BombLab 实验报告

phase1

```
0000000000002439 <phase_1>:
    2439:    48 83 ec 08              sub    $0x8,%rsp
    243d:    48 8d 35 04 1d 00 00     lea    0x1d04(%rip),%rsi        # 4148
<_IO_stdin_used+0x148>
    2444:    e8 f4 04 00 00           callq  293d <strings_not_equal>
    2449:    85 c0                    test   %eax,%eax
    244b:    75 05                    jne    2452 <phase_1+0x19>
    244d:    48 83 c4 08              add    $0x8,%rsp
    2451:    c3                       retq
    2452:    e8 c2 07 00 00           callq  2c19 <explode_bomb>
    2457:    eb f4                    jmp    244d <phase_1+0x14>
```

`2444: e8 f4 04 00 00 callq 293d <strings_not_equal>`行调用`<strings_not_equal>`函数，第一个参数在寄存器`%rdi`中，是我们输入的input string的地址，第二个在`%rsi`中，在`2444`行调用函数前设置断点，在gdb中使用`x/s $rsi`查看即可得到答案。

phase2

```
0000000000002459 <phase_2>:
    2459:    55                       push   %rbp
    245a:    53                       push   %rbx
    245b:    48 83 ec 28              sub    $0x28,%rsp
    245f:    64 48 8b 04 25 28 00     mov    %fs:0x28,%rax
    2466:    00 00
    2468:    48 89 44 24 18           mov    %rax,0x18(%rsp)
    246d:    31 c0                    xor    %eax,%eax
    246f:    48 89 e6                 mov    %rsp,%rsi
    2472:    e8 62 08 00 00           callq  2cd9 <read_six_numbers>
    2477:    83 3c 24 03              cmpl   $0x3,(%rsp)
    247b:    75 07                    jne    2484 <phase_2+0x2b>
    247d:    83 7c 24 04 11           cmpl   $0x11,0x4(%rsp)
    2482:    74 05                    je     2489 <phase_2+0x30>
    2484:    e8 90 07 00 00           callq  2c19 <explode_bomb>
    2489:    48 89 e3                 mov    %rsp,%rbx
    248c:    48 8d 6c 24 10           lea    0x10(%rsp),%rbp
    2491:    eb 09                    jmp    249c <phase_2+0x43>
    2493:    48 83 c3 04              add    $0x4,%rbx
    2497:    48 39 eb                 cmp    %rbp,%rbx
    249a:    74 14                    je     24b0 <phase_2+0x57>
    249c:    8b 03                    mov    (%rbx),%eax
    249e:    8d 04 40                 lea    (%rax,%rax,2),%eax
    24a1:    03 43 04                 add    0x4(%rbx),%eax
    24a4:    39 43 08                 cmp    %eax,0x8(%rbx)
```

```
    24a7:    74 ea                    je     2493 <phase_2+0x3a>
    24a9:    e8 6b 07 00 00           callq  2c19 <explode_bomb>
    24ae:    eb e3                    jmp    2493 <phase_2+0x3a>
    24b0:    48 8b 44 24 18           mov    0x18(%rsp),%rax
    24b5:    64 48 2b 04 25 28 00     sub    %fs:0x28,%rax
    24bc:    00 00
    24be:    75 07                    jne    24c7 <phase_2+0x6e>
    24c0:    48 83 c4 28              add    $0x28,%rsp
    24c4:    5b                       pop    %rbx
    24c5:    5d                       pop    %rbp
    24c6:    c3                       retq
    24c7:    e8 d4 fb ff ff           callq  20a0 <__stack_chk_fail@plt>
```

这个phase先调用`read_six_numbers`函数，于是我们不妨先看一下其汇编代码; 不难发现我们需要输入六个正数，保存在`%rsp`及其更高地址段内,分别对应`(%rsp)`,`(4+%rsp)`,`(8+%rsp)`,`(0xc+%rsp)`,`(0x10+%rsp)`,`(0x14+%rsp)`。

```
2477: 83 3c 24 03 cmpl $0x3,(%rsp)
```

```
247d: 83 7c 24 04 11 cmpl $0x11,0x4(%rsp)
```

调用后紧接着是上面的两个比较，发现必须都相等才行；于是，我们所输入的第一个数需要是`3`,第二个数需要是`0x11`也即`17`；此后，先`lea 0x10(%rsp),%rbp`，将第5个数对应的地址保存在了`%rbp`中，把第一个数的地址保存在了`%rbx`中，然后跳到循环:

```
    2493:    48 83 c3 04              add    $0x4,%rbx
    2497:    48 39 eb                 cmp    %rbp,%rbx
    249a:    74 14                    je     24b0 <phase_2+0x57>

    249c:    8b 03                    mov    (%rbx),%eax
    249e:    8d 04 40                 lea    (%rax,%rax,2),%eax
    24a1:    03 43 04                 add    0x4(%rbx),%eax
    24a4:    39 43 08                 cmp    %eax,0x8(%rbx)
    24a7:    74 ea                    je     2493 <phase_2+0x3a>
```

可见`%rbx`作为迭代器逐渐向高地址推，过程中不断检查相等，当且仅当`%rbx`推到`%rbp`处时完成检查。

循环中，计算"`%rdx`对应的地址上的数的三倍加上下一个数"的结果，将其与`%rbx`对应地址上的数的下下个数比较，二者相等才行。这样就不难看出为什么`%rbp`设置为`0x10(%rsp)`了，其对应第四个数作为起始检查完后`%rbx`加4后的地址。

phase3

```
0000000000024cc <phase_3>:
    24cc:    48 83 ec 18              sub    $0x18,%rsp
    24d0:    64 48 8b 04 25 28 00     mov    %fs:0x28,%rax
    24d7:    00 00
    24d9:    48 89 44 24 08           mov    %rax,0x8(%rsp)
    24de:    31 c0                    xor    %eax,%eax
```

```
    24e0:    48 8d 4c 24 04           lea    0x4(%rsp),%rcx
    24e5:    48 89 e2                 mov    %rsp,%rdx
    24e8:    48 8d 35 32 21 00 00     lea    0x2132(%rip),%rsi        # 4621
<array.0+0x421>
    24ef:    e8 5c fc ff ff           callq  2150 <__isoc99_sscanf@plt>
    24f4:    83 f8 01                 cmp    $0x1,%eax
    24f7:    7e 1d                    jle    2516 <phase_3+0x4a>
    24f9:    83 3c 24 07              cmpl   $0x7,(%rsp)
    24fd:    0f 87 c0 00 00 00        ja     25c3 <phase_3+0xf7>
    2503:    8b 04 24                 mov    (%rsp),%eax
    2506:    48 8d 15 d3 1c 00 00     lea    0x1cd3(%rip),%rdx        # 41e0
<_IO_stdin_used+0x1e0>
    250d:    48 63 04 82              movslq (%rdx,%rax,4),%rax
    2511:    48 01 d0                 add    %rdx,%rax
    2514:    ff e0                    jmpq   *%rax
    2516:    e8 fe 06 00 00           callq  2c19 <explode_bomb>
    251b:    eb dc                    jmp    24f9 <phase_3+0x2d>
    251d:    8b 15 fd 4b 00 00        mov    0x4bfd(%rip),%edx        # 7120
<delta.1>
    2523:    b8 b9 01 00 00           mov    $0x1b9,%eax
    2528:    29 d0                    sub    %edx,%eax
    252a:    8b 54 24 04              mov    0x4(%rsp),%edx
    252e:    85 d2                    test   %edx,%edx
    2530:    78 04                    js     2536 <phase_3+0x6a>
    2532:    39 c2                    cmp    %eax,%edx
    2534:    74 05                    je     253b <phase_3+0x6f>
    2536:    e8 de 06 00 00           callq  2c19 <explode_bomb>
    253b:    48 8b 44 24 08           mov    0x8(%rsp),%rax
    2540:    64 48 2b 04 25 28 00     sub    %fs:0x28,%rax
    2547:    00 00
    2549:    0f 85 83 00 00 00        jne    25d2 <phase_3+0x106>
    254f:    48 83 c4 18              add    $0x18,%rsp
    2553:    c3                       retq
    2554:    8b 15 c6 4b 00 00        mov    0x4bc6(%rip),%edx        # 7120
<delta.1>
    255a:    b8 df 03 00 00           mov    $0x3df,%eax
    255f:    29 d0                    sub    %edx,%eax
    2561:    eb c7                    jmp    252a <phase_3+0x5e>
    2563:    8b 15 b7 4b 00 00        mov    0x4bb7(%rip),%edx        # 7120
<delta.1>
    2569:    b8 7d 02 00 00           mov    $0x27d,%eax
    256e:    29 d0                    sub    %edx,%eax
    2570:    eb b8                    jmp    252a <phase_3+0x5e>
```

`24e8: 48 8d 35 32 21 00 00 lea 0x2132(%rip),%rsi`出现在`sscanf`之前，查看后发现是`%d %d`，可知输入需要是两个整数。抓函数中的几个比较`24f9: 83 3c 24 07 cmpl $0x7,(%rsp)`结合下一行`jg`可见第一个输入需要小于等于`7`；而后的一处比较`2532: 39 c2 cmp %eax,%edx`也需相等，查看`mov 0x4bfd(%rip),%edx`移动的数值可知为`916`。分析后续跳转不难发现这是一个`switch`语句，只有当第一个数是`1`第二个数是`916`时才能躲避炸弹。

phase4

sscanf的第二个参数%rsi仍为%d %d，输入是两个整数。调用的fuc4函数是个递归函数，感觉还要推合适输出，需要遍历一些值，直接通过汇编来看有点麻烦，就试着写了一下对应c语言代码，调通后如下：

```
int func4(int i, int x, int k)
{
    int ans = x;
    ans -= i;
    int y = ans;
    y = y>>31;
    y += ans;
    y = y>>1;
    y += i;
    if(y>k)
    {
        x = y-1;
        ans = func4(i, x, k);
        ans = ans*2;
        return ans;
    }
    if(y<k)
    {
        ans = 0;
        i = y+1;
        ans = func4(i, x, k);
        ans = 2*ans + 1;
        return ans;
    }
    ans=0;
    return ans;
}
int main()
{
    int x=14;
    int i=0;
    int k=6;
    int ans = func4(i,x,k);
    printf("%d", ans);
}
```

```
    265e:    8b 4c 24 04             mov    0x4(%rsp),%ecx
    2662:    8d 51 f4                lea    -0xc(%rcx),%edx
    2665:    89 54 24 04             mov    %edx,0x4(%rsp)
    2669:    83 fa 06                cmp    $0x6,%edx
    266c:    75 05                   jne    2673 <phase_4+0x5f>
    266e:    83 f8 06                cmp    $0x6,%eax
```

调用完func4后是上面的两个检查比较；可见func4的返回值需要是6，改变上述代码中的k尝试后发现6即为所求；第二个参数0x4(%rsp) 减去12后赋值给了%edx，需要等于6,可知第二个输入需要是18 。

phase5

```
0000000000002692 <phase_5>:
    2692:    53                      push   %rbx
    2693:    48 89 fb                mov    %rdi,%rbx
    2696:    e8 85 02 00 00          callq  2920 <string_length>
    269b:    83 f8 06                cmp    $0x6,%eax
    269e:    75 2c                   jne    26cc <phase_5+0x3a>
    26a0:    48 89 d8                mov    %rbx,%rax
    26a3:    48 8d 7b 06             lea    0x6(%rbx),%rdi
    26a7:    b9 00 00 00 00          mov    $0x0,%ecx
    26ac:    48 8d 35 4d 1b 00 00    lea    0x1b4d(%rip),%rsi        # 4200
<array.0>
    26b3:    0f b6 10                movzbl (%rax),%edx
    26b6:    83 e2 0f                and    $0xf,%edx
    26b9:    03 0c 96                add    (%rsi,%rdx,4),%ecx
    26bc:    48 83 c0 01             add    $0x1,%rax
    26c0:    48 39 f8                cmp    %rdi,%rax
    26c3:    75 ee                   jne    26b3 <phase_5+0x21>
    26c5:    83 f9 2b                cmp    $0x2b,%ecx
    26c8:    75 09                   jne    26d3 <phase_5+0x41>
    26ca:    5b                      pop    %rbx
    26cb:    c3                      retq
    26cc:    e8 48 05 00 00          callq  2c19 <explode_bomb>
    26d1:    eb cd                   jmp    26a0 <phase_5+0xe>
    26d3:    e8 41 05 00 00          callq  2c19 <explode_bomb>
    26d8:    eb f0                   jmp    26ca <phase_5+0x38>
```

在一开始，调用<string_length>后返回值需要为6，猜测输入字符串长度即需为6。gdb试了一下后发现读取的是%c(%s)类型，即保存的是ASCII码。使用123456发现，下面的代码是一个累加的过程,1对应加10，2对应加6，3对应加1，4对应加12，5对应加16，6对应加9；而最终需要的是0x2b即43，33=6*4+10+9,输入222216即可累加出所需结果。

phase6

```
00000000000026da <phase_6>:
    26da:    41 56                   push   %r14
    26dc:    41 55                   push   %r13
    26de:    41 54                   push   %r12
    26e0:    55                      push   %rbp
    26e1:    53                      push   %rbx
    26e2:    48 83 ec 60             sub    $0x60,%rsp
    26e6:    64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
    26ed:    00 00
    26ef:    48 89 44 24 58          mov    %rax,0x58(%rsp)
    26f4:    31 c0                   xor    %eax,%eax
    26f6:    49 89 e5                mov    %rsp,%r13
    26f9:    4c 89 ee                mov    %r13,%rsi
    26fc:    e8 d8 05 00 00          callq  2cd9 <read_six_numbers>
```

```
    2701:    41 be 01 00 00 00        mov    $0x1,%r14d
    2707:    49 89 e4                 mov    %rsp,%r12 #上面各个操作初始化几个寄存器
均保存输入数字串的起始地址（第一个数，top of stack）
    270a:    eb 28                    jmp    2734 <phase_6+0x5a>
    270c:    e8 08 05 00 00           callq  2c19 <explode_bomb>
    2711:    eb 30                    jmp    2743 <phase_6+0x69>

    2713:    48 83 c3 01              add    $0x1,%rbx #计数器ebx
    2717:    83 fb 05                 cmp    $0x5,%ebx
    271a:    7f 10                    jg     272c <phase_6+0x52> #确实互不相等，跳
出内层循环，执行下一个外层循环
    271c:    41 8b 04 9c              mov    (%r12,%rbx,4),%eax #取出当前数
    2720:    39 45 00                 cmp    %eax,0x0(%rbp) #和“滑动窗口”rbp的第一
个元素比较
    2723:    75 ee                    jne    2713 <phase_6+0x39> #必须不等。这个嵌
套循环在验证6个数是否满足互不相同
    2725:    e8 ef 04 00 00           callq  2c19 <explode_bomb>
    272a:    eb e7                    jmp    2713 <phase_6+0x39>
    272c:    49 83 c6 01              add    $0x1,%r14
    2730:    49 83 c5 04              add    $0x4,%r13
    2734:    4c 89 ed                 mov    %r13,%rbp #r13继承rbp，mov给rbp相当于
rbp上移了4字节
    2737:    41 8b 45 00              mov    0x0(%r13),%eax #取窗口第一个数
    273b:    83 e8 01                 sub    $0x1,%eax
    273e:    83 f8 05                 cmp    $0x5,%eax #这里在外层循环处验证当前窗口
的第一个数要<6
    2741:    77 c9                    ja     270c <phase_6+0x32>

    2743:    41 83 fe 05              cmp    $0x5,%r14d #>=6则看它所在的索引（从1开
始）是否到6了
    2747:    7f 05                    jg     274e <phase_6+0x74> #到了6，则继续接下
来的事情
    2749:    4c 89 f3                 mov    %r14,%rbx #还没到6呢，直接把r14给rbx继
续循环就行
    274c:    eb ce                    jmp    271c <phase_6+0x42> #BACK

    #至此，上述代码保证了每个数都不相等且都小于等于6
    274e:    be 00 00 00 00           mov    $0x0,%esi #计数器esi

    2753:    8b 0c b4                 mov    (%rsp,%rsi,4),%ecx #取出输入的第一个数
存到ecx
    2756:    b8 01 00 00 00           mov    $0x1,%eax
    275b:    48 8d 15 fe eb 00 00     lea    0xebfe(%rip),%rdx        # 11360
<node1>的地址赋给rdx

    2762:    83 f9 01                 cmp    $0x1,%ecx #第一个数小于等于1则jump
    2765:    7e 0b                    jle    2772 <phase_6+0x98>

    2767:    48 8b 52 08              mov    0x8(%rdx),%rdx #rdx窗口上移，推进到下
一个node
    276b:    83 c0 01                 add    $0x1,%eax #eax从1开始不断+1
    276e:    39 c8                    cmp    %ecx,%eax #没加1的eax和第一个数相等则存
node，否则继续推窗口，直到没加1的eax和第一个数相等时才存
    2770:    75 f5                    jne    2767 <phase_6+0x8d>
```

```
    2772:    48 89 54 f4 20           mov    %rdx,0x20(%rsp,%rsi,8) #把当前rdx中的
node的地址存到rsp的合适位置
    2777:    48 83 c6 01              add    $0x1,%rsi
    277b:    48 83 fe 06              cmp    $0x6,%rsi #计数器<=6则继续循环，rsi意思
是存好的node个数
    277f:    75 d2                    jne    2753 <phase_6+0x79>

    #至此，rsp中按照输入的数字串的顺序索引nodes，按序存好了
    2781:    48 8b 5c 24 20           mov    0x20(%rsp),%rbx #0x20的第一个相对位置
取出no.1的给rbx
    2786:    48 8b 44 24 28           mov    0x28(%rsp),%rax #0x20的第二个相对位置
取出no.2的给rax
    278b:    48 89 43 08              mov    %rax,0x8(%rbx) #用no.2覆盖第二个node地
址上的东西
    278f:    48 8b 54 24 30           mov    0x30(%rsp),%rdx  #把no.3给rdx
    2794:    48 89 50 08              mov    %rdx,0x8(%rax) #no.3给覆盖第三个的
    2798:    48 8b 44 24 38           mov    0x38(%rsp),%rax #取no.4
    279d:    48 89 42 08              mov    %rax,0x8(%rdx) #覆盖第四个
    27a1:    48 8b 54 24 40           mov    0x40(%rsp),%rdx #取no.5
    27a6:    48 89 50 08              mov    %rdx,0x8(%rax) #覆盖第五个
    27aa:    48 8b 44 24 48           mov    0x48(%rsp),%rax #取no.6
    27af:    48 89 42 08              mov    %rax,0x8(%rdx) #覆盖第六个
    27b3:    48 c7 40 08 00 00 00     movq   $0x0,0x8(%rax)
    27ba:    00
    ##上面更改结点顺序了

    27bb:    bd 05 00 00 00           mov    $0x5,%ebp
    27c0:    eb 09                    jmp    27cb <phase_6+0xf1>

    27c2:    48 8b 5b 08              mov    0x8(%rbx),%rbx #下一个数取出来
    27c6:    83 ed 01                 sub    $0x1,%ebp #必须检查完才退
    27c9:    74 11                    je     27dc <phase_6+0x102>

    27cb:    48 8b 43 08              mov    0x8(%rbx),%rax #rbx仍是rsp中的起始
    27cf:    8b 00                    mov    (%rax),%eax #
    27d1:    39 03                    cmp    %eax,(%rbx) #小于等于下一个数则回，大于
就不行了，因此要我们升序
    27d3:    7e ed                    jle    27c2 <phase_6+0xe8>

    27d5:    e8 3f 04 00 00           callq  2c19 <explode_bomb>
    27da:    eb e6                    jmp    27c2 <phase_6+0xe8>、

    27dc:    48 8b 44 24 58           mov    0x58(%rsp),%rax
    27e1:    64 48 2b 04 25 28 00     sub    %fs:0x28,%rax
    27e8:    00 00
    27ea:    75 0d                    jne    27f9 <phase_6+0x11f>
    27ec:    48 83 c4 60              add    $0x60,%rsp
    27f0:    5b                       pop    %rbx
    27f1:    5d                       pop    %rbp
    27f2:    41 5c                    pop    %r12
    27f4:    41 5d                    pop    %r13
    27f6:    41 5e                    pop    %r14
    27f8:    c3                       retq
```

```
    27f9:    e8 a2 f8 ff ff           callq  20a0 <__stack_chk_fail@plt>
```

代码很长，具体分析写在边上的注释中了。前两个循环对输入的6个数字做了一些检查，保证它们在1到6的范围内；实际上它们是作为对地址传入`%rdx`的nodes的索引，以合适的顺序使得索引到的nodes里的数值升序排列，`2 6 5 4 1 3`即为所求，索引到的数值从264到991升序排列。

secret phase

检查反汇编得到的代码，发现有`<secret_phase>`函数，怎么调用它呢？我们检查一些main函数，发现没有直接的接口，在vscode里对.s文件查找了一下发现，原来在`<phase_defused>`可以调用，但需要满足一些要求，即在其之前的`<strings_not_equal>`需要判断字符串相等；用gdb检查一下发现，与我们第三个phase的输入相关，所需字符串为`Testify`，添加到phase3的input后顺利调出secret phase

```
//#(gdb) x/30d $rdi
//0x55555555cae0 <t0>:    0       0       0       0       0       0       0
0
//0x55555555cae8 <t0+8>:  -96     60      86      85      85      85      0
0
//rsi即输入
00000000000027fe <fun7>:
    27fe:    55                       push   %rbp
    27ff:    53                       push   %rbx
    2800:    48 83 ec 08              sub    $0x8,%rsp
    2804:    48 89 fb                 mov    %rdi,%rbx #rbx赋值为t0地址
    2807:    48 89 f5                 mov    %rsi,%rbp #输入值存到rbp
    280a:    48 85 ff                 test   %rdi,%rdi #不能为NULL
    280d:    74 2b                    je     283a <fun7+0x3c>
    280f:    0f b6 55 00              movzbl 0x0(%rbp),%edx #输入的又给了edx
    2813:    84 d2                    test   %dl,%dl #edx的最后8位是0，则递归
    2815:    74 2a                    je     2841 <fun7+0x43>
    2817:    80 fa 61                 cmp    $0x61,%dl
    281a:    74 29                    je     2845 <fun7+0x47> #edx最后8位是61，则
jump，edx清零
    281c:    0f be d2                 movsbl %dl,%edx #只要最后八位，符号拓展
    281f:    83 ea 61                 sub    $0x61,%edx #减去61
    2822:    b8 01 00 00 00           mov    $0x1,%eax

    2827:    39 d0                    cmp    %edx,%eax #eax要一直加到等于edx
    2829:    74 1f                    je     284a <fun7+0x4c> #等了跳出循环
    282b:    83 c0 01                 add    $0x1,%eax
    282e:    83 f8 1a                 cmp    $0x1a,%eax #加1后的eax不能等于0x1a即
26！可见输入的不能大于61+26然后继续循环
    2831:    75 f4                    jne    2827 <fun7+0x29>


    2833:    e8 e1 03 00 00           callq  2c19 <explode_bomb>
    2838:    eb 21                    jmp    285b <fun7+0x5d>
    283a:    e8 da 03 00 00           callq  2c19 <explode_bomb>
    283f:    eb ce                    jmp    280f <fun7+0x11>
```

```
    2841:    8b 03                   mov    (%rbx),%eax
    2843:    eb 16                   jmp    285b <fun7+0x5d>

    2845:    ba 00 00 00 00          mov    $0x0,%edx

    284a:    48 8d 75 01             lea    0x1(%rbp),%rsi #输入的+1给了rsi
    284e:    48 63 d2                movslq %edx,%rdx #减去61的东西还给rdx
    2851:    48 8b 7c d3 08          mov    0x8(%rbx,%rdx,8),%rdi #从t0取出东西
    2856:    e8 a3 ff ff ff          callq  27fe <fun7>
    285b:    48 83 c4 08             add    $0x8,%rsp
    285f:    5b                      pop    %rbx
    2860:    5d                      pop    %rbp
    2861:    c3                      retq


0000000000002862 <secret_phase>:
    2862:   48 83 ec 18             sub    $0x18,%rsp
    2866:   c7 44 24 0c 0e 00 00    movl   $0xe,0xc(%rsp)
    286d:   00
    286e:   e8 a7 04 00 00          callq  2d1a <read_line>
    2873:   48 89 c6                mov    %rax,%rsi #输入的东西
    2876:   48 8d 3d 63 62 00 00    lea    0x6263(%rip),%rdi        # 8ae0 <t0>
    //#(gdb) x/30d $rdi
    //0x55555555cae0 <t0>:    0       0       0       0       0       0       0
0
    //0x55555555cae8 <t0+8>:  -96     60      86      85      85      85      0
0
    287d:   e8 7c ff ff ff          callq  27fe <fun7>
    2882:   8b 54 24 0c             mov    0xc(%rsp),%edx
    2886:   39 c2                   cmp    %eax,%edx
    2888:   75 16                   jne    28a0 <secret_phase+0x3e>
    288a:   48 8d 3d 0f 19 00 00    lea    0x190f(%rip),%rdi        # 41a0
<_IO_stdin_used+0x1a0>
    2891:   e8 da f7 ff ff          callq  2070 <puts@plt>
    2896:   e8 b9 05 00 00          callq  2e54 <phase_defused>
    289b:   48 83 c4 18             add    $0x18,%rsp
    289f:   c3                      retq
    28a0:   e8 74 03 00 00          callq  2c19 <explode_bomb>
    28a5:   eb e3                   jmp    288a <secret_phase+0x28>
```

通过

```
(gdb) set logging file tree.txt

(gdb) set logging on
```

我们试图把输出重定向到txt文件中。注释中t0的地址为0x0000000000008ae0，通过反汇编得到的.s文件中数据段地址的插值

发现是树的结构，第一个是字符，后面是0-25共26叉的子节点。搜出0xe，回溯找到parent，对字母a偏移输出即可。