

# bomblab实验报告

罗思佳2021201679

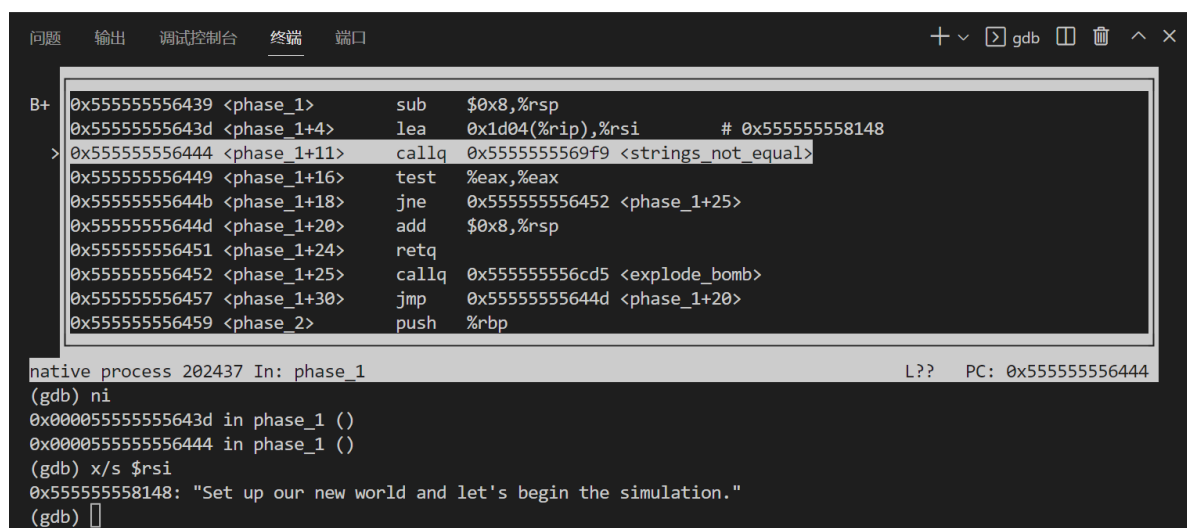
## phase\_1

### 理解思路:

分析phase\_1的代码可知, 函数先将 `0x1d04(%rip)` 的地址复制给%rsi, 然后调用了string\_not\_equal函数, 进行字符串的比较, 因为函数的第二个参数是放在%rsi里的, 可以推知这里应该是将输入的字符串与指定的字符串进行比较, 然后 `test %eax,%eax` 是在看函数返回值%eax是否为0, 如果不为0, 说明字符串不相等, 调用explode\_bomb函数爆炸, 因此只需要得到%rsi存的地址里的字符串, 输入该字符串即可成功。

### 解决方法:

进入gdb调试模式, 在phase\_1设断点, ni到调用string\_not\_euqal函数那一行, 输入命令 `x/s $rsi` 获取%rsi存的地址的内容, 如下图所示, 答案是"Set up our new world and let's begin the simulation."



```

B+ 0x55555556439 <phase_1>      sub    $0x8,%rsp
0x5555555643d <phase_1+4>      lea    0x1d04(%rip),%rsi      # 0x55555558148
> 0x55555556444 <phase_1+11>   callq  0x555555569f9 <strings_not_equal>
0x55555556449 <phase_1+16>      test   %eax,%eax
0x5555555644b <phase_1+18>      jne    0x55555556452 <phase_1+25>
0x5555555644d <phase_1+20>      add    $0x8,%rsp
0x55555556451 <phase_1+24>      retq
0x55555556452 <phase_1+25>      callq  0x55555556cd5 <explode_bomb>
0x55555556457 <phase_1+30>      jmp    0x5555555644d <phase_1+20>
0x55555556459 <phase_2>      push   %rbp

native process 202437 In: phase_1                                L??  PC: 0x55555556444
(gdb) ni
0x00005555555643d in phase_1 ()
0x000055555556444 in phase_1 ()
(gdb) x/s $rsi
0x55555558148: "Set up our new world and let's begin the simulation."
(gdb) 
```

## phase\_2

### 理解思路:

```

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 1e 09 00 00        callq  2d95 <read_six_numbers>
2477: 83 3c 24 01          cmpb   $0x1,(%rsp)
247b: 75 0a                jne     2487 <phase_2+0x2e>
247d: 48 89 e3              mov     %rsp,%rbx
2480: 48 8d 6c 24 14        lea     0x14(%rsp),%rbp
2485: eb 10                jmp     2497 <phase_2+0x3e>
```

首先大致分析一下汇编代码，由指令 `callq 2d95 <read_six_numbers>` 可知，需要输入六个数。下面的指令是在将栈顶元素和1进行比较，如果不相等就会跳到爆炸函数执行的地方，所以输入的第一个数一定是1。之后的指令 `mov %rsp,%rbx` 是把栈顶元素的地址复制到了%rbx里，然后移动栈底指针，`jmp`到2497行

```
248e: 48 83 c3 04      add    $0x4,%rbx
2492: 48 39 eb         cmp    %rbp,%rbx
2495: 74 15           je     24ac <phase_2+0x53>
2497: 8b 05 87 3c 00 00 mov    0x3c87(%rip),%eax    # 6124
<mul.2>
249d: 0f af 03       imul   (%rbx),%eax
24a0: 39 43 04       cmp    %eax,0x4(%rbx)
24a3: 74 e9         je     248e <phase_2+0x35>
24a5: e8 2b 08 00 00 callq  2cd5 <explode_bomb>
24aa: eb e2         jmp    248e <phase_2+0x35>
```

2497行，是将0x3c87(%rip)的内容拷贝到%eax，猜想这应该又是某个固定的数，下面一行指令是在进行乘法操作，将%rbx存的地址里的数和%eax相乘，然后比较%eax和0x4(%rbx)是否相等，0x4(%rbx)代表(%rbx)的下一个数，也就是输入的下一个数（一个数占4个字节），不等则爆炸，相等则跳到248e行，所以248e行到24aa行之间是一个循环。248e行是将%rbx加4，也就是移到下一个，看是否到了栈底，如果到了就结束循环。所以这个循环应该是在检查是不是等比数列，公比就是0x3c87(%rip)存的数。

### 解决方法

在gdb模式下，打印0x3c87(%rip)地址的值，输出5，因此公比是5，因为首项是1，所以答案是“1 5 25 125 625 3125”

```
(gdb) x/d 0x55555555a124
0x55555555a124 <mul.2>: 5
```

## phase\_3

理解思路：

```
24d5: 48 89 44 24 18      mov    %rax,0x18(%rsp)
24da: 31 c0              xor    %eax,%eax
24dc: 48 8d 4c 24 0f      lea    0xf(%rsp),%rcx
24e1: 48 8d 54 24 10      lea    0x10(%rsp),%rdx
24e6: 4c 8d 44 24 14      lea    0x14(%rsp),%r8
24eb: 48 8d 35 b4 1c 00 00 lea    0x1cb4(%rip),%rsi #
41a6<_IO_stdin_used+0x1a6>
24f2: e8 59 fc ff ff      callq  2150 <__isoc99_sscanf@plt>
24f7: 83 f8 02          cmp    $0x2,%eax
24fa: 7e 29            jle    2525 <phase_3+0x5d>
```

首先分析一下代码，前面的3个lea操作是将3个地址分别赋给%rcx、%rdx、%r8，然后将0x1cb4(%rip)的地址赋给%rsi，打印出来是“%d %c %d”，说明输入的是“数 字符 数”。

```
(gdb) x/s $rsi
0x5555555581a6: "%d %c %d"
```

下面的指令 `cmp $0x2,%eax` 说明输入要大于等于2个，小于的话就会爆炸。

```
24fc: 8b 05 1e 3c 00 00    mov     0x3c1e(%rip),%eax    # 6120
<mask.1>
2502: 30 44 24 0f         xor     %al,0xf(%rsp)
2506: 83 7c 24 10 07      cmpl    $0x7,0x10(%rsp)
250b: 0f 87 0c 01 00 00    ja      261d <phase_3+0x155>
2511: 8b 44 24 10         mov     0x10(%rsp),%eax
2515: 48 8d 15 a4 1c 00 00    lea     0x1ca4(%rip),%rdx #
41c0<_IO_stdin_used+0x1c0>
251c: 48 63 04 82         movslq  (%rdx,%rax,4),%rax
2520: 48 01 d0            add     %rdx,%rax
2523: ff e0              jmpq    *%rax
```

然后把0x3c1e(%rip)里的值复制给%eax，打印出来是32（暂时不知道有什么用）。

下面一行是异或操作，执行前先打印0xf(%rsp)的内容，是我输入的第二个字符“L”，执行后再打印，就变成了小写字母“l”，推测是进行了一个大写转小写的转换。

```
(gdb) x/c 0x7fffffffdddf
0x7fffffffdddf: 76 'L'
(gdb) ni
0x0000555555556506 in phase_3 ()
(gdb) x/c 0x7fffffffdddf
0x7fffffffdddf: 108 'l'
```

下面一行`cmpl`说明输入的第一个数要小于等于7，否则会爆炸。然后把输入的第一个数放到了%eax里，再把0x1ca4(%rip)的地址赋给%rdx。根据下面的 `jmpq *%rax` 以及一长串相似的代码段可推知这是switch语句，根据输入的第一个数决定跳到哪一个case。这里我输入的是1，所以跳到了下图的部分，先将0x6c复制给%eax，0x6c是108，对应小写字母“l”，然后下面的`cmpl`比较输入的第三个数是不是和0x149相等，相等就跳出switch，否则爆炸，因此当第一个数为1时，第三个数必须是0x149也就是329。输入其他的数也是同理，找到要比较的是哪个数。

```
0x55555555654e <phase_3+134>    mov     $0x6c,%eax
0x555555556553 <phase_3+139>    cmpl    $0x149,0x14(%rsp)
0x55555555655b <phase_3+147>    je      0x555555556627 <phase_3+351>
0x555555556561 <phase_3+153>    callq   0x555555556cd5 <explode_bomb>
```

跳出switch后，接着比较%al和0xf(%rsp)，%al存的是之前的108，0xf(%rsp)是输入的字符的小写，108是小写字母“l”的ASCII码，因此输入的字符应该是大写的“L”，所以其中一种答案是“1 L 329”。

### 解决方法：

理清思路后解决方法就比较好说了，有8种答案，“0 W 0x344” “1 L 0x149” “2 O 0xa3” “3 N 0x222” “4 Q 0x1bd” “5 G 0x109” “6 U 0x118” “7 l 0x122”

## phase\_4

### 理解思路:

阅读前几行代码时, 又看到了 `lea 0x1f23(%rip),%rsi` 这种操作, 不妨先打印里面的内容看看, 根据 `"%d %d"` 可知是要输入两个数, 而且下面的 `cmp $0x2,%eax` 也证明了这点。

```
(gdb) x/s 0x5555555595d1
0x5555555595d1: "%d %d"
```

下面的 `cmpl $0xe, (%rsp)` 指令是在比较栈顶元素和0xe的大小, 如果大于0xe就会爆炸, 所以**第一个数要小于等于14**。

jbe跳转之后, 执行了3个mov操作,

```
26c3:  ba 0e 00 00 00      mov    $0xe,%edx
26c8:  be 00 00 00 00      mov    $0x0,%esi
26cd:  8b 3c 24             mov    (%rsp),%edi
```

结合下面一行 `callq 264e <func4>` 可知是在准备函数的参数, 第一个参数是(`%rsp`)也就是栈顶元素的值, 第二个参数是0, 第三个参数是14。

因为func4是个递归, 比较复杂, 所以我先不进入func4, 继续看phase\_4下面的代码, 看有没有什么提示

```
26d0:  e8 79 ff ff ff      callq 264e <func4>
26d5:  8b 4c 24 04          mov    0x4(%rsp),%ecx
26d9:  8d 51 f5             lea    -0xb(%rcx),%edx
26dc:  89 54 24 04          mov    %edx,0x4(%rsp)
26e0:  83 fa 05             cmp    $0x5,%edx
26e3:  75 05               jne    26ea <phase_4+0x5f>
26e5:  83 f8 05             cmp    $0x5,%eax
26e8:  74 05               je     26ef <phase_4+0x64>
26ea:  e8 e6 05 00 00      callq 2cd5 <explode_bomb>
```

func4调用后, 先是将0x4(%rsp)即输入的第二个数赋给%ecx, 然后用lea指令将%rcx减0xb即11再把结果赋给%edx, 比较%edx和5的大小, 不等就爆炸, 所以**输入的第二个数是16** (5+11), 下面又比较了%eax和5的大小, 推知**func4返回值必须是5**。

再去看看func4, 因为直接Fenix汇编代码比较难懂, 所以我尝试着根据汇编代码写了c语言的代码

```
int fun(int x, int a, int b)
{
    int c = (b - a) >> 31;
    int res = ((b - a) + c) >> 1;
    c = res + a;
    if (c == x)
        return 0;
    if (c > x)
    {
        res = fun(x, a, c - 1);
        return res * 2;
    }
    else
    {
```

```

        res = fun(x, c + 1, b);
        return res * 2 + 1;
    }

}

```

看起来貌似是二分法，对这个函数的作用还不是很清楚，但不妨碍得出正确答案，因为输入的第一个数范围在0到14之间，只需要得到每个数对应的返回值是多少，找到返回值为5的即可。

### 解决方法：

写一个简单的测试程序，将0到14的数都试一遍，看哪些数对应的函数返回值是5

```

#include <stdio.h>

int fun(int x, int a, int b)
{
    int c = (b - a) >> 31;
    int res = ((b - a) + c) >> 1;
    c = res + a;
    if (c == x)
        return 0;
    if (c > x)
    {
        res = fun(x, a, c - 1);
        return res * 2;
    }
    else
    {
        res = fun(x, c + 1, b);
        return res * 2 + 1;
    }
}

int main()
{
    for (int i = 0; i <= 14; i++)
    {
        printf("%d %d\n", i, func4(i, 0, 14));
    }
    return 0;
}

```

最终得出10对应是返回值是5，所以答案是“10 16”。

## phase\_5

### 理解思路：

```

271a: 48 89 44 24 08      mov    %rax,0x8(%rsp)
271f: 31 c0               xor     %eax,%eax
2721: e8 b6 02 00 00      callq  29dc <string_length>
2726: 83 f8 06            cmp     $0x6,%eax
2729: 75 58              jne     2783 <phase_5+0x7a>
272b: ba 00 00 00 00      mov     $0x0,%edx
2730: 48 8d 0d a9 1a 00 00 lea     0x1aa9(%rip),%rcx      # 41e0
<array.0>

```

phase\_5的前面几行代码中，`callq 29dc <string_length>` 说明输入的是一个字符串，接着 `cmp $0x6,%eax` 说明字符串包含6个字符，接下来又看到了 `lea 0x1aa9(%rip),%rcx`，旁边的注释里写了array，猜测是数组，不妨打印一下内容，发现连提示语也打出来了，所以猜测要用到是字符串应该是前面的几个字符 `maduiersnfotvbyl`

```

(gdb) x/s 0x5555555581e0
0x5555555581e0 <array.0>:      "maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"

```

接下来是一个循环，`%rdx`相当于计数器，等于6就跳出循环，`%rcx`存的是上面那一串字符串的首地址，`movzbl`和`mov`两句应该是以`%rax`作为索引取出指定字符串的字符，然后放到 `0x1(%rsp,%rdx,1)` 里，循环之后`%rsp`存的地址后面应该是存了连续的6个字符。

```

2737: 0f be 04 13      movsbl (%rbx,%rdx,1),%eax
273b: 83 c0 0f         add     $0xf,%eax
273e: 83 e0 0f         and     $0xf,%eax
2741: 0f b6 04 01      movzbl (%rcx,%rax,1),%eax
2745: 88 44 14 01      mov     %al,0x1(%rsp,%rdx,1)
2749: 48 83 c2 01      add     $0x1,%rdx
274d: 48 83 fa 06      cmp     $0x6,%rdx
2751: 75 e4           jne     2737 <phase_5+0x2e>

```

跳出循环后，程序调用了 `strings_not_equal` 函数，推测是在将新生成的字符串与指定字符串进行比较，打印一下 `0x1a4b(%rip)`，得到

```

(gdb) x/s 0x5555555581af
0x5555555581af: "flames"

```

是一个长度为6的单词，而且这六个字母前面的字符串都有，说明输入的6个字符是索引。

### 解决方法：

找到 `flames` 在前面的字符串中的索引值，为“9 15 1 0 5 7”，因为循环前有`add`和`and`操作，所以要从26个字母中找到相应的字符，它们和`0xf`相加再与上`0xf`的结果等于索引值。最终计算出来答案是 `jpbfah`。

## phase\_6

### 理解思路：

先是读入6个数字，存到栈里，然后进行两个`mov`操作，`jmp`到27f0，

```

000000000002796 <phase_6>:
2796: 41 56           push    %r14
2798: 41 55           push    %r13
279a: 41 54           push    %r12
279c: 55             push    %rbp
279d: 53             push    %rbx

```

```

279e: 48 83 ec 60      sub    $0x60,%rsp
27a2: 64 48 8b 04 25 28 00 mov    %fs:0x28,%rax
27a9: 00 00
27ab: 48 89 44 24 58      mov    %rax,0x58(%rsp)
27b0: 31 c0             xor    %eax,%eax
27b2: 49 89 e5           mov    %rsp,%r13
27b5: 4c 89 ee           mov    %r13,%rsi
27b8: e8 d8 05 00 00      callq  2d95 <read_six_numbers>    # 读6个数
27bd: 41 be 01 00 00 00    mov    $0x1,%r14d                # 1复制
给%r14d
27c3: 49 89 e4           mov    %rsp,%r12
27c6: eb 28             jmp     27f0 <phase_6+0x5a>

```

27f0到27fd说明输入的数不超过6，然后看%r14d有没有超过5，大于的话就跳出，否则将%r14复制给%rbx，跳到27d8，27cf到27df又是一个循环，是在循环比较（%rbp）后面的数有没有和它相等，相等则爆炸，说明输入的数和它后面的数是互异的，这层循环结束后，回到27e8，去检查下一个数是不是小于等于6，然后循环检查后面的数是不是和它互异，因此这是一个循环套循环，这段代码的作业用是：**输入的6个数不超过6且互异。**

```

27cf: 48 83 c3 01      add    $0x1,%rbx                # 计数器，加一
27d3: 83 fb 05          cmp    $0x5,%ebx
27d6: 7f 10            jg     27e8 <phase_6+0x52>
27d8: 41 8b 04 9c      mov    (%r12,%rbx,4),%eax        # 下一个数
27dc: 39 45 00          cmp    %eax,0x0(%rbp)
27df: 75 ee           jne    27cf <phase_6+0x39>
27e1: e8 ef 04 00 00    callq  2cd5 <explode_bomb>
27e6: eb e7           jmp     27cf <phase_6+0x39>
27e8: 49 83 c6 01      add    $0x1,%r14
27ec: 49 83 c5 04      add    $0x4,%r13
27f0: 4c 89 ed         mov    %r13,%rbp
27f3: 41 8b 45 00      mov    0x0(%r13),%eax            # 把输入的第一个数复
制给%eax
27f7: 83 e8 01         sub    $0x1,%eax
27fa: 83 f8 05          cmp    $0x5,%eax                # 减一后和5比较
27fd: 77 c9           ja     27c8 <phase_6+0x32>        # 说明输入的数不大于
6
27ff: 41 83 fe 05      cmp    $0x5,%r14d
2803: 7f 05           jg     280a <phase_6+0x74>
2805: 4c 89 f3         mov    %r14,%rbx
2808: eb ce           jmp     27d8 <phase_6+0x42>

```

再看下面几行，先把0复制给%esi，然后取输入的第一个数复制给%ecx，下面又是熟悉的lea操作，猜测又是要用到某个定义好的常量

```

280a: be 00 00 00 00    mov    $0x0,%esi
280f: 8b 0c b4          mov    (%rsp,%rsi,4),%ecx
2812: b8 01 00 00 00    mov    $0x1,%eax
2817: 48 8d 15 42 db 00 00 lea     0xdb42(%rip),%rdx          # 10360
<node1>

```

打印一下，发现输出了5个节点，并且每个节点的第二个数是序号，第三个和第四个拼起来是下一个节点的地址，猜测第一个数是node的值，因此这是一个链表，



```
(gdb) x/30wx 0x555555564360
0x555555564360 <node1>: 0x000003d9      0x00000001      0x555564370      0x00005555
0x555555564370 <node2>: 0x00000370      0x00000002      0x555564380      0x00005555
0x555555564380 <node3>: 0x00000245      0x00000003      0x555564390      0x00005555
0x555555564390 <node4>: 0x000001b6      0x00000004      0x5555643a0      0x00005555
0x5555555643a0 <node5>: 0x000001d6      0x00000005      0x5555baa0      0x00005555
0x5555555643b0: 0x00000000      0x00000000      0x00000000      0x00000000
0x5555555643c0 <host_table>: 0x5555862c      0x00005555      0x55558631      0x00005555
0x5555555643d0 <host_table+16>: 0x00000000      0x00000000
```

但是按理说应该有6个节点，打出来只有5个，好在可以根据node5里存的地址找到node6，因此六个节点的值依次是0x3d9 0x370 0x245 0x1b6 0x1d6 0x1c0。

```
(gdb) x/4wx 0x55555555baa0
0x55555555baa0 <node6>: 0x000001c0      0x00000006      0x00000000      0x00000000
```

之后的一段代码比较复杂，不容易看懂，于是我继续往后看，

下面这段代码是在循环比较当前节点和它后一个节点，如果大于就爆炸，说明链表是**升序排列**，因此之前的一番看不太懂的操作是将链表按照输入的顺序重新排列了，因此需要将链表升序排列，然后一次输入每个节点的序号

```
2877:  bd 05 00 00 00      mov     $0x5,%ebp
287c:  eb 09              jmp     2887 <phase_6+0xf1>
287e:  48 8b 5b 08        mov     0x8(%rbx),%rbx
2882:  83 ed 01          sub     $0x1,%ebp
2885:  74 11             je      2898 <phase_6+0x102>
2887:  48 8b 43 08        mov     0x8(%rbx),%rax
288b:  8b 00             mov     (%rax),%eax
288d:  39 03             cmp     %eax,(%rbx)  #比较当前节点值和它下一个节点的值
288f:  7e ed             jle     287e <phase_6+0xe8>
2891:  e8 3f 04 00 00     callq   2cd5 <explode_bomb>
2896:  eb e6             jmp     287e <phase_6+0xe8>
2898:  48 8b 44 24 58     mov     0x58(%rsp),%rax
289d:  64 48 2b 04 25 80  sub     %fs:0x28,%rax
```

### 解决方法:

由以上分析可知，节点1-6按升序排列，对应的序号为 4 6 5 3 2 1，这就是所需的答案。

## secret\_phase

### 触发的方法:

6个密码全部输入后并没有出现隐藏关，而是显示已拆除全部炸弹，因此需要找到进入的方法。在 phase\_defused函数中，找到了secret\_phase的调用，因此是在这个函数里触发的隐藏关。

分析一下phase\_defused的代码，cmpl指令比较0xd93d(%rip)和6的大小关系，等于6就跳转到2f5c，说明要先通过前6个phase

```
2f34:  83 3d 3d d9 00 00 06  cmpl    $0x6,0xd93d(%rip)  # 10878
<num_input_strings>
2f3b:  74 1f             je      2f5c <phase_defused+0x4c>
```

跳到2f5c后，又看到0x16a9(%rip),%rsi 这种操作，于是打印一下地址的内容，为 "%d %d %s"，再打印一下下面一行%rdi存的地址的内容，为phase\_4输入的答案"10 16"，结合下面的 `cmpl $0x3,%eax` 推知phase\_4要在数字后面再输入一个字符串才能触发隐藏关，



```

2f5c: 48 8d 4c 24 0c      lea    0xc(%rsp),%rcx
2f61: 48 8d 54 24 08      lea    0x8(%rsp),%rdx
2f66: 4c 8d 44 24 10      lea    0x10(%rsp),%r8
2f6b: 48 8d 35 a9 16 00 00 lea    0x16a9(%rip),%rsi # 461b
<array.0+0x43b>
2f72: 48 8d 3d 6f da 00 00 lea    0xda6f(%rip),%rdi# 109e8
<input_strings+0x168>
2f79: b8 00 00 00 00      mov     $0x0,%eax
2f7e: e8 cd f1 ff ff      callq  2150 <__isoc99_sscanf@plt>
2f83: 83 f8 03            cmp     $0x3,%eax
2f86: 74 1a              je      2fa2 <phase_defused+0x92>

```

先随便输入一个字符串，gdb调试到2fa2行，有字符串比较的操作，打印%rsi存的地址里的内容，如下图，说明要输入的字符串就是 `Testify`，隐藏关触发的方法就是，在phase\_4时输入 `10 16 Testify`。

```

2fa2: 48 8d 7c 24 10      lea    0x10(%rsp),%rdi
2fa7: 48 8d 35 76 16 00 00 lea    0x1676(%rip),%rsi # 4624
<array.0+0x444>
2fae: e8 46 fa ff ff      callq  29f9 <strings_not_equal>
2fb3: 85 c0              test    %eax,%eax
2fb5: 75 d1              jne     2f88 <phase_defused+0x78>

```

```

(gdb) x/s 0x555555558624
0x555555558624: "Testify"

```

### 找答案的过程：

这里又调用了函数fun7，因为fun7函数用到了递归，不好理解，所以先往后看，`mov 0xc(%rsp),%edx` 把0xc(%rsp)里的值也就是0x11复制给%edx，然后比较fun7返回值和%edx是否相等，因此**fun7的返回值必须是0x11即17**。再关注一下fun7的参数，第二个参数也就是%rsi存的是read\_line函数的返回值，打印出来是这一关输入的字符串，因此**第二个参数是指向输入字符串的指针**，%rdi存的是0x51a7(%rip)地址，打印一下，发现是个结点t0，因此**第一个参数是指向t0的指针**。

```

00000000000291e <secret_phase>:
291e: 48 83 ec 18      sub     $0x18,%rsp
2922: c7 44 24 0c 11 00 00 movl    $0x11,0xc(%rsp) # 把0x11复制给
0xc(%rsp)
2929: 00
292a: e8 a7 04 00 00      callq  2dd6 <read_line> # 读入
292f: 48 89 c6          mov     %rax,%rsi
2932: 48 8d 3d a7 51 00 00 lea     0x51a7(%rip),%rdi # 7ae0 <t0>
2939: e8 7c ff ff ff      callq  28ba <fun7>
293e: 8b 54 24 0c          mov     0xc(%rsp),%edx
2942: 39 c2            cmp     %eax,%edx #比较fun7返回值和%edx
2944: 75 16            jne     295c <secret_phase+0x3e>
2946: 48 8d 3d 33 18 00 00 lea     0x1833(%rip),%rdi# 4180
<_IO_stdin_used+0x180>
294d: e8 1e f7 ff ff      callq  2070 <puts@plt>
2952: e8 b9 05 00 00      callq  2f10 <phase_defused>
2957: 48 83 c4 18          add     $0x18,%rsp
295b: c3              retq
295c: e8 74 03 00 00      callq  2cd5 <explode_bomb>
2961: eb e3            jmp     2946 <secret_phase+0x28>

```

```

native process 216469 In: secret phase
0x55555555bae0 <t0>: 0x00000000 0x00000000 0x55562ca0 0x00005555
0x55555555baf0 <t0+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x55555555bb00 <t0+32>: 0x55561a40 0x00005555 0x00000000 0x00000000
0x55555555bb10 <t0+48>: 0x55561180 0x00005555 0x55560700 0x00005555
0x55555555bb20 <t0+64>: 0x5555fba0 0x00005555 0x00000000 0x00000000
0x55555555bb30 <t0+80>: 0x00000000 0x00000000 0x00000000 0x00000000
0x55555555bb40 <t0+96>: 0x5555f120 0x00005555 0x00000000 0x00000000
0x55555555bb50 <t0+112>: 0x00000000 0x00000000 0x00000000 0x00000000
0x55555555bb60 <t0+128>: 0x5555e320 0x00005555 0x00000000 0x00000000
0x55555555bb70 <t0+144>: 0x5555dc20 0x00005555 0x5555cd40 0x00005555
---Type <return> to continue, or q <return> to quit---

```

再看fun7,又是难懂的递归,因此我试着根据汇编代码把c-like的代码写出来

```

int fun7(Node* node, int *p)
{
    if(node == NULL) explode_bomb();//节点为空,爆炸
    int num = *p;
    if(num == 0)//空字符,结束递归
    {
        return *node;
    }
    if(num == 0x61)//等于a
    {
        p++;
        node = 0x8(node, 0, 8);
        fun7(node, p);
    }
    num -= 0x61;
    int b = 1;
    while(1)
    {
        if(num == b)
        {
            p++;
            node = 0x8(node, b, 8);
            fun7(node, p);
        }
        b++;
        if(b == 26) explode_bomb();//超过了26个字母
    }
}

```

根据以上的代码可知,输入的字符串作为索引,找下一个node的地址,下一个node的地址不能为0,否则会爆炸;当字符串用完时,返回当前也就是最后一个node的值。因此需要输入合适的字符串使函数最后返回0x11,即最后的节点的值是0x11。再观察一下t0的结构,8个字节为一组的话,一共有28组,第一组存放值,加上26个字母对应26个地址,刚好是27组,最后一组空着,印证了之前的猜测。

```

0x55555555bae0 <t0>: 0x00000000 0x00000000 0x55562ca0 0x00005555
0x55555555baf0 <t0+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x55555555bb00 <t0+32>: 0x55561a40 0x00005555 0x00000000 0x00000000
0x55555555bb10 <t0+48>: 0x55561180 0x00005555 0x55560700 0x00005555
0x55555555bb20 <t0+64>: 0x5555fba0 0x00005555 0x00000000 0x00000000
0x55555555bb30 <t0+80>: 0x00000000 0x00000000 0x00000000 0x00000000
0x55555555bb40 <t0+96>: 0x5555f120 0x00005555 0x00000000 0x00000000
0x55555555bb50 <t0+112>: 0x00000000 0x00000000 0x00000000 0x00000000
0x55555555bb60 <t0+128>: 0x5555e320 0x00005555 0x00000000 0x00000000
0x55555555bb70 <t0+144>: 0x5555dc20 0x00005555 0x5555cd40 0x00005555

```

```
0x5555555bb80 <t0+160>: 0x5555bbc0 0x00005555 0x00000000 0x00000000
0x5555555bb90 <t0+176>: 0x00000000 0x00000000 0x00000000 0x00000000
0x5555555bba0 <t0+192>: 0x00000000 0x00000000 0x00000000 0x00000000
0x5555555bbb0 <t0+208>: 0x00000000 0x00000000 0x00000000 0x00000000
0x5555555bbc0 <t69>: 0x00000000 0x00000045 0x5555c480 0x00005555
```

所以接下来要做的就是找到值为0x11（第一个四字节）的节点序号，然后倒推回t0，找到从t0节点到该节点的路径。

因为节点在内存里的顺序没有规律，所以索性将t0前后几百行地址的内容都打出来，将输出重定向到一个txt文件里，然后查找值为0x11的节点,为 t112,

```
0x5555555b480 <t112>: 0x00000011 0x00000070 0x00000000 0x00000000`
```

然后根据t112的地址查找上一个节点, 为t111

```
0x555555562bc0 <t111>: 0x00000000 0x0000006f 0x00000000 0x00000000
.....
0x555555562c30 <t111+112>: 0x5555b480 0x00005555 0x00000000 0x00000000
```

接着是t110

```
0x555555562ae0 <t110>: 0x00000000 0x0000006e 0x00000000 0x00000000
.....
0x555555562b50 <t110+112>: 0x00000000 0x00000000 0x55562bc0 0x00005555
```

t109

```
0x555555562a00 <t109>: 0x00000000 0x0000006d 0x00000000 0x00000000
.....
0x555555562a40 <t109+64>: 0x00000000 0x00000000 0x55562ae0 0x00005555
```

t108

```
0x555555562920 <t108>: 0x00000000 0x0000006c 0x00000000 0x00000000
.....
0x555555562980 <t108+96>: 0x55562a00 0x00005555 0x00000000 0x00000000
```

t107

```
0x555555562840 <t107>: 0x00000000 0x0000006b 0x00000000 0x00000000
.....
0x555555562860 <t107+32>: 0x00000000 0x00000000 0x55562920 0x00005555
```

t37

```
0x555555562300 <t37>: 0x00000000 0x00000025 0x00000000 0x00000000
.....
0x555555562320 <t37+32>: 0x55562840 0x00005555 0x00000000 0x00000000
```

t36

```
0x55555562220 <t36>: 0x00000000 0x00000024 0x00000000 0x00000000
.....
0x55555562290 <t36+112>: 0x55562300 0x00005555 0x00000000 0x00000000
```

t35

```
0x55555561a40 <t35>: 0x00000000 0x00000023 0x55562220 0x00005555
```

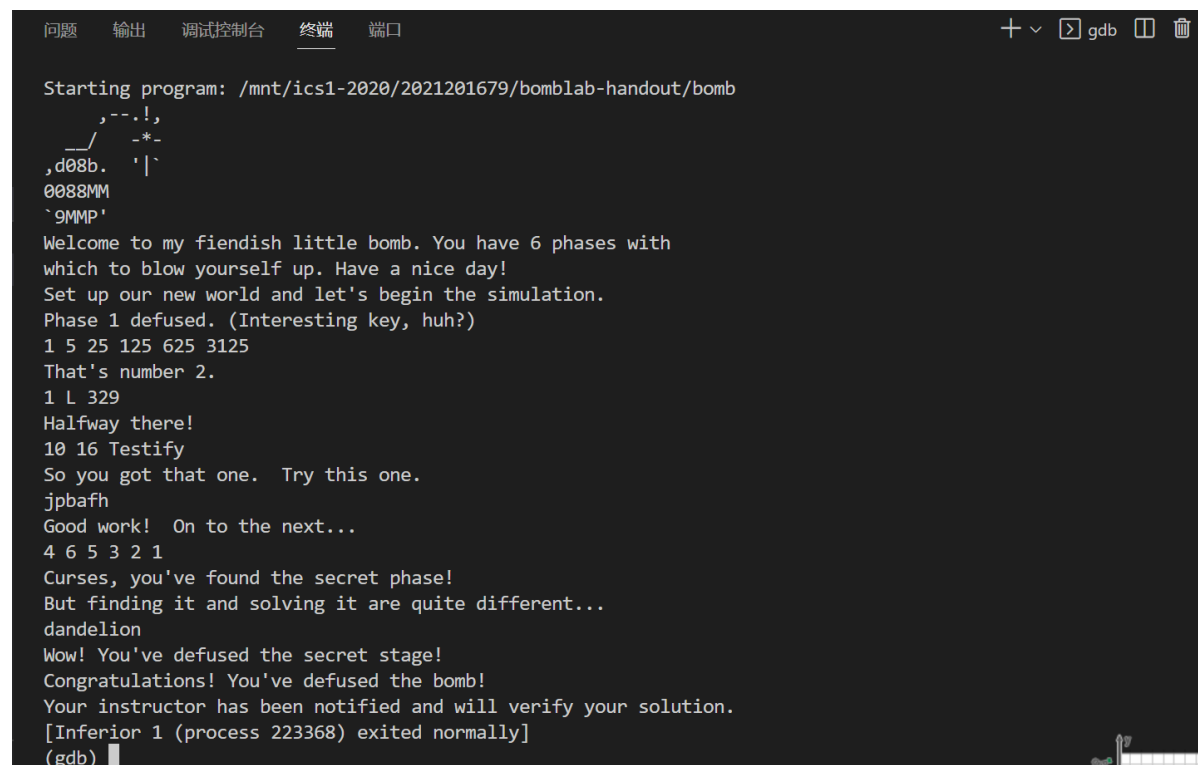
t0

```
0x5555555bae0 <t0>: 0x00000000 0x00000000 0x55562ca0 0x00005555
.....
0x5555555bb00 <t0+32>: 0x55561a40 0x00005555 0x00000000 0x00000000
```

因此顺序是 t0->t35->t36->t37->t107->t108->t109->t110->t111->t112，所以答案的索引字符串为 dandelion。

## 实验结果

通过所有关卡，如图



```
问题 输出 调试控制台 终端 端口
Starting program: /mnt/ics1-2020/2021201679/bomblab-handout/bomb
,--.!,
 _/  -*
,d08b. '|`
0088MM
`9MMP'
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Set up our new world and let's begin the simulation.
Phase 1 defused. (Interesting key, huh?)
1 5 25 125 625 3125
That's number 2.
1 L 329
Halfway there!
10 16 Testify
So you got that one. Try this one.
jpbafh
Good work! On to the next...
4 6 5 3 2 1
Curses, you've found the secret phase!
But finding it and solving it are quite different...
dandelion
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 223368) exited normally]
(gdb)
```