017dc <+32>:    jl     0x17ec <func4+48>
   0x00000000000017de <+34>:    pop    rbp
   0x00000000000017df <+35>:    ret
   0x00000000000017e0 <+36>:    lea    edx,[rcx-0x1]
   0x00000000000017e3 <+39>:    call   0x17bc <func4>
   0x00000000000017e8 <+44>:    add    eax,eax
   0x00000000000017ea <+46>:    jmp    0x17de <func4+34>
   0x00000000000017ec <+48>:    lea    esi,[rcx+0x1]
   0x00000000000017ef <+51>:    call   0x17bc <func4>
   0x00000000000017f4 <+56>:    lea    eax,[rax+rax*1+0x1]
   0x00000000000017f8 <+60>:    jmp    0x17de <func4+34>
End of assembler dump.
```

查看内存地址

```
0x0000000000001815 <+27>:   lea    rcx,[rbp-0xc]
0x0000000000001819 <+31>:   lea    rdx,[rbp-0x10]
0x000000000000181d <+35>:   lea    rsi,[rip+0x1b81]        # 0x33a5
0x0000000000001824 <+42>:   call   0x12c0 <__isoc99_sscanf@plt>
```

```
(gdb) x/s 0x33a5
```



需要输入两个数字，分别存到 `[rbp-0x10]` 和 `[rbp-0xc]` 里

阅读 `phase_4` 程序我们发现

第一个数字必须小于等于 `14`：

```
0x000000000000182e <+52>:   cmp    DWORD PTR [rbp-0x10],0xe
0x0000000000001832 <+56>:   jbe    0x1839 <phase_4+63>
0x0000000000001834 <+58>:   call   0x1e02 <explode_bomb>
```

`func4` 函数返回值必须为 `6`：

```
x0000000000001846 <+76>:    call   0x17bc <func4>
0x000000000000184b <+81>:   cmp    eax,0x6
0x000000000000184e <+84>:   jne    0x1856 <phase_4+92>
0x0000000000001856 <+92>:   call   0x1e02 <explode_bomb>
```

第二个数字必须为 `6`：

```
0x0000000000001850 <+86>:   cmp    DWORD PTR [rbp-0xc],0x6
0x0000000000001854 <+90>:   je     0x185b <phase_4+97>
0x0000000000001856 <+92>:   call   0x1e02 <explode_bomb>
```

通过解析 `func4` 函数以及需要传入的参数，我们可以推理出，第一个数字也必须为 `6`

故答案为：

> `6  6`

- **phase_5**

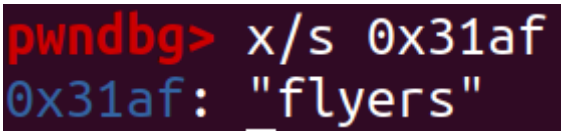
反汇编 `phase_5` 函数

```
Dump of assembler code for function phase_5:
   0x0000000000001871 <+0>:     endbr64
   0x0000000000001875 <+4>:     push   rbp
   0x0000000000001876 <+5>:     mov    rbp,rsp
   0x0000000000001879 <+8>:     push   rbx
   0x000000000000187a <+9>:     sub    rsp,0x18
   0x000000000000187e <+13>:    mov    rbx,rdi
   0x0000000000001881 <+16>:    mov    rax,QWORD PTR fs:0x28
   0x000000000000188a <+25>:    mov    QWORD PTR [rbp-0x18],rax
   0x000000000000188e <+29>:    xor    eax,eax
   0x0000000000001890 <+31>:    call   0x1b65 <string_length>
   0x0000000000001895 <+36>:    cmp    eax,0x6
   0x0000000000001898 <+39>:    jne    0x18ed <phase_5+124>
   0x000000000000189a <+41>:    mov    eax,0x0
   0x000000000000189f <+46>:    lea    rcx,[rip+0x193a]        # 0x31e0 <array.3473>
   0x00000000000018a6 <+53>:    movzx  edx,BYTE PTR [rbx+rax*1]
   0x00000000000018aa <+57>:    and    edx,0xf
   0x00000000000018ad <+60>:    movzx  edx,BYTE PTR [rcx+rdx*1]
   0x00000000000018b1 <+64>:    mov    BYTE PTR [rbp+rax*1-0x1f],dl
   0x00000000000018b5 <+68>:    add    rax,0x1
   0x00000000000018b9 <+72>:    cmp    rax,0x6
   0x00000000000018bd <+76>:    jne    0x18a6 <phase_5+53>
   0x00000000000018bf <+78>:    mov    BYTE PTR [rbp-0x19],0x0
   0x00000000000018c3 <+82>:    lea    rdi,[rbp-0x1f]
   0x00000000000018c7 <+86>:    lea    rsi,[rip+0x18e1]        # 0x31af
   0x00000000000018ce <+93>:    call   0x1b86 <strings_not_equal>
   0x00000000000018d3 <+98>:    test   eax,eax
   0x00000000000018d5 <+100>:   jne    0x18f4 <phase_5+131>
   0x00000000000018d7 <+102>:   mov    rax,QWORD PTR [rbp-0x18]
   0x00000000000018db <+106>:   xor    rax,QWORD PTR fs:0x28
   0x00000000000018e4 <+115>:   jne    0x18fb <phase_5+138>
   0x00000000000018e6 <+117>:   add    rsp,0x18
   0x00000000000018ea <+121>:   pop    rbx
   0x00000000000018eb <+122>:   pop    rbp
   0x00000000000018ec <+123>:   ret
   0x00000000000018ed <+124>:   call   0x1e02 <explode_bomb>
   0x00000000000018f2 <+129>:   jmp    0x189a <phase_5+41>
   0x00000000000018f4 <+131>:   call   0x1e02 <explode_bomb>
   0x00000000000018f9 <+136>:   jmp    0x18d7 <phase_5+102>
   0x00000000000018fb <+138>:   call   0x1220 <__stack_chk_fail@plt>
End of assembler dump.
```

老规矩，查看内存地址

```
0x00000000000018c7 <+86>:   lea    rsi,[rip+0x18e1]        # 0x31af
```

```
(gdb) x/s 0x31af
```



但是我们发现，这关与第一关不一样，并不是直接复制粘贴就行

当我们查看另一个内存地址时

```
0x000000000000189f <+46>:   lea    rcx,[rip+0x193a]        # 0x31e0 <array.3473>
```

```
pwndbg> x/s 0x31e0
0x31e0 <array.3473>:    "maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"
```

> So you think you can stop the bomb with ctrl-c, do you?
>
> 所以你认为你可以用ctrl-c来阻止炸弹，是吗？

显然不行

分析程序知，是通过取我们输入六个字符的 `ASCII` 码的低四位作为索引值，查找 `maduiersnfotvbyl` 里的字符组成

`maduiersnfotvbyl` 中 `f` 为第 `9` 位， `l` 为第 `15` 位， `y` 第 `14` 位， `e` 第 `5` 位， `r` 第 `6` 位， `s` 第 `7` 位

也就是说，我们需要输入6个字符，使它们 `ASCII` 码低四位分别是： `1001` , `1111` , `1110` , `0101` , `0110` , `0111`

所以同样有多种答案（每一位的所有可能的答案如下所示）：

> 1. `I` `Y` `i` `y` ...
> 2. `O` `o` ...
> 3. `N` `n` ...
> 4. `E` `U` `e` `u` ...
> 5. `F` `V` `f` `v` ...
> 6. `G` `W` `g` `w` ...

· phase_6

反汇编 `phase_6` 函数

```
Dump of assembler code for function phase_6:
   0x0000000000001900 <+0>:     endbr64
   0x0000000000001904 <+4>:     push   rbp
   0x0000000000001905 <+5>:     mov    rbp,rsp
   0x0000000000001908 <+8>:     push   r15
   0x000000000000190a <+10>:    push   r14
   0x000000000000190c <+12>:    push   r13
   0x000000000000190e <+14>:    push   r12
   0x0000000000001910 <+16>:    push   rbx
   0x0000000000001911 <+17>:    sub    rsp,0x68
   0x0000000000001915 <+21>:    mov    rax,QWORD PTR fs:0x28
   0x000000000000191e <+30>:    mov    QWORD PTR [rbp-0x38],rax
   0x0000000000001922 <+34>:    xor    eax,eax
   0x0000000000001924 <+36>:    lea    r14,[rbp-0x90]
   0x000000000000192b <+43>:    mov    rsi,r14
   0x000000000000192e <+46>:    call   0x1e42 <read_six_numbers>
   0x0000000000001933 <+51>:    mov    r15d,0x1
   0x0000000000001939 <+57>:    mov    r13,r14
```

```
   0x000000000000193c <+60>:    jmp    0x1968 <phase_6+104>
   0x000000000000193e <+62>:    call   0x1e02 <explode_bomb>
   0x0000000000001943 <+67>:    jmp    0x1976 <phase_6+118>
   0x0000000000001945 <+69>:    add    rbx,0x1
   0x0000000000001949 <+73>:    cmp    ebx,0x5
   0x000000000000194c <+76>:    jg     0x1960 <phase_6+96>
   0x000000000000194e <+78>:    mov    eax,DWORD PTR [r13+rbx*4+0x0]
   0x0000000000001953 <+83>:    cmp    DWORD PTR [r12],eax
   0x0000000000001957 <+87>:    jne    0x1945 <phase_6+69>
   0x0000000000001959 <+89>:    call   0x1e02 <explode_bomb>
   0x000000000000195e <+94>:    jmp    0x1945 <phase_6+69>
   0x0000000000001960 <+96>:    add    r15,0x1
   0x0000000000001964 <+100>:   add    r14,0x4
   0x0000000000001968 <+104>:   mov    r12,r14
   0x000000000000196b <+107>:   mov    eax,DWORD PTR [r14]
   0x000000000000196e <+110>:   sub    eax,0x1
   0x0000000000001971 <+113>:   cmp    eax,0x5
   0x0000000000001974 <+116>:   ja     0x193e <phase_6+62>
   0x0000000000001976 <+118>:   cmp    r15d,0x5
   0x000000000000197a <+122>:   jg     0x1981 <phase_6+129>
   0x000000000000197c <+124>:   mov    rbx,r15
   0x000000000000197f <+127>:   jmp    0x194e <phase_6+78>
   0x0000000000001981 <+129>:   mov    esi,0x0
   0x0000000000001986 <+134>:   mov    ecx,DWORD PTR [rbp+rsi*4-0x90]
   0x000000000000198d <+141>:   mov    eax,0x1
   0x0000000000001992 <+146>:   lea    rdx,[rip+0x3c97]        # 0x5630 <node1>
   0x0000000000001999 <+153>:   cmp    ecx,0x1
   0x000000000000199c <+156>:   jle    0x19a9 <phase_6+169>
   0x000000000000199e <+158>:   mov    rdx,QWORD PTR [rdx+0x8]
   0x00000000000019a2 <+162>:   add    eax,0x1
   0x00000000000019a5 <+165>:   cmp    eax,ecx
   0x00000000000019a7 <+167>:   jne    0x199e <phase_6+158>
   0x00000000000019a9 <+169>:   mov    QWORD PTR [rbp+rsi*8-0x70],rdx
   0x00000000000019ae <+174>:   add    rsi,0x1
   0x00000000000019b2 <+178>:   cmp    rsi,0x6
   0x00000000000019b6 <+182>:   jne    0x1986 <phase_6+134>
   0x00000000000019b8 <+184>:   mov    rbx,QWORD PTR [rbp-0x70]
   0x00000000000019bc <+188>:   mov    rax,QWORD PTR [rbp-0x68]
   0x00000000000019c0 <+192>:   mov    QWORD PTR [rbx+0x8],rax
   0x00000000000019c4 <+196>:   mov    rdx,QWORD PTR [rbp-0x60]
   0x00000000000019c8 <+200>:   mov    QWORD PTR [rax+0x8],rdx
   0x00000000000019cc <+204>:   mov    rax,QWORD PTR [rbp-0x58]
   0x00000000000019d0 <+208>:   mov    QWORD PTR [rdx+0x8],rax
   0x00000000000019d4 <+212>:   mov    rdx,QWORD PTR [rbp-0x50]
   0x00000000000019d8 <+216>:   mov    QWORD PTR [rax+0x8],rdx
   0x00000000000019dc <+220>:   mov    rax,QWORD PTR [rbp-0x48]
   0x00000000000019e0 <+224>:   mov    QWORD PTR [rdx+0x8],rax
   0x00000000000019e4 <+228>:   mov    QWORD PTR [rax+0x8],0x0
   0x00000000000019ec <+236>:   mov    r12d,0x5
   0x00000000000019f2 <+242>:   jmp    0x19fe <phase_6+254>
   0x00000000000019f4 <+244>:   mov    rbx,QWORD PTR [rbx+0x8]
   0x00000000000019f8 <+248>:   sub    r12d,0x1
   0x00000000000019fc <+252>:   je     0x1a0f <phase_6+271>
   0x00000000000019fe <+254>:   mov    rax,QWORD PTR [rbx+0x8]
   0x0000000000001a02 <+258>:   mov    eax,DWORD PTR [rax]
   0x0000000000001a04 <+260>:   cmp    DWORD PTR [rbx],eax
```

```
   0x0000000000001a06 <+262>:   jge    0x19f4 <phase_6+244>
   0x0000000000001a08 <+264>:   call   0x1e02 <explode_bomb>
   0x0000000000001a0d <+269>:   jmp    0x19f4 <phase_6+244>
   0x0000000000001a0f <+271>:   mov    rax,QWORD PTR [rbp-0x38]
   0x0000000000001a13 <+275>:   xor    rax,QWORD PTR fs:0x28
   0x0000000000001a1c <+284>:   jne    0x1a2d <phase_6+301>
   0x0000000000001a1e <+286>:   add    rsp,0x68
   0x0000000000001a22 <+290>:   pop    rbx
   0x0000000000001a23 <+291>:   pop    r12
   0x0000000000001a25 <+293>:   pop    r13
   0x0000000000001a27 <+295>:   pop    r14
   0x0000000000001a29 <+297>:   pop    r15
   0x0000000000001a2b <+299>:   pop    rbp
   0x0000000000001a2c <+300>:   ret
   0x0000000000001a2d <+301>:   call   0x1220 <__stack_chk_fail@plt>
End of assembler dump.
```

题目要求输入6个数字：

```
0x000000000000192e <+46>:   call   0x1e42 <read_six_numbers>
```

输入的数小于等于6且两两不能相等：

```
0x0000000000001945 <+69>:   add    rbx,0x1
0x0000000000001949 <+73>:   cmp    ebx,0x5
0x000000000000194c <+76>:   jg     0x1960 <phase_6+96>
0x000000000000194e <+78>:   mov    eax,DWORD PTR [r13+rbx*4+0x0]
0x0000000000001953 <+83>:   cmp    DWORD PTR [r12],eax
0x0000000000001957 <+87>:   jne    0x1945 <phase_6+69>
0x0000000000001959 <+89>:   call   0x1e02 <explode_bomb>
```

同时通过地址我们发现了链表，其含6个节点：

```
pwndbg> x/24wx 0x5630
0x5630 <node1>: 0x00000145      0x00000001      0x00005640      0x00000000
0x5640 <node2>: 0x00000273      0x00000002      0x00005650      0x00000000
0x5650 <node3>: 0x000000f4      0x00000003      0x00005660      0x00000000
0x5660 <node4>: 0x0000016c      0x00000004      0x00005670      0x00000000
0x5670 <node5>: 0x00000205      0x00000005      0x00005120      0x00000000
```

分析程序的最后部分发现，根据链表顺序最终的数字应是从大到小排列

> `2 5 4 1 6 3`

· secret_phase

反汇编 `phase_defused` 函数

```
Dump of assembler code for function phase_defused:
   0x0000000000001fc9 <+0>:     endbr64
   0x0000000000001fcd <+4>:     push   rbp
   0x0000000000001fce <+5>:     mov    rbp,rsp
   0x0000000000001fd1 <+8>:     sub    rsp,0x70
   0x0000000000001fd5 <+12>:    mov    rax,QWORD PTR fs:0x28
   0x0000000000001fde <+21>:    mov    QWORD PTR [rbp-0x8],rax
   0x0000000000001fe2 <+25>:    xor    eax,eax
   0x0000000000001fe4 <+27>:    mov    edi,0x1
```

```
   0x0000000000001fe9 <+32>:    call   0x1d1b <send_msg>
   0x0000000000001fee <+37>:    cmp    DWORD PTR [rip+0x3ab7],0x6        # 0x5aac
<num_input_strings>
   0x0000000000001ff5 <+44>:    je     0x2008 <phase_defused+63>
   0x0000000000001ff7 <+46>:    mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000001ffb <+50>:    xor    rax,QWORD PTR fs:0x28
   0x0000000000002004 <+59>:    jne    0x2083 <phase_defused+186>
   0x0000000000002006 <+61>:    leave
   0x0000000000002007 <+62>:    ret
   0x0000000000002008 <+63>:    lea    rcx,[rbp-0x64]
   0x000000000000200c <+67>:    lea    rdx,[rbp-0x68]
   0x0000000000002010 <+71>:    lea    r8,[rbp-0x60]
   0x0000000000002014 <+75>:    lea    rsi,[rip+0x13d4]        # 0x33ef
   0x000000000000201b <+82>:    lea    rdi,[rip+0x3b8e]        # 0x5bb0
<input_strings+240>
   0x0000000000002022 <+89>:    mov    eax,0x0
   0x0000000000002027 <+94>:    call   0x12c0 <__isoc99_sscanf@plt>
   0x000000000000202c <+99>:    cmp    eax,0x3
   0x000000000000202f <+102>:   je     0x204b <phase_defused+130>
   0x0000000000002031 <+104>:   lea    rdi,[rip+0x1278]        # 0x32b0
   0x0000000000002038 <+111>:   call   0x1200 <puts@plt>
   0x000000000000203d <+116>:   lea    rdi,[rip+0x129c]        # 0x32e0
   0x0000000000002044 <+123>:   call   0x1200 <puts@plt>
   0x0000000000002049 <+128>:   jmp    0x1ff7 <phase_defused+46>
   0x000000000000204b <+130>:   lea    rdi,[rbp-0x60]
   0x000000000000204f <+134>:   lea    rsi,[rip+0x13a2]        # 0x33f8
   0x0000000000002056 <+141>:   call   0x1b86 <strings_not_equal>
   0x000000000000205b <+146>:   test   eax,eax
   0x000000000000205d <+148>:   jne    0x2031 <phase_defused+104>
   0x000000000000205f <+150>:   lea    rdi,[rip+0x11ea]        # 0x3250
   0x0000000000002066 <+157>:   call   0x1200 <puts@plt>
   0x000000000000206b <+162>:   lea    rdi,[rip+0x1206]        # 0x3278
   0x0000000000002072 <+169>:   call   0x1200 <puts@plt>
   0x0000000000002077 <+174>:   mov    eax,0x0
   0x000000000000207c <+179>:   call   0x1a70 <secret_phase>
   0x0000000000002081 <+184>:   jmp    0x2031 <phase_defused+104>
   0x0000000000002083 <+186>:   call   0x1220 <__stack_chk_fail@plt>
End of assembler dump.
```

查看这里的几个内存地址

```
0x000000000000205f <+150>:  lea    rdi,[rip+0x11ea]        # 0x3250
```

```
pwndbg> x/s 0x3250
0x3250: "Curses, you've found the secret phase!"
```

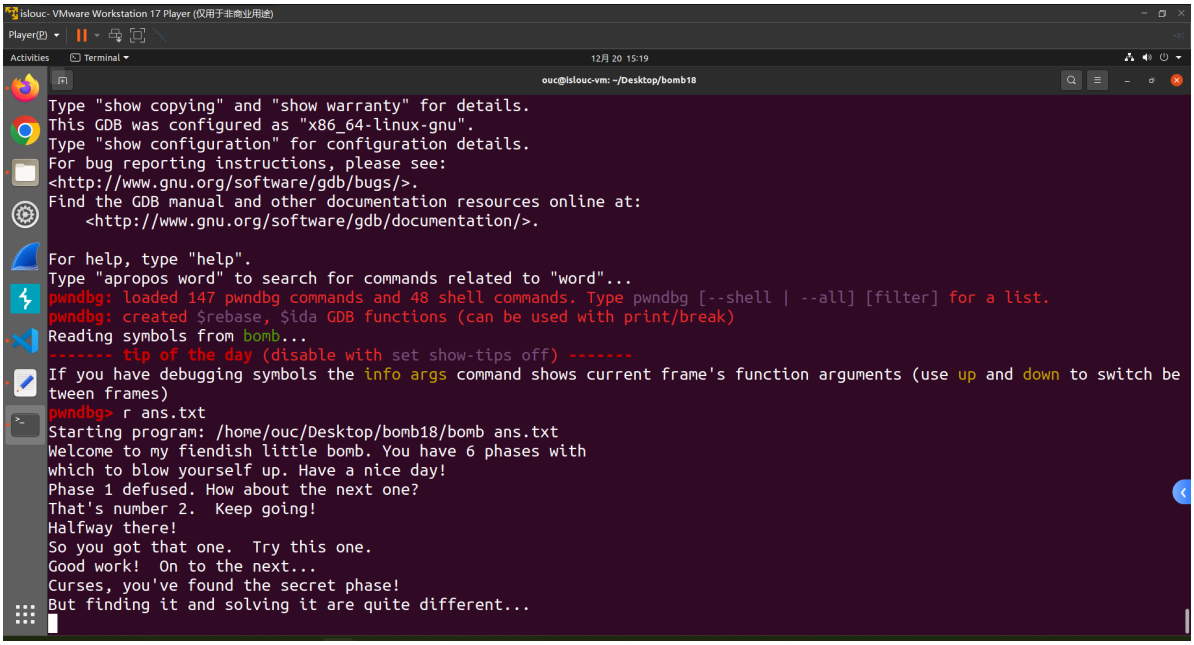
```
0x000000000000204f <+134>:  lea    rsi,[rip+0x13a2]        # 0x33f8
```

```
pwndbg> x/s 0x33f8
0x33f8: "DrEvil"
```

```
0x0000000000002014 <+75>:   lea    rsi,[rip+0x13d4]        # 0x33ef
```

```
pwndbg> x/s 0x33ef
0x33ef: "%d %d %s"
```

显然有个地方是需要我们输入两个数字加一个字符串的，能输入两个数字的就是第四关，我们在其答案的后面添加上 `DrEvil`，成功进入隐藏关卡



反汇编 `secret_phase` 函数

```
Dump of assembler code for function secret_phase:
   0x0000555555555a70 <+0>:     endbr64
   0x0000555555555a74 <+4>:     push   rbp
   0x0000555555555a75 <+5>:     mov    rbp,rsp
   0x0000555555555a78 <+8>:     push   rbx
   0x0000555555555a79 <+9>:     sub    rsp,0x8
   0x0000555555555a7d <+13>:    call   0x555555555e84 <read_line>
   0x0000555555555a82 <+18>:    mov    rdi,rax
   0x0000555555555a85 <+21>:    mov    edx,0xa
   0x0000555555555a8a <+26>:    mov    esi,0x0
   0x0000555555555a8f <+31>:    call   0x5555555552a0 <strtol@plt>
   0x0000555555555a94 <+36>:    mov    rbx,rax
   0x0000555555555a97 <+39>:    lea    eax,[rax-0x1]
   0x0000555555555a9a <+42>:    cmp    eax,0x3e8
   0x0000555555555a9f <+47>:    ja     0x555555555acc <secret_phase+92>
   0x0000555555555aa1 <+49>:    mov    esi,ebx
   0x0000555555555aa3 <+51>:    lea    rdi,[rip+0x3aa6]        # 0x555555559550 <n1>
   0x0000555555555aaa <+58>:    call   0x555555555a32 <fun7>
   0x0000555555555aaf <+63>:    cmp    eax,0x2
   0x0000555555555ab2 <+66>:    jne    0x555555555ad3 <secret_phase+99>
   0x0000555555555ab4 <+68>:    lea    rdi,[rip+0x16c5]        # 0x555555557180
   0x0000555555555abb <+75>:    call   0x555555555200 <puts@plt>
   0x0000555555555ac0 <+80>:    call   0x555555555fc9 <phase_defused>
   0x0000555555555ac5 <+85>:    add    rsp,0x8
   0x0000555555555ac9 <+89>:    pop    rbx
   0x0000555555555aca <+90>:    pop    rbp
   0x0000555555555acb <+91>:    ret
   0x0000555555555acc <+92>:    call   0x555555555e02 <explode_bomb>
   0x0000555555555ad1 <+97>:    jmp    0x555555555aa1 <secret_phase+49>
   0x0000555555555ad3 <+99>:    call   0x555555555e02 <explode_bomb>
   0x0000555555555ad8 <+104>:   jmp    0x555555555ab4 <secret_phase+68>
End of assembler dump.
```

反汇编 `fun7` 函数

```
Dump of assembler code for function fun7:
   0x0000555555555a32 <+0>:     endbr64
   0x0000555555555a36 <+4>:     test   rdi,rdi
   0x0000555555555a39 <+7>:     je     0x555555555a6a <fun7+56>
   0x0000555555555a3b <+9>:     push   rbp
   0x0000555555555a3c <+10>:    mov    rbp,rsp
   0x0000555555555a3f <+13>:    mov    edx,DWORD PTR [rdi]
   0x0000555555555a41 <+15>:    cmp    edx,esi
   0x0000555555555a43 <+17>:    jg     0x555555555a4e <fun7+28>
   0x0000555555555a45 <+19>:    mov    eax,0x0
   0x0000555555555a4a <+24>:    jne    0x555555555a5b <fun7+41>
   0x0000555555555a4c <+26>:    pop    rbp
   0x0000555555555a4d <+27>:    ret
   0x0000555555555a4e <+28>:    mov    rdi,QWORD PTR [rdi+0x8]
   0x0000555555555a52 <+32>:    call   0x555555555a32 <fun7>
   0x0000555555555a57 <+37>:    add    eax,eax
   0x0000555555555a59 <+39>:    jmp    0x555555555a4c <fun7+26>
   0x0000555555555a5b <+41>:    mov    rdi,QWORD PTR [rdi+0x10]
   0x0000555555555a5f <+45>:    call   0x555555555a32 <fun7>
   0x0000555555555a64 <+50>:    lea    eax,[rax+rax*1+0x1]
   0x0000555555555a68 <+54>:    jmp    0x555555555a4c <fun7+26>
   0x0000555555555a6a <+56>:    mov    eax,0xffffffff
   0x0000555555555a6f <+61>:    ret
End of assembler dump.
```

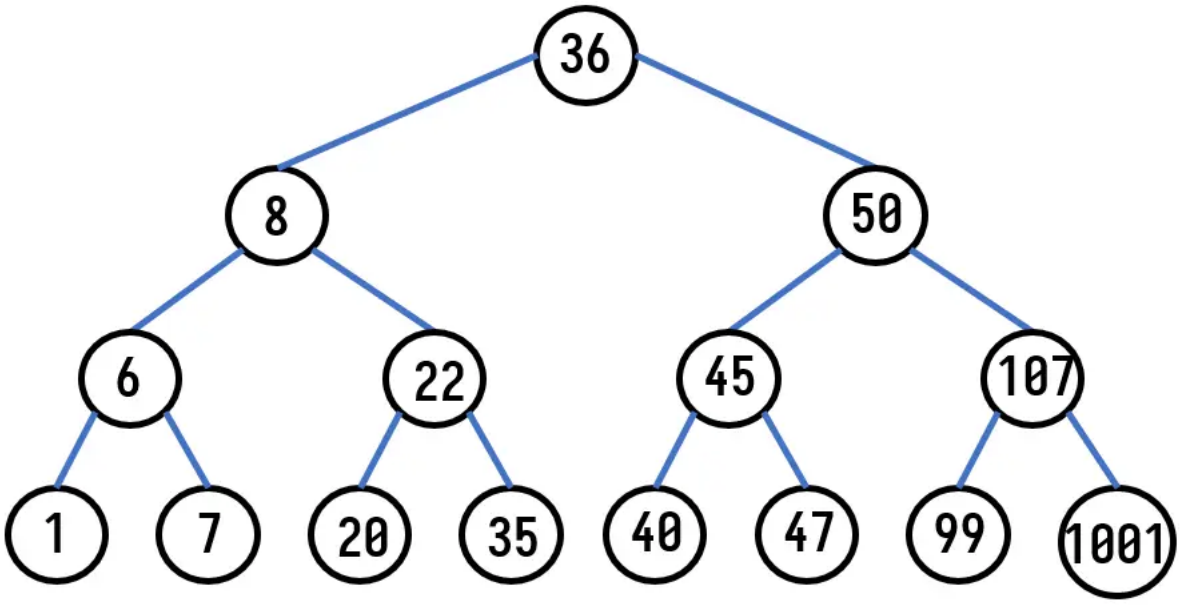
先分析 `secret_phase` 函数：

```
0x0000555555555aa1 <+49>:   mov    esi,ebx
0x0000555555555aa3 <+51>:   lea    rdi,[rip+0x3aa6]        # 0x555555559550 <n1>
0x0000555555555aaa <+58>:   call   0x555555555a32 <fun7>
0x0000555555555aaf <+63>:   cmp    eax,0x2
```

这里向 `fun7` 传了两个参数，并要求返回值为2，其中一个参数 `esi` 存的是我们输入的数，另一个是内存地址：

```
pwndbg> x/56x 0x555555559550
0x555555559550 <n1>:        0x00000024  0x00000000  0x00005570  0x00000000
0x555555559560 <n1+16>:     0x00005590  0x00000000  0x00000000  0x00000000
0x555555559570 <n21>:       0x00000008  0x00000000  0x000055f0  0x00000000
0x555555559580 <n21+16>:    0x000055b0  0x00000000  0x00000000  0x00000000
0x555555559590 <n22>:       0x00000032  0x00000000  0x000055d0  0x00000000
0x5555555595a0 <n22+16>:    0x00005610  0x00000000  0x00000000  0x00000000
0x5555555595b0 <n32>:       0x00000016  0x00000000  0x000050c0  0x00000000
0x5555555595c0 <n32+16>:    0x00005080  0x00000000  0x00000000  0x00000000
0x5555555595d0 <n33>:       0x0000002d  0x00000000  0x00005020  0x00000000
0x5555555595e0 <n33+16>:    0x000050e0  0x00000000  0x00000000  0x00000000
0x5555555595f0 <n31>:       0x00000006  0x00000000  0x00005040  0x00000000
0x555555559600 <n31+16>:    0x000050a0  0x00000000  0x00000000  0x00000000
0x555555559610 <n34>:       0x0000006b  0x00000000  0x00005060  0x00000000
0x555555559620 <n34+16>:    0x00005100  0x00000000  0x00000000  0x00000000
```

这类似于树的结构，根据节点关系，可以画出如下图所示的树：



根据对 `fun7` 函数的分析，想要返回2，推理出 `22` 是正确的结果

> 22

四、实验结果：

- phase_1

```
pwndbg> r
Starting program: /home/ouc/Desktop/bomb18/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
When a problem comes along, you must zip it!
Phase 1 defused. How about the next one?
```

- phase_2

```
2 3 5 8 12 17
That's number 2.  Keep going!
```

- phase_3

```
7 w 225
Halfway there!
```

- phase_4

```
6 6
So you got that one.  Try this one.
```

- phase_5

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

- phase_6

```
2 5 4 1 6 3
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
[Inferior 1 (process 45018) exited normally]
```

- secret_phase

```
pwndbg> r ans.txt
Starting program: /home/ouc/Desktop/bomb18/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!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
[Inferior 1 (process 45068) exited normally]
```

Open ans.txt ~/Desktop/bomb18

```
When a problem comes along, you must zip it!
2 3 5 8 12 17
7 w 225
6 6 DrEvil
ionuvw
2 5 4 1 6 3
20
```

五、实验总结：

> - 实验过程中，我经历了七个不同阶段的挑战，每个阶段都考察了机器级语言程序的不同方面，随着阶段的递增，难度也逐渐提升。在这个过程中，我深刻体会到了计算机科学中的一些重要概念，并从中获得了丰富的经验和收获。
> - 在第一阶段的字符串比较中，我对字符串的处理有了更深入的了解，学会了如何有效地进行字符串比较操作。这为后续的阶段奠定了基础，也提升了我的编程技能。
> - 第二阶段涉及循环结构，我学到了如何使用循环语句来处理复杂的任务，同时也加深了对程序执行流程的理解。这对于编写高效且可维护的代码至关重要。
> - 在第三阶段，我面对条件和分支结构，包括switch语句的运用。这使我更加熟悉如何根据不同的情况执行不同的代码段，提高了我的程序设计灵活性。
> - 第四阶段涉及递归调用和栈的使用，这是一个相对较复杂的部分。通过解决这一阶段的问题，我深入了解了递归的原理和栈的运作机制，为处理更加复杂的程序问题打下了基础。
> - 指针是编程中一个重要而复杂的概念，在第五阶段，我深入研究了指针的应用。这帮助我更好地理解内存管理和数据结构，为高效的内存操作提供了技能支持。
> - 第六阶段涉及链表、指针和结构的操作，这对于处理更加复杂的数据结构和算法问题至关重要。我通过这一阶段的实践，掌握了处理复杂数据结构的技能。
> - 在实验的隐藏阶段，我面对了一些挑战，特别是在处理阶段四之后附加特定字符串的情况。这要求我灵活运用之前学到的知识，解决问题的能力得到了锻炼。
> - 总的来说，通过这个实验，我不仅加深了对机器级语言程序设计的理解，还提高了解决复杂问题的能力。在未来的学习和工作中，我将继续运用这些经验和技能，不断提升自己在计算机科学领域的水平。这次实验是一次深刻的学习过程，为我未来的编程生涯打下了坚实的基础。