计算机系统——BombLab实验报告

> 丁海桐——软件2203——202226010304

一、实验介绍

1. 实验题目

“二进制炸弹”是一个由六个“阶段”组成的Linux可执行C程序。每个阶段要求学生在标准输入（stdin）中输入特定字符串。如果学生输入了预期的字符串，则该阶段被视为“解除”。否则，炸弹将通过打印“BOOM!!!”而“爆炸”。学生的任务是尽可能多地解除各个阶段。

每个炸弹阶段测试机器语言程序的不同方面：

- 第1阶段：字符串比较
- 第2阶段：循环
- 第3阶段：条件判断/开关
- 第4阶段：递归调用和堆栈规则
- 第5阶段：指针
- 第6阶段：链表/指针/结构体

阶段难度逐渐增加。还有一个“秘密阶段”，只有当在第4阶段解决方案末尾附加特定字符串时才会出现。

2. 实验目的

炸弹实验室旨在教授学生机器级程序原理以及通用调试器和逆向工程技能。

二、实验内容

首先对`bomb`文件进行反汇编，将汇编代码放入新文件`asm.txt`文件中。

```
objdump -d bomb > asm.txt
```

同时创建`ans.txt`文件，作为`bomb`的输入文件，此后执行`bomb`就可以像如下一样：

```
./bomb ans.txt
```

本次实验分为如下 7 个阶段，笔者也会按照此顺序进行分析。

- `phase_1`
- `phase_2`
- `phase_3`
- `phase_4`
- `phase_5`

- phase_6
- secret_phase

phase_1

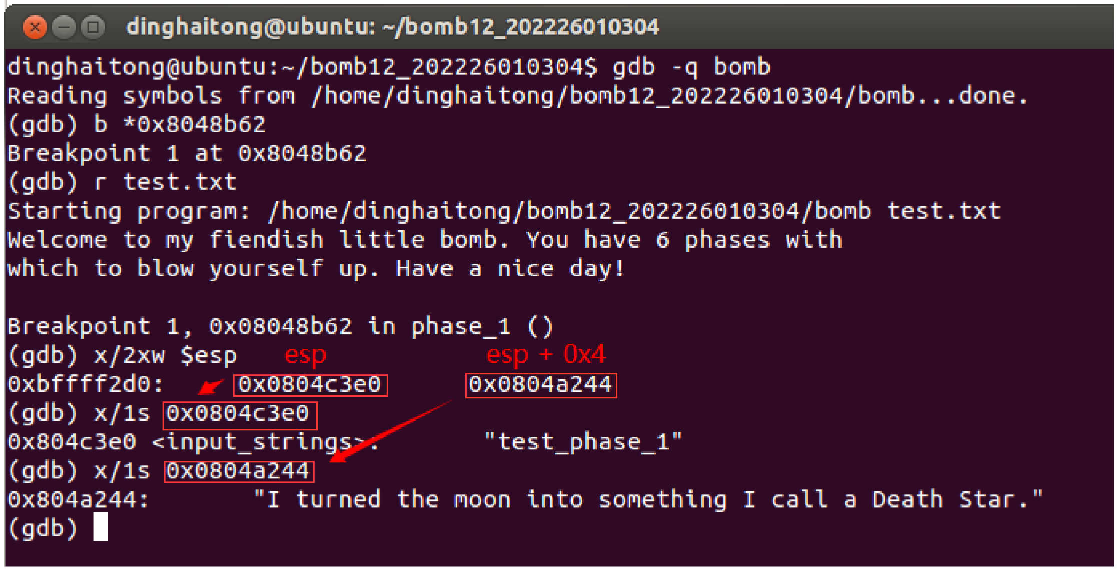
在 asm.txt 中找到 <phase_1>，汇编代码如下：

```
08048b50 <phase_1>:
 8048b50:       83 ec 1c                sub    $0x1c,%esp
 8048b53:       c7 44 24 04 44 a2 04    movl   $0x804a244,0x4(%esp)   // (0x804a244) -> 答案ans
 8048b5a:       08
 8048b5b:       8b 44 24 20             mov    0x20(%esp),%eax        // 0x20(%esp)  -> 输入
input
 8048b5f:       89 04 24                mov    %eax,(%esp)            // 这里把ans和input放到
(%esp)和0x4(%esp)是作为传递参数
 8048b62:       e8 fd 04 00 00          call   8049064 <strings_not_equal> //%eax携带的是
strings_not_equal函数的返回值
                                                                     //从函数字面意义上返回
0(false)表示不是不相同，也就是相同
 8048b67:       85 c0                   test   %eax,%eax              // ZF = %exa & %exa
 8048b69:       74 05                   je     8048b70 <phase_1+0x20> // if ZF==0 jump
 8048b6b:       e8 06 06 00 00          call   8049176 <explode_bomb>
 8048b70:       83 c4 1c                add    $0x1c,%esp             //收回栈帧
 8048b73:       c3                      ret
```

<phase_1> 中调用了函数 <strings_not_equal> 并传递了 2 个参数 (%esp) 和 0x4(%esp) 即 (0x804a244)，返回值存储在 %eax 中，若是 1 表示二者不同，反之。之后 test %eax,%eax，若返回值是 0，就收回栈帧，此阶段完成若返回值是 1 就顺序执行 <explode_bomb>。我们不想炸弹爆炸，就要保证函数返回值是 0，也就是传入的 2 个参数相同

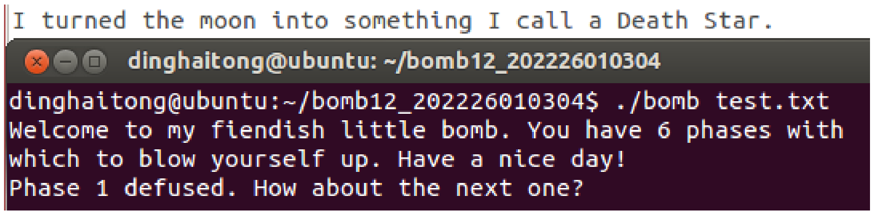
在 gdb 中我们可以找到传入的参数。可以发现传入的参数是两个地址，分别找到其指向的数据，可以发现一个是我们的输入，一个是字符串 "I turned the moon into something I call a Death Star."，所以当我们也输入它时，炸弹就不会爆炸。

test_phase_1



```
dinghaitong@ubuntu:~/bomb12_202226010304$ gdb -q bomb
Reading symbols from /home/dinghaitong/bomb12_202226010304/bomb...done.
(gdb) b *0x8048b62
Breakpoint 1 at 0x8048b62
(gdb) r test.txt
Starting program: /home/dinghaitong/bomb12_202226010304/bomb test.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!

Breakpoint 1, 0x08048b62 in phase_1 ()
(gdb) x/2xw $esp
0xbffff2d0:     0x0804c3e0      0x0804a244
(gdb) x/1s 0x0804c3e0
0x804c3e0 <input_strings>:      "test_phase_1"
(gdb) x/1s 0x0804a244
0x804a244:      "I turned the moon into something I call a Death Star."
(gdb) 
```

验证正确



```
I turned the moon into something I call a Death Star.
```

```
dinghaitong@ubuntu:~/bomb12_202226010304$ ./bomb test.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?
```

> ***the ans of phase_1 ： I turned the moon into something I call a Death Star.***

phase_2

1. `cmp %eax, %ebx`：根据 `ebx - eax` 的值，写入ZF、CF、SF、OF标志。
2. `cmpl`：对比的是无符号数，`cmp` 是有符号数。
3. `lea 0x18(%esp),%eax`：`eax = esp + 18`。**这里只是地址在计算，并不会去内存中取值，要和mov区分。**

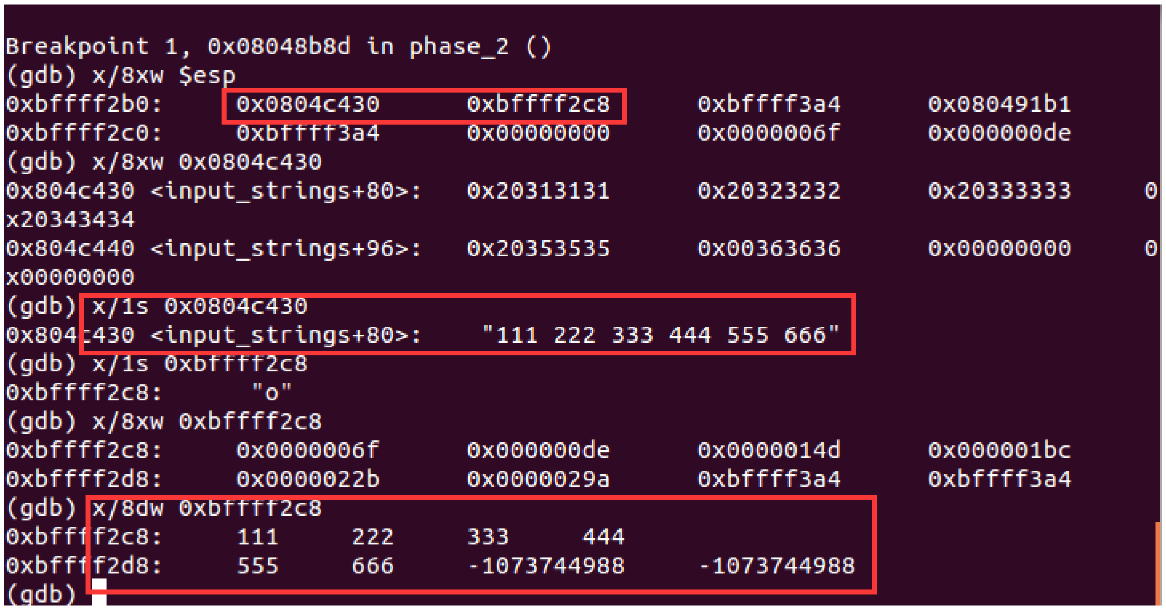
- `break *addr` 在指令addr处设置断点
- `continue` 能快速运行到下一个断点

在 `asm.txt` 中找到 `<phase_2>`，汇编代码如下：

```
08048b74 <phase_2>:
 8048b74:	56                   	push   %esi
 8048b75:	53                   	push   %ebx
 8048b76:	83 ec 34             	sub    $0x34,%esp
 8048b79:	8d 44 24 18          	lea    0x18(%esp),%eax
 8048b7d:	89 44 24 04          	mov    %eax,0x4(%esp)          // (%esp+4) = %esp + 18, 指向a[0]地址
 8048b81:	8b 44 24 40          	mov    0x40(%esp),%eax
 8048b85:	89 04 24             	mov    %eax,(%esp)             // (%esp) = (%esp + 40), 指向输入的字符串
 8048b88:	e8 1e 07 00 00       	call   80492ab <read_six_numbers>
 8048b8d:	83 7c 24 18 01       	cmpl   $0x1,0x18(%esp)         // a[0] 和 1 比较
 8048b92:	74 05                	je     8048b99 <phase_2+0x25>  // if a[0] == 1 进入下一步
 8048b94:	e8 dd 05 00 00       	call   8049176 <explode_bomb>  // if a[0] != 1 就爆炸
 8048b99:	8d 5c 24 1c          	lea    0x1c(%esp),%ebx         // %ebx = %esp + 1c, %exb = &a[1]
 8048b9d:	8d 74 24 30          	lea    0x30(%esp),%esi         // %esi = %esp + 30, %esi = &a[6], 也就是六个数结束的地址
 8048ba1:	8b 43 fc             	mov    -0x4(%ebx),%eax         // %eax = a[0] = (%esp + 18)
 8048ba4:	01 c0                	add    %eax,%eax               // %eax = 2 * a[0]
 8048ba6:	39 03                	cmp    %eax,(%ebx)             // a[1] - 2 * a[0]
 8048ba8:	74 05                	je     8048baf <phase_2+0x3b>  // if a[1] == 2*a[0] 进入下一步
 8048baa:	e8 c7 05 00 00       	call   8049176 <explode_bomb>  // if a[1] != 2*a[0] 爆炸
 8048baf:	83 c3 04             	add    $0x4,%ebx               // %ebx = %esp + 20, %exb = &a[2]
 8048bb2:       39 f3                   cmp    %esi,%ebx
 8048bb4:       75 eb                   jne    8048ba1 <phase_2+0x2d>      // if %ebx <= %esi 循
环
 8048bb6:       83 c4 34                add    $0x34,%esp                  // 收回栈帧
 8048bb9:       5b                      pop    %ebx
 8048bba:       5e                      pop    %esi
 8048bbb:       c3                      ret
```

首先看到 `<phase_2>` 调用了 `<read_six_numbers>` 函数，猜想要输入 6 个数。

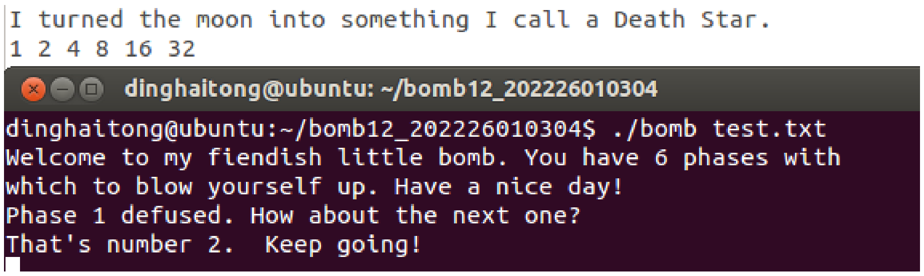
在 `0x8048b8d` 处打断点，查看传入的参数 `(%esp+4)` 和 `(%esp)` 分别是什么


```
Breakpoint 1, 0x08048b8d in phase_2 ()
(gdb) x/8xw $esp
0xbffff2b0:     0x0804c430      0xbffff2c8      0xbffff3a4      0x080491b1
0xbffff2c0:     0xbffff3a4      0x00000000      0x0000006f      0x000000de
(gdb) x/8xw 0x0804c430
0x804c430 <input_strings+80>:   0x20313131      0x20323232      0x20333333      0
x20343434
0x804c440 <input_strings+96>:   0x20353535      0x00363636      0x00000000      0
x00000000
(gdb) x/1s 0x0804c430
0x804c430 <input_strings+80>:    "111 222 333 444 555 666"
(gdb) x/1s 0xbffff2c8
0xbffff2c8:      "o"
(gdb) x/8xw 0xbffff2c8
0xbffff2c8:     0x0000006f      0x000000de      0x0000014d      0x000001bc
0xbffff2d8:     0x0000022b      0x0000029a      0xbffff3a4      0xbffff3a4
(gdb) x/8dw 0xbffff2c8
0xbffff2c8:     111     222     333     444
0xbffff2d8:     555     666     -1073744988     -1073744988
(gdb)
```

可以看到 `(%esp+4)` 存储的是输入的 6 个数字组成数组的首地址 `0xbffff2c8`，也就是 `%esp+0x18`；`(%esp)` 存储的是输入的字符串的地址，在执行完 `<read_six_numbers>` 函数，`(%esp+0x18)` 开始 24 个字节被填充。数组 a 的首地址是 `%esp+0x18`，尾地址是 `%esp+0x30`。

`a[0]` 必须为 1，不然就爆炸；`%ebx = &a[1]`，`%ebx` 必须等于 `2 * (%esp+0x18)`，也就是 `a[1] == 2 * a[0]`，`%ebx += 4`，也就是 `%ebx = &a[2]`；`%esi = &a[6]`，如果 `%ebx <= %esi` 开始下一次循环。我们可以发现输入的 6 个数字是有规律的，满足 `a[0] = 1, a[i] = 2*a[i-1]`。所以答案是 `1 2 4 8 16 32`。

验证正确。


```
I turned the moon into something I call a Death Star.
1 2 4 8 16 32
```

```
dinghaitong@ubuntu:~/bomb12_202226010304$ ./bomb test.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!
```

> ***the ans of phase_2： 1 2 4 8 16 32***

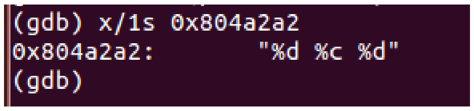
phase_3

在 `asm.txt` 中找到 `<phase_3>`，汇编代码如下：

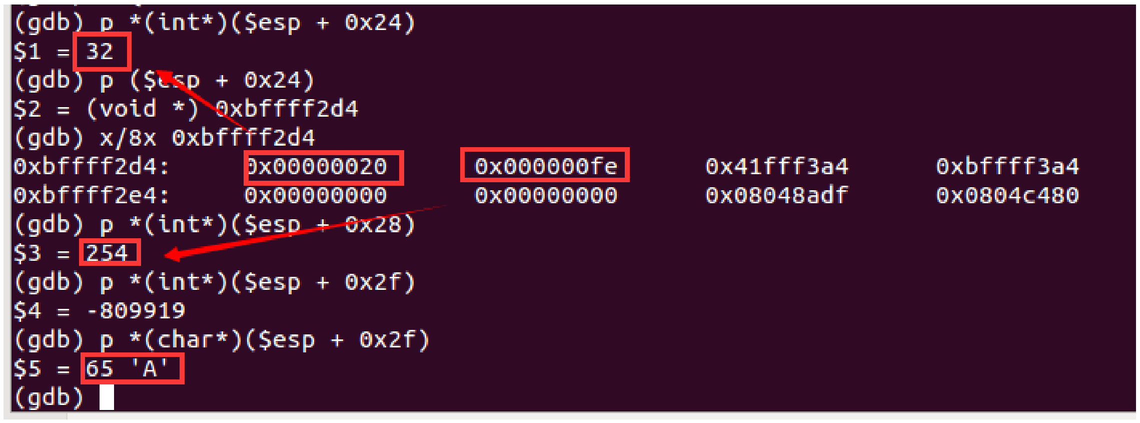
```
08048bbc <phase_3>:
 8048bbc:	83 ec 3c             	sub    $0x3c,%esp
 8048bbf:	8d 44 24 28          	lea    0x28(%esp),%eax
 8048bc3:	89 44 24 10          	mov    %eax,0x10(%esp)          // (%esp + 0x10) = 
%esp + 0x28,  (%esp + 0x28)=第3个数z
 8048bc7:	8d 44 24 2f          	lea    0x2f(%esp),%eax
 8048bcb:	89 44 24 0c          	mov    %eax,0xc(%esp)           // (%esp + 0xc) = 
%esp + 0x2f,   (%esp + 0x2f)=第2个数y
 8048bcf:	8d 44 24 24          	lea    0x24(%esp),%eax
 8048bd3:	89 44 24 08          	mov    %eax,0x8(%esp)           // (%esp + 0x8) = 
%esp + 0x24,   (%esp + 0x24)=第1个数x
 8048bd7:	c7 44 24 04 a2 a2 04 	movl   $0x804a2a2,0x4(%esp)     // (%esp + 0x4) = 
$0x804a2a2
 8048bde:	08                                                      // x/1s 0x804a2a2
= "%d %c %d"
 8048bdf:	8b 44 24 40          	mov    0x40(%esp),%eax
 8048be3:	89 04 24             	mov    %eax,(%esp)              // (%esp) = 输入
 8048be6:	e8 85 fc ff ff       	call   8048870 <__isoc99_sscanf@plt> // 用于从字符串中读
取数据并将其赋值给变量,返回变量个数
 8048beb:	83 f8 02             	cmp    $0x2,%eax                 // %eax 中存的是
函数sscanf的返回值
 8048bee:	7f 05                	jg     8048bf5 <phase_3+0x39>    // 变量个数<=2就爆
炸
 8048bf0:	e8 81 05 00 00       	call   8049176 <explode_bomb>
 8048bf5:	83 7c 24 24 07       	cmpl   $0x7,0x24(%esp)           // 无符号数比较, x
要 <= 7
 8048bfa:	0f 87 fc 00 00 00    	ja     8048cfc <phase_3+0x140>
 8048c00:	8b 44 24 24          	mov    0x24(%esp),%eax           // %eax = x
 8048c04:	ff 24 85 c0 a2 04 08 	jmp    *0x804a2c0(,%eax,4)       // jump
(0x804a2c0 + 4 * %eax)
 8048c0b:	b8 79 00 00 00       	mov    $0x79,%eax                // x = 0
 8048c10:	81 7c 24 28 08 02 00 	cmpl   $0x208,0x28(%esp)         // z = 0x208 = 
520, y = 0x79 = 121 = 'y'
 8048c17:	00 
 8048c18:	0f 84 e8 00 00 00    	je     8048d06 <phase_3+0x14a>
 8048c1e:	e8 53 05 00 00       	call   8049176 <explode_bomb>
 8048c23:	b8 79 00 00 00       	mov    $0x79,%eax
 8048c28:	e9 d9 00 00 00       	jmp    8048d06 <phase_3+0x14a>
 8048c2d:	b8 62 00 00 00       	mov    $0x62,%eax                 // x = 1
 8048c32:	81 7c 24 28 3c 02 00 	cmpl   $0x23c,0x28(%esp)          // z = 0x23c = 
572, y = 0x62 = 98 = 'b'
 8048c39:	00 
 8048c3a:	0f 84 c6 00 00 00    	je     8048d06 <phase_3+0x14a>
 8048c40:	e8 31 05 00 00       	call   8049176 <explode_bomb>
 8048c45:	b8 62 00 00 00       	mov    $0x62,%eax
 8048c4a:	e9 b7 00 00 00       	jmp    8048d06 <phase_3+0x14a>
 8048c4f:	b8 69 00 00 00       	mov    $0x69,%eax                 // x = 2
 8048c54:	81 7c 24 28 bc 02 00 	cmpl   $0x2bc,0x28(%esp)          // z = 0x2bc = 
700, y = 0x69 = 105 = 'i'
```

```
 8048c5b:	00
 8048c5c:	0f 84 a4 00 00 00    	je     8048d06 <phase_3+0x14a>
 8048c62:	e8 0f 05 00 00       	call   8049176 <explode_bomb>
 8048c67:	b8 69 00 00 00       	mov    $0x69,%eax
 8048c6c:	e9 95 00 00 00       	jmp    8048d06 <phase_3+0x14a>
 8048c71:	b8 63 00 00 00       	mov    $0x63,%eax                    // x = 3
 8048c76:	81 7c 24 28 9f 01 00 	cmpl   $0x19f,0x28(%esp)             // z = 0x19f =
415,  y = 0x63 = 99 = 'c'
 8048c7d:	00
 8048c7e:	0f 84 82 00 00 00    	je     8048d06 <phase_3+0x14a>
 8048c84:	e8 ed 04 00 00       	call   8049176 <explode_bomb>
 8048c89:	b8 63 00 00 00       	mov    $0x63,%eax
 8048c8e:	eb 76                	jmp    8048d06 <phase_3+0x14a>
 8048c90:	b8 6e 00 00 00       	mov    $0x6e,%eax                    // x = 4
 8048c95:	81 7c 24 28 26 01 00 	cmpl   $0x126,0x28(%esp)             // z = 0x126 =
294, y = 0x6e = 110 = 'n'
 8048c9c:	00
 8048c9d:	74 67                	je     8048d06 <phase_3+0x14a>
 8048c9f:	e8 d2 04 00 00       	call   8049176 <explode_bomb>
 8048ca4:	b8 6e 00 00 00       	mov    $0x6e,%eax
 8048ca9:	eb 5b                	jmp    8048d06 <phase_3+0x14a>
 8048cab:	b8 6d 00 00 00       	mov    $0x6d,%eax                    // x = 5
 8048cb0:	81 7c 24 28 40 03 00 	cmpl   $0x340,0x28(%esp)             // z = 0x340 =
832, y = 0x6d = 109 = 'm'
 8048cb7:	00
 8048cb8:	74 4c                	je     8048d06 <phase_3+0x14a>
 8048cba:	e8 b7 04 00 00       	call   8049176 <explode_bomb>
 8048cbf:	b8 6d 00 00 00       	mov    $0x6d,%eax
 8048cc4:	eb 40                	jmp    8048d06 <phase_3+0x14a>
 8048cc6:	b8 72 00 00 00       	mov    $0x72,%eax                    // x = 6
 8048ccb:	81 7c 24 28 8c 03 00 	cmpl   $0x38c,0x28(%esp)             // z = 0x38c =
908, y = 0x72 = 114 = 'r'
 8048cd2:	00
 8048cd3:	74 31                	je     8048d06 <phase_3+0x14a>
 8048cd5:	e8 9c 04 00 00       	call   8049176 <explode_bomb>
 8048cda:	b8 72 00 00 00       	mov    $0x72,%eax
 8048cdf:	eb 25                	jmp    8048d06 <phase_3+0x14a>
 8048ce1:	b8 70 00 00 00       	mov    $0x70,%eax                    // x = 7
 8048ce6:	81 7c 24 28 67 01 00 	cmpl   $0x167,0x28(%esp)             // z = 0x167 =
359, y = 0x70 = 112 = 'p'
 8048ced:	00
 8048cee:	74 16                	je     8048d06 <phase_3+0x14a>
 8048cf0:	e8 81 04 00 00       	call   8049176 <explode_bomb>
 8048cf5:	b8 70 00 00 00       	mov    $0x70,%eax
 8048cfa:	eb 0a                	jmp    8048d06 <phase_3+0x14a>
 8048cfc:	e8 75 04 00 00       	call   8049176 <explode_bomb>
 8048d01:	b8 6c 00 00 00       	mov    $0x6c,%eax
 8048d06:	3a 44 24 2f          	cmp    0x2f(%esp),%al
 8048d0a:	74 05                	je     8048d11 <phase_3+0x155>
 8048d0c:	e8 65 04 00 00       	call   8049176 <explode_bomb>
 8048d11:	83 c4 3c             	add    $0x3c,%esp
 8048d14:	c3                   	ret
```

`<phase_3>` 中调用了 `<__isoc99_sscanf@plt>` 函数，类似于 `scanf`，参数要提供输入格式和输入的数据，返回输入数据的个数。在调用前的准备工作中出现了地址 `0x804a2a2`，查看发现是字符串 `"%d %c %d"`，这就是该函数的输入格式，说明我们要输入 3 个数据，分别是整数、字符、整数（`32, A, 254`）。



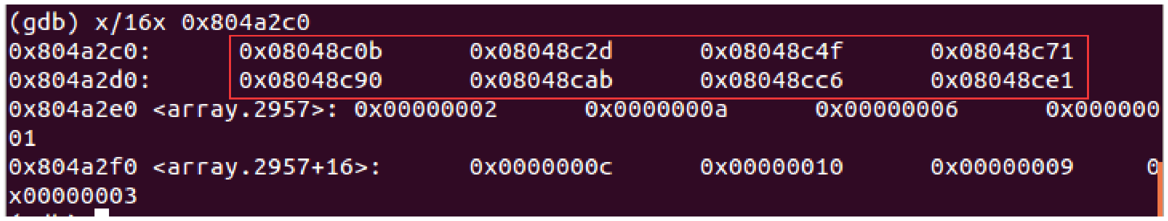
```
(gdb) x/1s 0x804a2a2
0x804a2a2:	 "%d %c %d"
(gdb)
```



```
(gdb) p *(int*)($esp + 0x24)
$1 = 32
(gdb) p ($esp + 0x24)
$2 = (void *) 0xbffff2d4
(gdb) x/8x 0xbffff2d4
0xbffff2d4:	0x00000020	0x000000fe	0x41fff3a4	0xbffff3a4
0xbffff2e4:	0x00000000	0x00000000	0x08048adf	0x0804c480
(gdb) p *(int*)($esp + 0x28)
$3 = 254
(gdb) p *(int*)($esp + 0x2f)
$4 = -809919
(gdb) p *(char*)($esp + 0x2f)
$5 = 65 'A'
(gdb) 
```

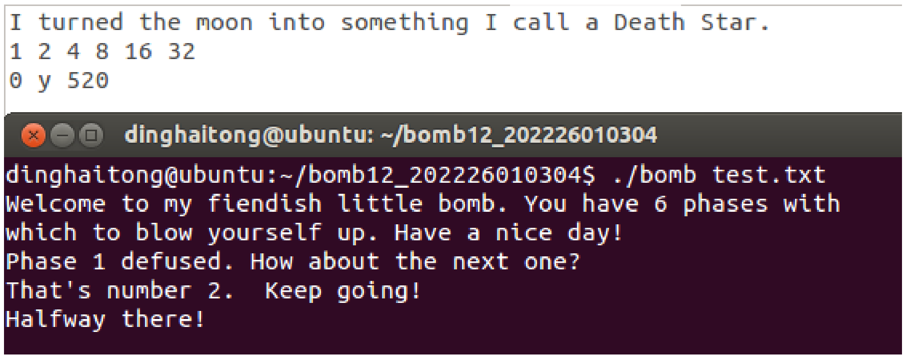
接着将参数 `x` 和 `0x7` 进行无符号数比较，若是 x 大于 7 就爆炸，说明 `0 <= x < 7`。

将 `x` 作为索引值，跳转到 `(0x804a2c0 + 4 * %eax)` 指向的指令地址，可以分析出来这类似于 switch 表。



```
(gdb) x/16x 0x804a2c0
0x804a2c0:	0x08048c0b	0x08048c2d	0x08048c4f	0x08048c71
0x804a2d0:	0x08048c90	0x08048cab	0x08048cc6	0x08048ce1
0x804a2e0 <array.2957>:	0x00000002	0x0000000a	0x00000006	0x000000
01
0x804a2f0 <array.2957+16>:	0x0000000c	0x00000010	0x00000009	0
x00000003
```

这里以 `x=0` 为例，z 必须为 `0x23c`，继续跳转，`y` 必须为 `%eax = 0x79` 的后 8 位，即 `y` 必须为 `0x79 = 'y'`，所以 `0 y 520` 就是其中一个正确答案，由于 `x` 可以取 `[0, 7]` 中的整数，所以一共有 8 个答案，在上述代码中全部写出。



```
I turned the moon into something I call a Death Star.
1 2 4 8 16 32
0 y 520
```

dinghaitong@ubuntu: ~/bomb12_202226010304

```
dinghaitong@ubuntu:~/bomb12_202226010304$ ./bomb test.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!
```

> ***the ans of phase_3： 0 y 520***

phase_4

- `jne` -> `jump if not equal`

- `js` -> `jump if sign`

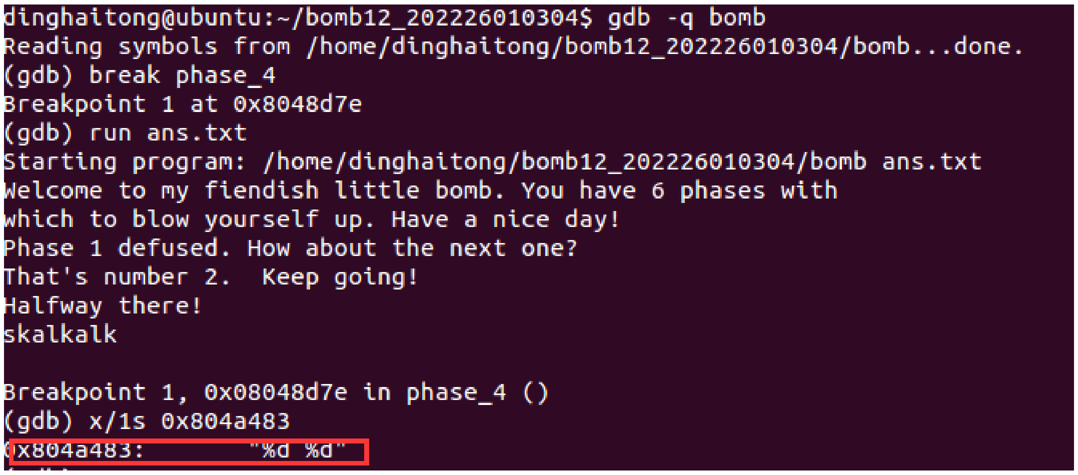
在 `asm.txt` 中找到 `<phase_4>`，汇编代码如下：

```
08048d7e <phase_4>:
 8048d7e:	83 ec 2c             	sub    $0x2c,%esp
 8048d81:	8d 44 24 1c          	lea    0x1c(%esp),%eax
 8048d85:	89 44 24 0c          	mov    %eax,0xc(%esp)                  // (%esp + 
0xc) = %esp + 0x1c, (%esp + 0x1c) = y
 8048d89:	8d 44 24 18          	lea    0x18(%esp),%eax
 8048d8d:	89 44 24 08          	mov    %eax,0x8(%esp)                  // (%esp + 
0x8) = %esp + 0x18, (%esp + 0x18) = x
 8048d91:	c7 44 24 04 83 a4 04 	movl   $0x804a483,0x4(%esp)
 8048d98:	08 
 8048d99:	8b 44 24 30          	mov    0x30(%esp),%eax
 8048d9d:	89 04 24             	mov    %eax,(%esp)                    // （%esp） = 
输入
 8048da0:	e8 cb fa ff ff       	call   8048870 <__isoc99_sscanf@plt>
 8048da5:	83 f8 02             	cmp    $0x2,%eax
 8048da8:	75 0d                	jne    8048db7 <phase_4+0x39>         //  如果返回
值%eax != 2 就跳转爆炸
 8048daa:	8b 44 24 18          	mov    0x18(%esp),%eax                //  %eax = x
 8048dae:	85 c0                	test   %eax,%eax
 8048db0:	78 05                	js     8048db7 <phase_4+0x39>         //  if  x < 0 
跳转爆炸
 8048db2:	83 f8 0e             	cmp    $0xe,%eax
 8048db5:	7e 05                	jle    8048dbc <phase_4+0x3e>         //  if  x > 
0xe = 14 跳转爆炸
 8048db7:	e8 ba 03 00 00       	call   8049176 <explode_bomb>
 8048dbc:	c7 44 24 08 0e 00 00 	movl   $0xe,0x8(%esp)                 //  let (%esp
+ 0x8) = 0xe = 14
 8048dc3:	00 
 8048dc4:	c7 44 24 04 00 00 00 	movl   $0x0,0x4(%esp)                 //  let (%esp
+ 0x4) = 0x0 = 0
 8048dcb:	00 
 8048dcc:	8b 44 24 18          	mov    0x18(%esp),%eax                //  %eax = x
 8048dd0:	89 04 24             	mov    %eax,(%esp)                    //  let (%esp)
= %eax = x
 8048dd3:	e8 3d ff ff ff       	call   8048d15 <func4>                //  func4(14, 
0, x)
                                                                    //  在调用fun4之前，会
存储main下一条指令0x8048dd8，占用4字节
                                                                    //  也就是
0xbffff2bc: 0x8048dd8
 8048dd8:	83 f8 23             	cmp    $0x23,%eax
 8048ddb:	75 07                	jne    8048de4 <phase_4+0x66>         // if 返回值!= 
0x23 = 35 跳转爆炸
 8048ddd:	83 7c 24 1c 23       	cmpl   $0x23,0x1c(%esp)               // y 必须= 0x23
= 35
 8048de2:	74 05                	je     8048de9 <phase_4+0x6b>
 8048de4:	e8 8d 03 00 00       	call   8049176 <explode_bomb>
 8048de9:	83 c4 2c             	add    $0x2c,%esp
 8048dec:	c3                   	ret    

```

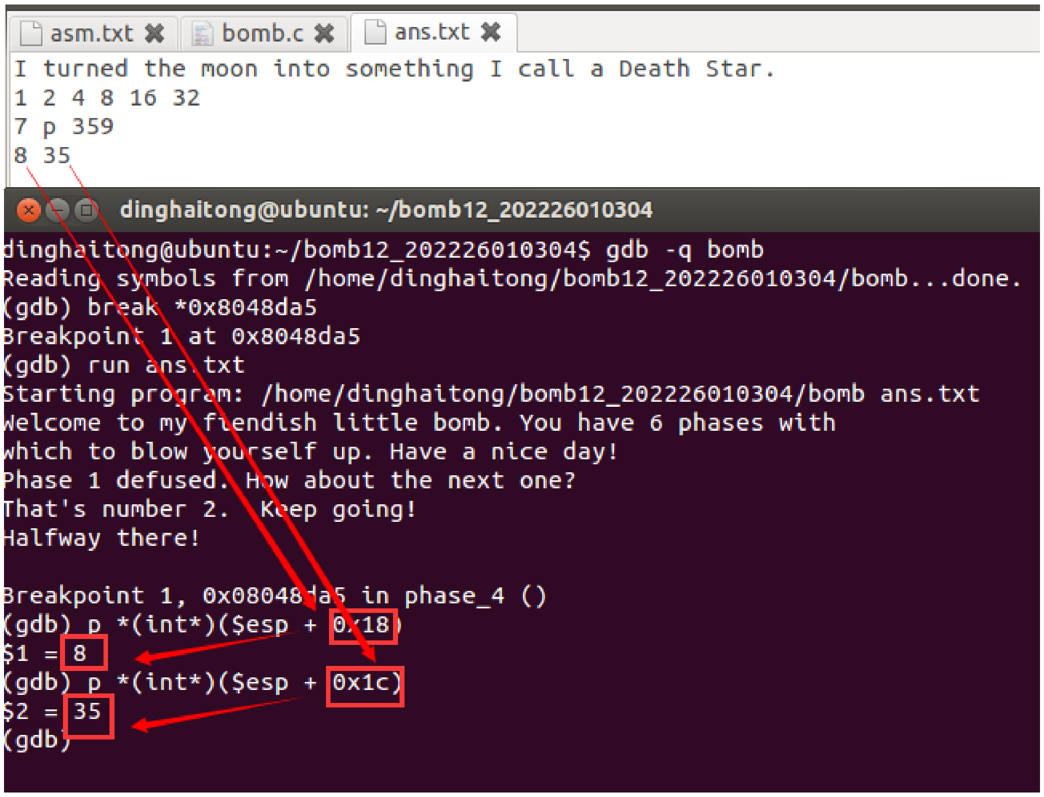
```
08048d15 <func4>:
 8048d15:	83 ec 1c             	sub    $0x1c,%esp
 8048d18:	89 5c 24 14          	mov    %ebx,0x14(%esp)
 8048d1c:	89 74 24 18          	mov    %esi,0x18(%esp)
 8048d20:	8b 44 24 20          	mov    0x20(%esp),%eax // c x
 8048d24:	8b 54 24 24          	mov    0x24(%esp),%edx // b 0
 8048d28:	8b 74 24 28          	mov    0x28(%esp),%esi // a 14
 8048d2c:	89 f1                	mov    %esi,%ecx       // %ecx = a
 8048d2e:	29 d1                	sub    %edx,%ecx       // %ecx = a - b
 8048d30:	89 cb                	mov    %ecx,%ebx       // %ebx = a - b
 8048d32:	c1 eb 1f             	shr    $0x1f,%ebx      // %ebx = (a - b) >> 31 
(logical)  --->得到符号位
 8048d35:	01 d9                	add    %ebx,%ecx       // %ecx = (a - b) + [(a - b) >> 
31 (logical)]  ---> a-b + off
 8048d37:	d1 f9                	sar    %ecx            // %ecx = %ecx >> 1 (arithmetic) 
--->  (a-b)/2 (向0取整)
 8048d39:	8d 1c 11             	lea    (%ecx,%edx,1),%ebx    // %ebx = (a-b)/2 + b
 8048d3c:	39 c3                	cmp    %eax,%ebx             // %eax = c, (a-b)/2 + b - 
c

 8048d3e:	7e 17                	jle    8048d57 <func4+0x42> // if (a-b)/2 + b <= c jump
 8048d40:	8d 4b ff             	lea    -0x1(%ebx),%ecx  //v1 > v2   //%ecx = (a-b)/2 + 
b - 1
 8048d43:	89 4c 24 08          	mov    %ecx,0x8(%esp)        // (%esp + 0x8) = %ecx = 
(a-b)/2 + b - 1
 8048d47:	89 54 24 04          	mov    %edx,0x4(%esp)        // (%esp + 0x4) = %edx = b
 8048d4b:	89 04 24             	mov    %eax,(%esp)           // (%esp) = %eax = c
 8048d4e:	e8 c2 ff ff ff       	call   8048d15 <func4>       // func4( (a-b)/2 + b - 1, 
b, c )
 8048d53:	01 c3                	add    %eax,%ebx             // %ebx = func4((a-b)/2 + b 
- 1, b, c)  + (a-b)/2 + b
 8048d55:	eb 19                	jmp    8048d70 <func4+0x5b> // ---->return func4((a-
b)/2 + b - 1, b, c)  + (a-b)/2 + b
 8048d57:	39 c3                	cmp    %eax,%ebx

 8048d59:	7d 15                	jge    8048d70 <func4+0x5b> //if (a-b)/2 + b = c jump
 8048d5b:	89 74 24 08          	mov    %esi,0x8(%esp)   //v1 < v2     // (%esp + 0x8) = 
%esi = a
 8048d5f:	8d 53 01             	lea    0x1(%ebx),%edx        // %edx = %ebx + 0x1 = (a-
b)/2 + b + 1
 8048d62:	89 54 24 04          	mov    %edx,0x4(%esp)        // (%esp + 0x4) = %edx = 
(a-b)/2 + b + 1
 8048d66:	89 04 24             	mov    %eax,(%esp)           // (%esp) = %eax = c
 8048d69:	e8 a7 ff ff ff       	call   8048d15 <func4>       // func4(a, (a-b)/2 + b + 
1, c)
 8048d6e:	01 c3                	add    %eax,%ebx             // ---->return func4(a, (a-
b)/2 + b + 1, c) + (a-b)/2 + b
```

```
 8048d70:       89 d8                   mov    %ebx,%eax         //v1 == v2      // ---->return 
(a-b)/2 + b
 8048d72:       8b 5c 24 14             mov    0x14(%esp),%ebx
 8048d76:       8b 74 24 18             mov    0x18(%esp),%esi
 8048d7a:       83 c4 1c                add    $0x1c,%esp
 8048d7d:       c3                      ret
```

这里同样调用了 `<__isoc99_sscanf@plt>` 函数，通过 `0x804a483` 可以得知，输入模式为 `"%d %d"`。设两个参数 `(%esp + 0 x 1 c) = y`，`(%esp + 0 x 18) = x`



```
dinghaitong@ubuntu:~/bomb12_202226010304$ gdb -q bomb
Reading symbols from /home/dinghaitong/bomb12_202226010304/bomb...done.
(gdb) break phase_4
Breakpoint 1 at 0x8048d7e
(gdb) run ans.txt
Starting program: /home/dinghaitong/bomb12_202226010304/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!
skalkalk

Breakpoint 1, 0x08048d7e in phase_4 ()
(gdb) x/1s 0x804a483
0x804a483:        "%d %d"
```



```
I turned the moon into something I call a Death Star.
1 2 4 8 16 32
7 p 359
8 35
```

```
dinghaitong@ubuntu:~/bomb12_202226010304$ gdb -q bomb
Reading symbols from /home/dinghaitong/bomb12_202226010304/bomb...done.
(gdb) break *0x8048da5
Breakpoint 1 at 0x8048da5
(gdb) run ans.txt
Starting program: /home/dinghaitong/bomb12_202226010304/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!

Breakpoint 1, 0x08048da5 in phase_4 ()
(gdb) p *(int*)($esp + 0x18)
$1 = 8
(gdb) p *(int*)($esp + 0x1c)
$2 = 35
(gdb)
```

之后判断 x 的取值范围，要求在 `[1, 13]` 之间的整数。

接着会调用 `<func4>` 的函数，可以发现向其传递了三个参数 `(%esp + 0x8) = 0xe = 14`、`(%esp + 0x4) = 0x0 = 0`、`x`，也即调用 `func4(14, 0, x)`。返回值存储在 `%eax` 中，如果返回值不是 `35` 就爆炸，`0x1c(%esp)` 也即 `y` 不是 35 就爆炸。所以我们得到 2 个等式：`fun4(14, 0, x) = 35, y = 35`。接下来我们分析 `func4`。

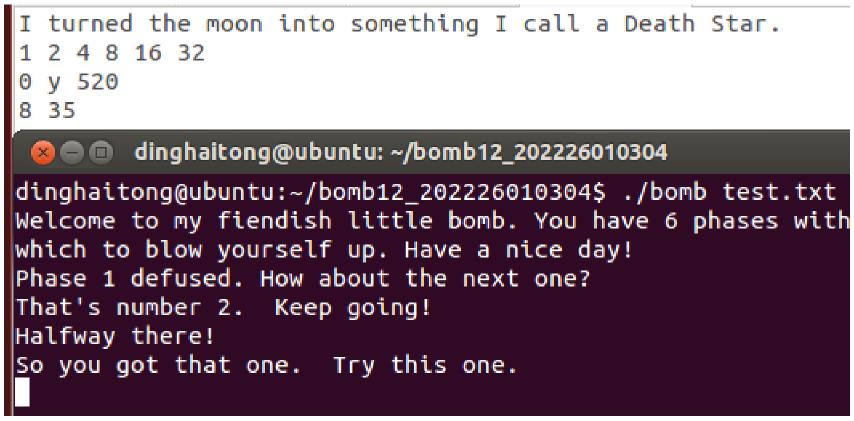
截至 `8048d39` 之前，一方面是把传入的参数写到寄存器中；另一方面是实现 `%ecx = (a - b) / 2` （向零取整），之后会根据 `(a - b) / 2 +  b - c` 的正负值分支，并再次调用 `func4` 函数，说明这是一个递归函数，这里将 `func4` 函数的 C 代码实现写在下方：

```c
int func4(int a, int b, int c)
{
        int v1 = (a - b) / 2 +  b;
        int v2 = c;
        if(v1 == v2)
        {
                return (a - b) / 2 +  b;
        }
        else if(v1 < v2)
        {
                return func4(a, (a-b)/2 + b + 1, c) + (a-b)/2 + b
        }
        else
        {
                return func4((a-b)/2 + b - 1, b, c)  + (a-b)/2 + b
        }
}
```

最后我们需要得到 `fun4(14, 0, x) = 35` 中 `x` 的值，可以再写一个 `solve` 函数求解：

```c
int solve()
{
        int i= 0;
        for(; i <= 13; i++)
                if(func4(14, 0, i) == 35)
                        break;

        return i;
}
```

得到 `x = 8, y = 35` , 验证成功



```
I turned the moon into something I call a Death Star.
1 2 4 8 16 32
0 y 520
8 35
```

dinghaitong@ubuntu: ~/bomb12_202226010304

```
dinghaitong@ubuntu:~/bomb12_202226010304$ ./bomb test.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.
```

> ***the ans of phase_4 ： 8 35***

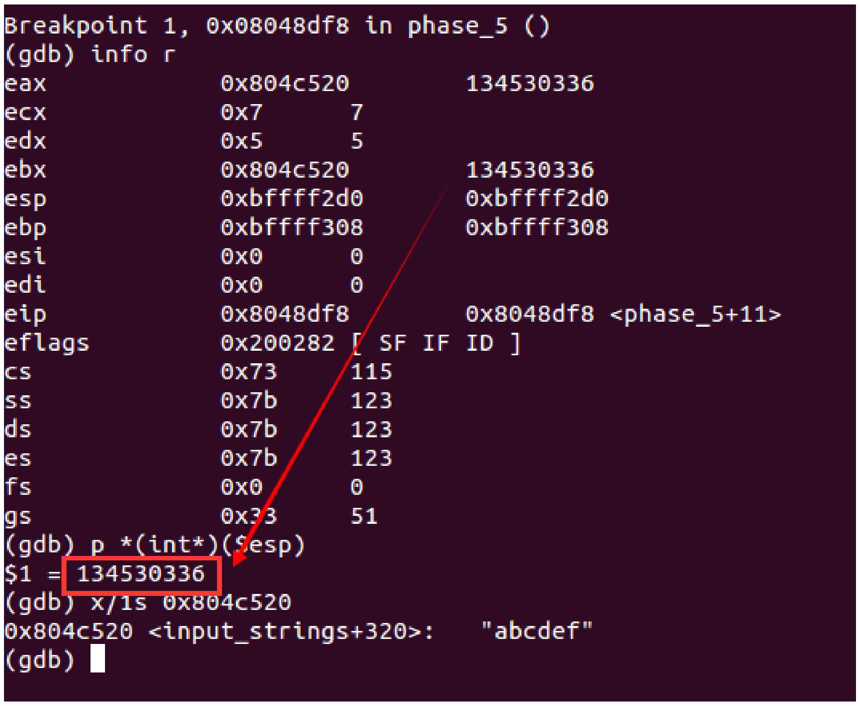
phase_5

- `movsbl`：符号扩展地传送

在 `asm.txt` 中找到 `<phase_5>`，汇编代码如下：

```
08048ded <phase_5>:
 8048ded:	53                   	push   %ebx
 8048dee:	83 ec 18             	sub    $0x18,%esp
 8048df1:	8b 5c 24 20          	mov    0x20(%esp),%ebx          // %ebx = (%esp + 0x20)
 8048df5:	89 1c 24             	mov    %ebx,(%esp)              // (%esp) = %ebx = (%esp + 0x20) = 输入
 8048df8:	e8 4e 02 00 00       	call   804904b <string_length>
 8048dfd:	83 f8 06             	cmp    $0x6,%eax                // %eax - 0x6
 8048e00:	74 05                	je     8048e07 <phase_5+0x1a>   // if string_length == 6 进入下一步
 8048e02:	e8 6f 03 00 00       	call   8049176 <explode_bomb>
 8048e07:	ba 00 00 00 00       	mov    $0x0,%edx                // %edx = 0
 8048e0c:	b8 00 00 00 00       	mov    $0x0,%eax                // %eax = 0
 8048e11:	0f be 0c 03          	movsbl (%ebx,%eax,1),%ecx       // %ecx = (%eax + %ebx) 符号扩展
 8048e15:	83 e1 0f             	and    $0xf,%ecx                // %ecx = %ecx & $0xf -> index
 8048e18:	03 14 8d e0 a2 04 08 	add    0x804a2e0(,%ecx,4),%edx  // %dex += (0x804a2e0 + 4 * %ecx)
 8048e1f:	83 c0 01             	add    $0x1,%eax                // %eax++
 8048e22:	83 f8 06             	cmp    $0x6,%eax                // %eax ? 6
 8048e25:	75 ea                	jne    8048e11 <phase_5+0x24>
 8048e27:	83 fa 2e             	cmp    $0x2e,%edx
 8048e2a:	74 05                	je     8048e31 <phase_5+0x44>
 8048e2c:	e8 45 03 00 00       	call   8049176 <explode_bomb>
 8048e31:	83 c4 18             	add    $0x18,%esp
 8048e34:	5b                   	pop    %ebx
 8048e35:	c3                   	ret
```

调用函数 `<string_length>`，传入 1 个参数 `0x20(%esp)`，也即此次输入。

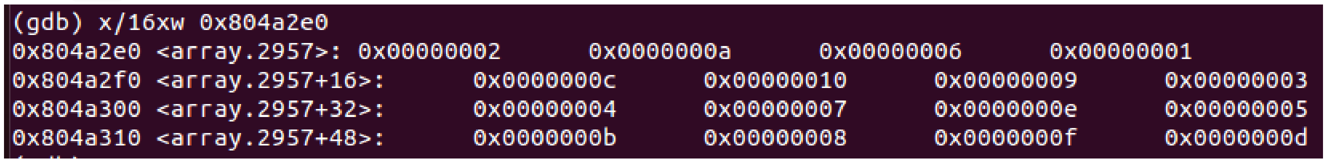


```
Breakpoint 1, 0x08048df8 in phase_5 ()
(gdb) info r
eax            0x804c520        134530336
ecx            0x7      7
edx            0x5      5
ebx            0x804c520        134530336
esp            0xbffff2d0       0xbffff2d0
ebp            0xbffff308       0xbffff308
esi            0x0      0
edi            0x0      0
eip            0x8048df8        0x8048df8 <phase_5+11>
eflags         0x200282 [ SF IF ID ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
(gdb) p *(int*)($esp)
$1 = 134530336
(gdb) x/1s 0x804c520
0x804c520 <input_strings+320>:   "abcdef"
(gdb) 
```

返回值为 6 不爆炸，说明输入字符串长度要为 6。
之后进入了一个循环， `%eax` 是索引值 `i` ，增长顺序为 `0 -> 1 -> ... -> 5` ，根据 `i` 每次从输入的 6 个字符中按顺序取出一个字符 `c` ，取其后四位作为索引值 `j` 。把 `j` 作为数组 `val` 的索引（ `val = 0x804a2e0` ），取值将其加到 `%edx` 。最后需要保证 `%edx == 0x2e` 。将其写成 C 代码如下：

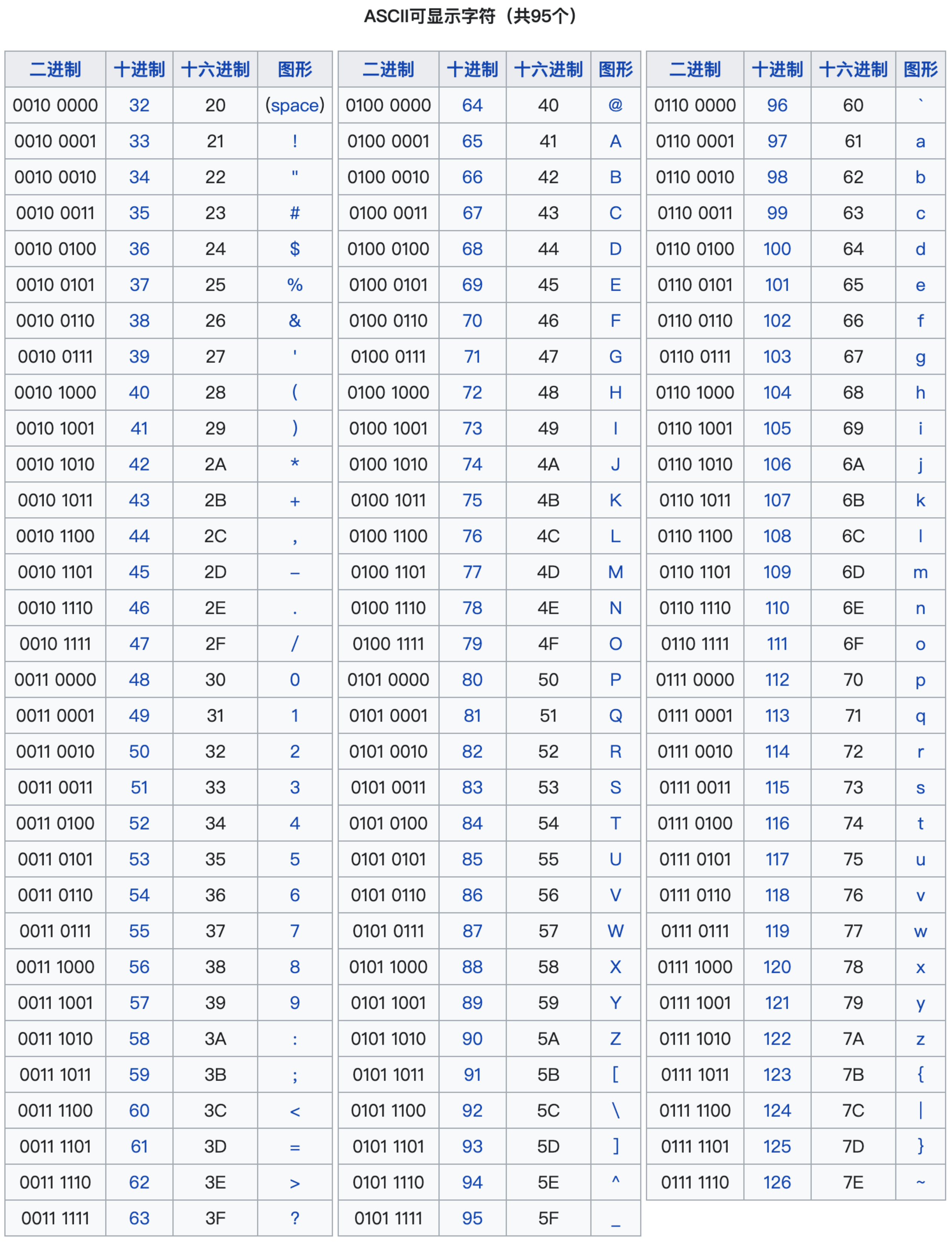
```
char[6] str = { 输入... }; // str = %ebx
int sum = 0;
int[?] val = {...}  //val = 0x804a2e0
for(int i = 0; i < 6; i++)
{
        int index = char[i] & 0xf;
        sum = val[index];
}
if (sum != 0x2e) bomb();
//通关...
```

数组 `val` 的情况如下：



```
(gdb) x/16xw 0x804a2e0
0x804a2e0 <array.2957>: 0x00000002      0x0000000a      0x00000006      0x00000001
0x804a2f0 <array.2957+16>:      0x0000000c      0x00000010      0x00000009      0x00000003
0x804a300 <array.2957+32>:      0x00000004      0x00000007      0x0000000e      0x00000005
0x804a310 <array.2957+48>:      0x0000000b      0x00000008      0x0000000f      0x0000000d
```

根据 ASCII 表可以组合出很多种答案，这里



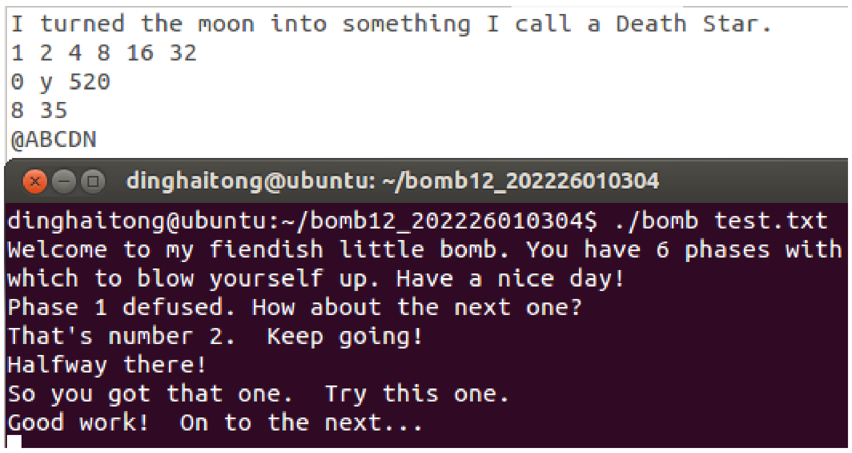
ASCII可显示字符（共95个）

二进制	十进制	十六进制	图形
0010 0000	32	20	(space)
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29	)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	,
0010 1101	45	2D	–
0010 1110	46	2E	.
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:
0011 1011	59	3B	;
0011 1100	60	3C	<
0011 1101	61	3D	=
0011 1110	62	3E	>
0011 1111	63	3F	?
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W
0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	\
0101 1101	93	5D	]
0101 1110	94	5E	^
0101 1111	95	5F	_
0110 0000	96	60	`
0110 0001	97	61	a
0110 0010	98	62	b
0110 0011	99	63	c
0110 0100	100	64	d
0110 0101	101	65	e
0110 0110	102	66	f
0110 0111	103	67	g
0110 1000	104	68	h
0110 1001	105	69	i
0110 1010	106	6A	j
0110 1011	107	6B	k
0110 1100	108	6C	l
0110 1101	109	6D	m
0110 1110	110	6E	n
0110 1111	111	6F	o
0111 0000	112	70	p
0111 0001	113	71	q
0111 0010	114	72	r
0111 0011	115	73	s
0111 0100	116	74	t
0111 0101	117	75	u
0111 0110	118	76	v
0111 0111	119	77	w
0111 1000	120	78	x
0111 1001	121	79	y
0111 1010	122	7A	z
0111 1011	123	7B	{
0111 1100	124	7C	|
0111 1101	125	7D	}
0111 1110	126	7E	~

`0x2e = 0x2 + 0xa + 0x6 + 0x1 + 0xc + 0xf`

`ans = @ A B C D N`

验证成功：



```
I turned the moon into something I call a Death Star.
1 2 4 8 16 32
0 y 520
8 35
@ABCDN
```

```
dinghaitong@ubuntu: ~/bomb12_202226010304
dinghaitong@ubuntu:~/bomb12_202226010304$ ./bomb test.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...
```

> ***the ans of phase_5： @ABCDN***

phase_6

在 `asm.txt` 中找到 `<phase_6>`，汇编代码如下，由于汇编代码较长，笔者会分部分讲解：

1. 输入

```
08048e36 <phase_6>:
 8048e36:	56                   	push   %esi
 8048e37:	53                   	push   %ebx
 8048e38:	83 ec 44             	sub    $0x44,%esp
 8048e3b:	8d 44 24 10          	lea    0x10(%esp),%eax
 8048e3f:	89 44 24 04          	mov    %eax,0x4(%esp)
 8048e43:	8b 44 24 50          	mov    0x50(%esp),%eax
 8048e47:	89 04 24             	mov    %eax,(%esp)
 8048e4a:	e8 5c 04 00 00       	call   80492ab <read_six_numbers>
```

调用函数<read_six_numbers>可知，此次输入是 6 个数字。

2. 校验输入

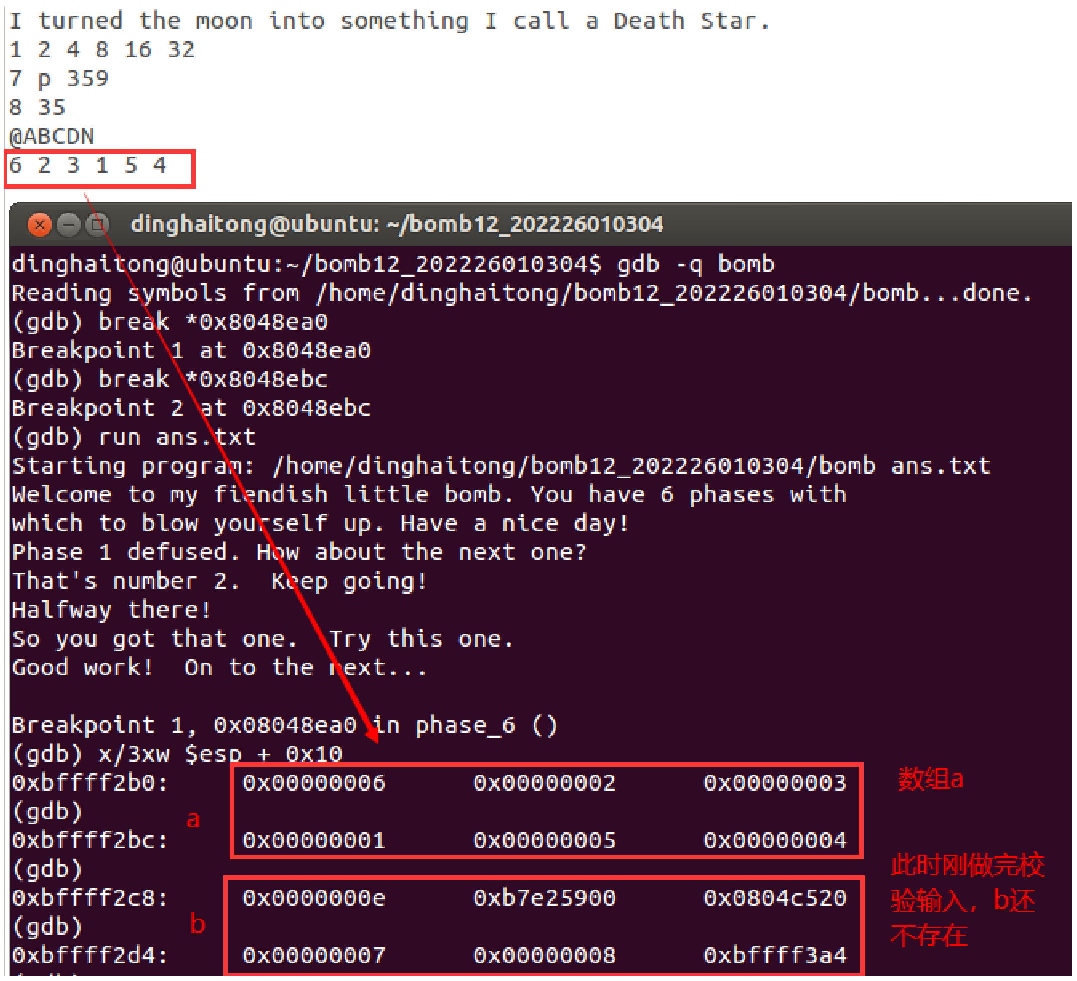
```
 8048e4f:	be 00 00 00 00       	mov    $0x0,%esi
 8048e54:	8b 44 b4 10          	mov    0x10(%esp,%esi,4),%eax
 8048e58:	83 e8 01             	sub    $0x1,%eax              //这一步如果eax = 0, eax就会变成大数，说明输入不能是0
 8048e5b:	83 f8 05             	cmp    $0x5,%eax
 8048e5e:	76 05                	jbe    8048e65 <phase_6+0x2f>  //jbe是无符号数比较,说明读入的数不能是负数
 8048e60:	e8 11 03 00 00       	call   8049176 <explode_bomb>
 8048e65:	83 c6 01             	add    $0x1,%esi
 8048e68:	83 fe 06             	cmp    $0x6,%esi
 8048e6b:	74 33                	je     8048ea0 <phase_6+0x6a>
 8048e6d:	89 f3                	mov    %esi,%ebx
 8048e6f:	8b 44 9c 10          	mov    0x10(%esp,%ebx,4),%eax
 8048e73:	39 44 b4 0c          	cmp    %eax,0xc(%esp,%esi,4)
 8048e77:	75 05                	jne    8048e7e <phase_6+0x48>
 8048e79:	e8 f8 02 00 00       	call   8049176 <explode_bomb>
 8048e7e:	83 c3 01             	add    $0x1,%ebx
 8048e81:	83 fb 05             	cmp    $0x5,%ebx
 8048e84:	7e e9                	jle    8048e6f <phase_6+0x39>
 8048e86:	eb cc                	jmp    8048e54 <phase_6+0x1e>
```

这里面共用内外两层循环。`(%esp +10) = a[0]`，数组 `a` 就是输入的数据组成的数组。
`%esi` 表示外循环索引值 `i`，`%eax = a[i]`，这里如果 `a[i] > 6` 就爆炸。同时又用无符号数比较，表示 `1 <= a[i] <= 6`。索引值 `i` 自增。
`%ebx = %esi`，`%ebx` 作为内循环索引值 `j` 之后开始循环，如果 `a[i-1] == a[j]` 就爆炸。索引值 `j` 自增。

这里给出反推出来的 C 代码：

```
esi = 0
	loop1:
	eax = (0x10 + esp + esi * 4);
	eax--;
	if(eax - 0x5 > 0) bomb();
	esi++;
	if(esi == 6) goto break
	ebx = esi
		loop2:
		eax = (0x10 + esp + ebx * 4);
		if(0xc + esp + esi * 4 == eax) bomb();
		ebx++;
		if(ebx <= 5) goto loop2
	goto loop1

a[0] ~ a[5] 之间不能有重复,且要在[1, 6]之间
for(int i = 0; i < 6; )
{
	if(a[i] > 6) bomb();
	i++;
	for(int j = i; j < 6; j++)
	{
		if(a[i-1] == a[j]) bomb();
	}
}
```

完成之后查看数组 `a` 和数组 `b`，此时数组 `b` 还不存在：



```
I turned the moon into something I call a Death Star.
1 2 4 8 16 32
7 p 359
8 35
@ABCDN
6 2 3 1 5 4
```

```
dinghaitong@ubuntu: ~/bomb12_202226010304
dinghaitong@ubuntu:~/bomb12_202226010304$ gdb -q bomb
Reading symbols from /home/dinghaitong/bomb12_202226010304/bomb...done.
(gdb) break *0x8048ea0
Breakpoint 1 at 0x8048ea0
(gdb) break *0x8048ebc
Breakpoint 2 at 0x8048ebc
(gdb) run ans.txt
Starting program: /home/dinghaitong/bomb12_202226010304/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...

Breakpoint 1, 0x08048ea0 in phase_6 ()
(gdb) x/3xw $esp + 0x10
0xbffff2b0:     0x00000006      0x00000002      0x00000003
(gdb)
0xbffff2bc:     0x00000001      0x00000005      0x00000004
(gdb)
0xbffff2c8:     0x0000000e      0xb7e25900      0x0804c520
(gdb)
0xbffff2d4:     0x00000007      0x00000008      0xbffff3a4
```

3. 用数组 b 表示数组 a 对应的 node

```
 8048e88:       8b 52 08                mov    0x8(%edx),%edx         // if a[%ebx] > 1, here, %edx = a[%ebx+2]
 8048e8b:       83 c0 01                add    $0x1,%eax              // %eax++
 8048e8e:       39 c8                   cmp    %ecx,%eax              // %eax ? a[%ebx]
 8048e90:       75 f6                   jne    8048e88 <phase_6+0x52>
 8048e92:       89 54 b4 28             mov    %edx,0x28(%esp,%esi,4) // a[i + 6] = <node?>
 8048e96:       83 c3 01                add    $0x1,%ebx
 8048e99:       83 fb 06                cmp    $0x6,%ebx
 8048e9c:       75 07                   jne    8048ea5 <phase_6+0x6f>
 8048e9e:       eb 1c                   jmp    8048ebc <phase_6+0x86>
 *8048ea0:      bb 00 00 00 00          mov    $0x0,%ebx               // %ebx = 0x0
 8048ea5:       89 de                   mov    %ebx,%esi               // %esi = %ebx
 8048ea7:       8b 4c 9c 10             mov    0x10(%esp,%ebx,4),%ecx  // %ecx = a[%ebx]
 8048eab:       b8 01 00 00 00          mov    $0x1,%eax               // %eax = 1
 8048eb0:       ba 3c c1 04 08          mov    $0x804c13c,%edx         // %edx = $0x804c13c <node1>
 8048eb5:       83 f9 01                cmp    $0x1,%ecx               // a[%ebx] ? 1
 8048eb8:       7f ce                   jg     8048e88 <phase_6+0x52>
 8048eba:       eb d6                   jmp    8048e92 <phase_6+0x5c>
```

这一部分实现逻辑：
假如你的输入为 `6 2 3 1 5 4`，这个时候要开辟一个数组 `b`，`b[i] = &Node_a[i]`

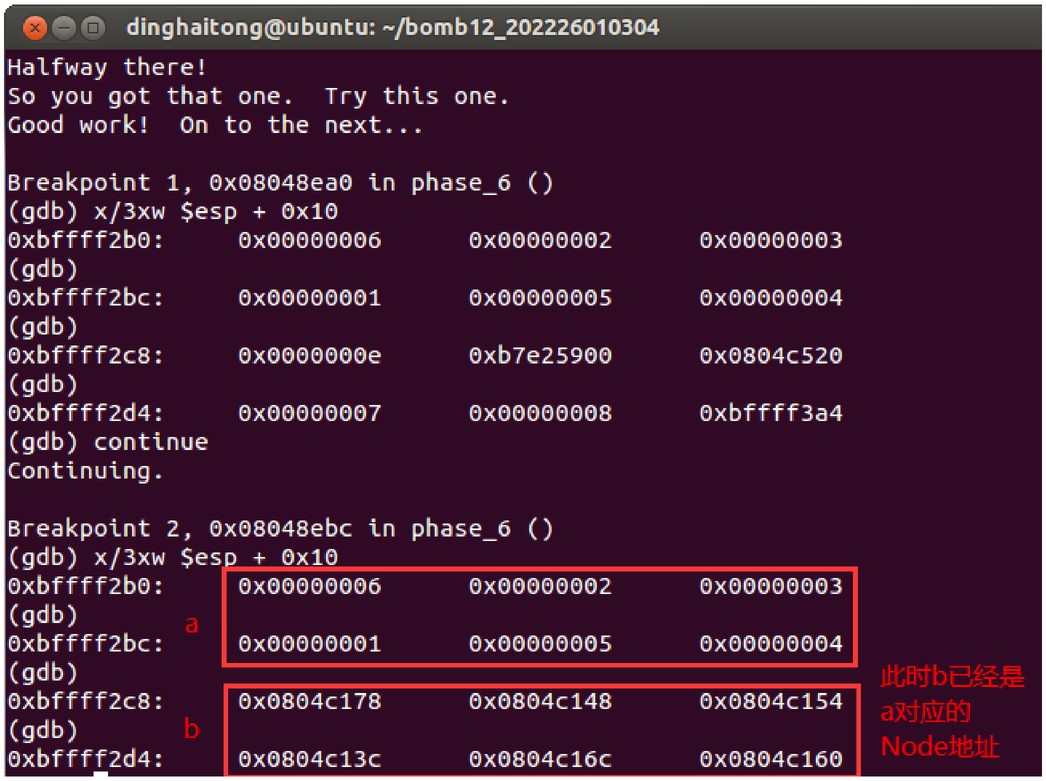
```
b 中存的是节点地址，长度是 4 字节
数组 a 0x10(%esp):        6        2        3        1        5        4
数组 b 0x28(%esp):     <node6>  <node2>  <node3>  <node1>  <node5>  <node4>
```

下面给出倒推出来的 C 代码：

```
for(ebx = 0; ebx < 6; ebx++)
{
        esi = ebx; //备份ebx
        ecx = a[ebx];
        eax = 1
        edx = $0x804c13c
        if(a[ebx] > 1)
        {
                //此循环结束后eax = a[ebx]
                //这一循环过程就是在循环访问链表，
                //每个节点有12字节，存储三项数据，分别是 {data, id, next_addr}
                //edx每次访问的都是next_addr，b数组中存储数据是地址
                for(;eax < a[ebx];eax++)
                {
                        edx = (edx + 0x8)
                }
        }
        b[ebx] = a[6 + ebx] = edx
}

b = a + 0x4*(ebx + 0x6)
for(int i = 0; i < 6; i++)
{
        Node *head = 0x804c13c
        if(a[i] > 1)
        {
                for(int j = 1; j < a[i]; j++)
                {
                        head = head->next;
                }
        }
        b[i] = head;
}
```

结束之后，查看数组 `a` 和数组 `b`，此时数组 `b` 已经存在。

```
1 2 4 8 16 32
7 p 359
8 35
@ABCDN
6 2 3 1 5 4
```



```
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...

Breakpoint 1, 0x08048ea0 in phase_6 ()
(gdb) x/3xw $esp + 0x10
0xbffff2b0:     0x00000006      0x00000002      0x00000003
(gdb)
0xbffff2bc:     0x00000001      0x00000005      0x00000004
(gdb)
0xbffff2c8:     0x0000000e      0xb7e25900      0x0804c520
(gdb)
0xbffff2d4:     0x00000007      0x00000008      0xbffff3a4
(gdb) continue
Continuing.

Breakpoint 2, 0x08048ebc in phase_6 ()
(gdb) x/3xw $esp + 0x10
0xbffff2b0:     0x00000006      0x00000002      0x00000003
(gdb)
0xbffff2bc:     0x00000001      0x00000005      0x00000004
(gdb)
0xbffff2c8:     0x0804c178      0x0804c148      0x0804c154
(gdb)
0xbffff2d4:     0x0804c13c      0x0804c16c      0x0804c160
```

dinghaitong@ubuntu: ~/bomb12_202226010304
a
b
此时b已经是a对应的Node地址

4. 利用数组 b 重排链表之间的连接顺序

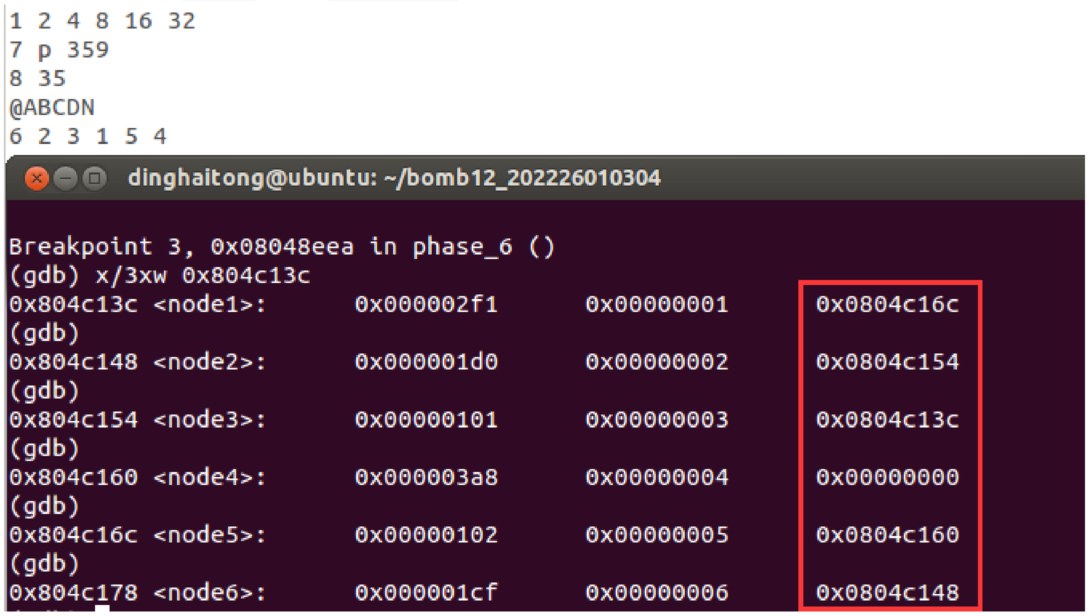
```
8048ebc:     8b 5c 24 28             mov    0x28(%esp),%ebx      // %ebx = b[0]
8048ec0:     8b 44 24 2c             mov    0x2c(%esp),%eax      // %eax = b[1]
8048ec4:     89 43 08                mov    %eax,0x8(%ebx)       // b[0].next_addr = b[1]
8048ec7:     8b 54 24 30             mov    0x30(%esp),%edx      // %edx = b[2]
8048ecb:     89 50 08                mov    %edx,0x8(%eax)       // b[1].next_addr = b[2]
8048ece:     8b 44 24 34             mov    0x34(%esp),%eax      // ...
8048ed2:     89 42 08                mov    %eax,0x8(%edx)
8048ed5:     8b 54 24 38             mov    0x38(%esp),%edx
8048ed9:     89 50 08                mov    %edx,0x8(%eax)
8048edc:     8b 44 24 3c             mov    0x3c(%esp),%eax
8048ee0:     89 42 08                mov    %eax,0x8(%edx)
8048ee3:     c7 40 08 00 00 00 00    movl   $0x0,0x8(%eax)       // b[5].next_addr = 0x0
```

利用 `%edx` 和 `%eax` 遍历链表，同时完成对节点中 `next_addr` 进行修改。

```
b 中存的是节点地址，长度是 4 字节
数组 a0x10(%esp):        6        2        3        1        5        4
数组 b0x28(%esp):     <node6>  <node2>  <node3>  <node1>  <node5>  <node4>
```

```
原本的连接顺序   : head -> <node1> -> <node2> -> <node3> -> <node4> -> <node5> -> <node6> -> 0
现在的连接顺序   : head -> <node6> -> <node2> -> <node3> -> <node1> -> <node5> -> <node4> -> 0
```

结束后，各个 `Node` 中的 `next_addr` 已经改变



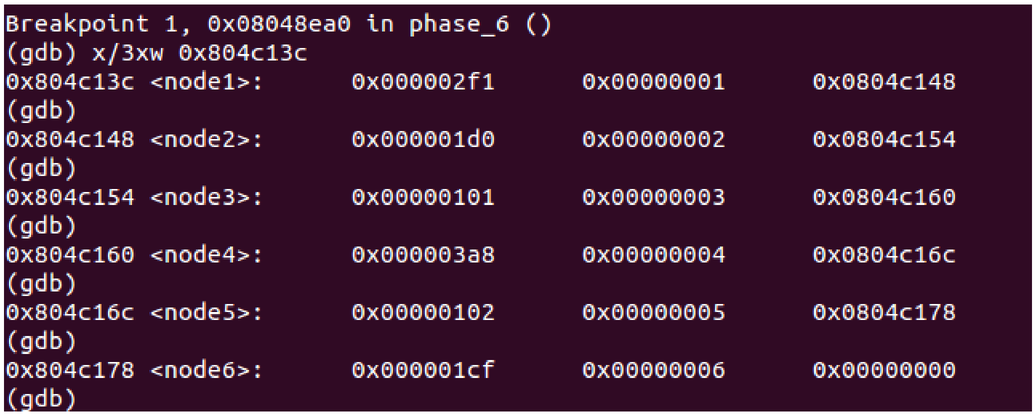
```
1 2 4 8 16 32
7 p 359
8 35
@ABCDN
6 2 3 1 5 4
```

```
dinghaitong@ubuntu: ~/bomb12_202226010304

Breakpoint 3, 0x08048eea in phase_6 ()
(gdb) x/3xw 0x804c13c
0x804c13c <node1>:      0x000002f1      0x00000001      0x0804c16c
(gdb)
0x804c148 <node2>:      0x000001d0      0x00000002      0x0804c154
(gdb)
0x804c154 <node3>:      0x00000101      0x00000003      0x0804c13c
(gdb)
0x804c160 <node4>:      0x000003a8      0x00000004      0x00000000
(gdb)
0x804c16c <node5>:      0x00000102      0x00000005      0x0804c160
(gdb)
0x804c178 <node6>:      0x000001cf      0x00000006      0x0804c148
```

5. 确保 data 递减

```
                                                                     // 此时%ebx = b[0]
  8048eea:        be 05 00 00 00          mov    $0x5,%esi               // %esi = 0x5
  8048eef:        8b 43 08                mov    0x8(%ebx),%eax          // %eax = b[0].next_addr
  8048ef2:        8b 10                   mov    (%eax),%edx             // %edx = b[1].data
  8048ef4:        39 13                   cmp    %edx,(%ebx)             // (%ebx) = b[0].data
  8048ef6:        7d 05                   jge    8048efd <phase_6+0xc7>  // 要确保b[0].data >=
b[1].data
  8048ef8:        e8 79 02 00 00          call   8049176 <explode_bomb>
  8048efd:        8b 5b 08                mov    0x8(%ebx),%ebx
  8048f00:        83 ee 01                sub    $0x1,%esi
  8048f03:        75 ea                   jne    8048eef <phase_6+0xb9>
  8048f05:        83 c4 44                add    $0x44,%esp
  8048f08:        5b                      pop    %ebx
  8048f09:        5e                      pop    %esi
  8048f0a:        c3                      ret
```

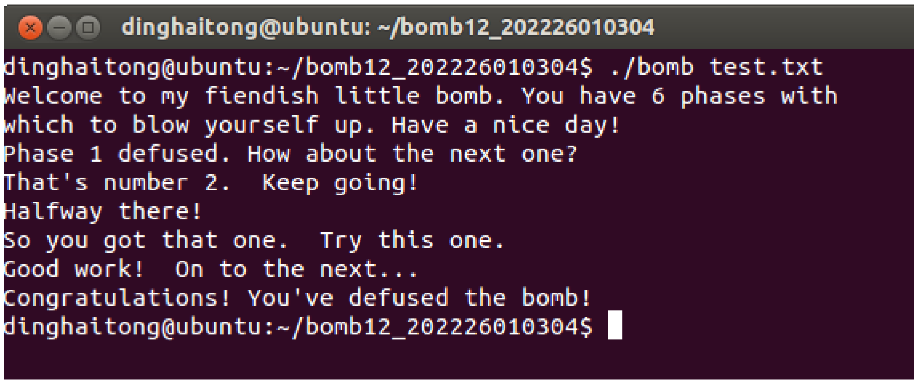
这一部分要确定如果按照我们的输入对节点重新连接，那么按顺序，节点中的 `data` 应该是递减的。
下面是未重连之前的节点数据，节点的前四个字节存储的就是 data，如果要确保递减，那么输入的顺序就应该是 `[4 1 2 6 5 3]`。



```
Breakpoint 1, 0x08048ea0 in phase_6 ()
(gdb) x/3xw 0x804c13c
0x804c13c <node1>:      0x000002f1      0x00000001      0x0804c148
(gdb)
0x804c148 <node2>:      0x000001d0      0x00000002      0x0804c154
(gdb)
0x804c154 <node3>:      0x00000101      0x00000003      0x0804c160
(gdb)
0x804c160 <node4>:      0x000003a8      0x00000004      0x0804c16c
(gdb)
0x804c16c <node5>:      0x00000102      0x00000005      0x0804c178
(gdb)
0x804c178 <node6>:      0x000001cf      0x00000006      0x00000000
(gdb)
```

验证成功：

```
I turned the moon into something I call a Death Star.
1 2 4 8 16 32
0 y 520
8 35
@ABCDN
4 1 2 6 5 3
```

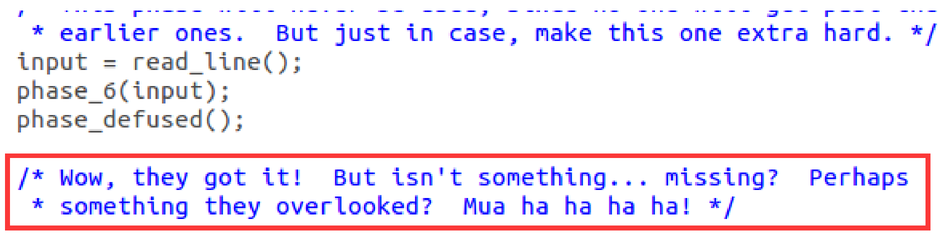


```
dinghaitong@ubuntu: ~/bomb12_202226010304
dinghaitong@ubuntu:~/bomb12_202226010304$ ./bomb test.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Congratulations! You've defused the bomb!
dinghaitong@ubuntu:~/bomb12_202226010304$
```

> ***the ans of phase_6： 4 1 2 6 5 3***

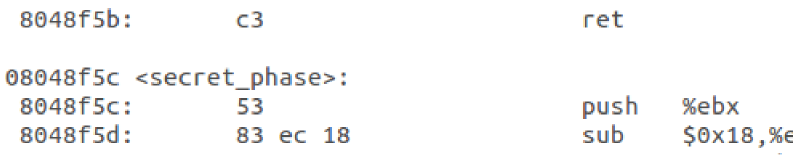
secret_phase

这一部分最初是在 bomb. c 文件中，当 6 个 phase 全部完成后，出现了如下的注释，提醒有遗漏的地方。



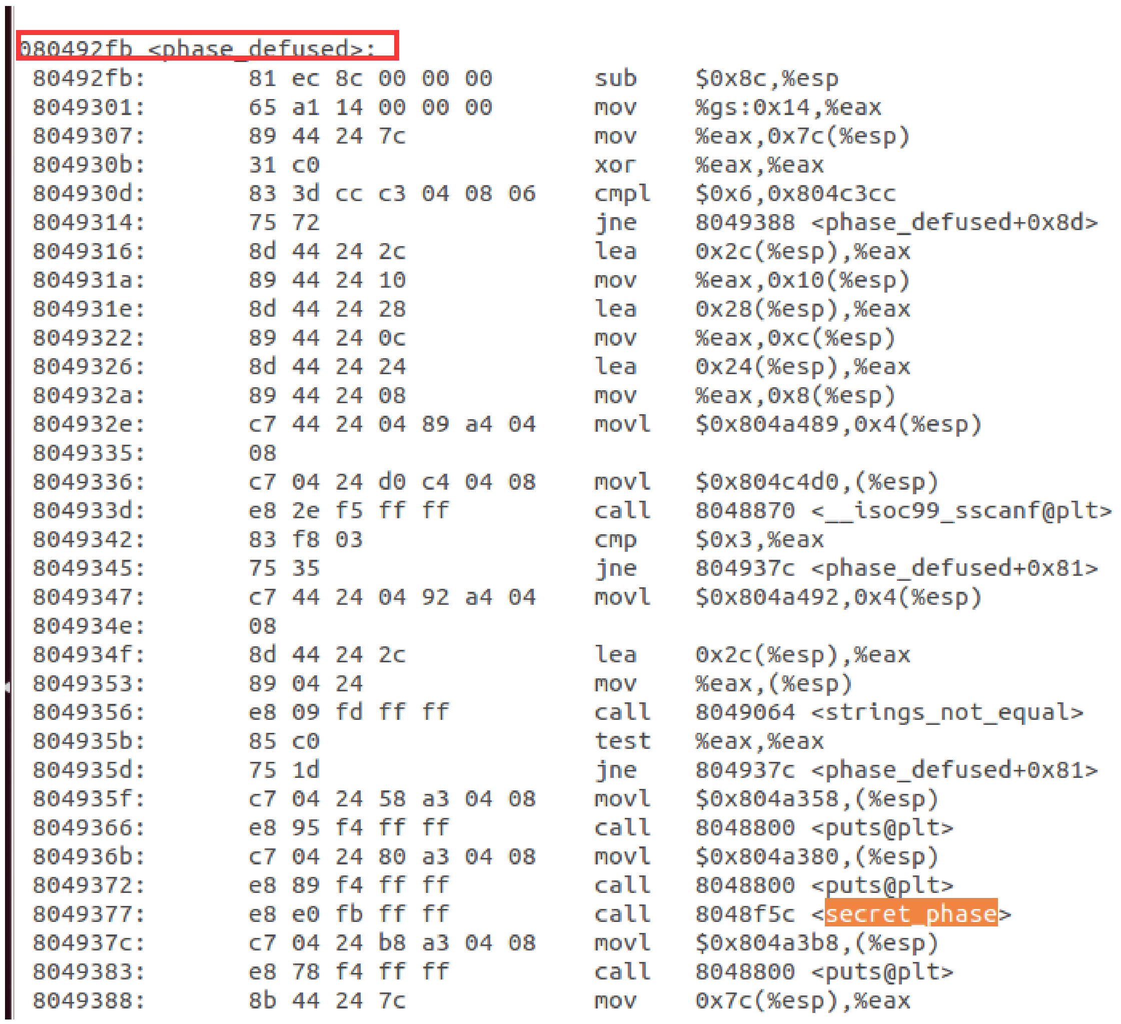
```
     * earlier ones.  But just in case, make this one extra hard. */
    input = read_line();
    phase_6(input);
    phase_defused();

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

接着就在汇编代码中找，在 phase_6 之后发现了 secret_phase。



```
 8048f5b:	c3                   	ret

08048f5c <secret_phase>:
 8048f5c:	53                   	push   %ebx
 8048f5d:	83 ec 18             	sub    $0x18,%e
```

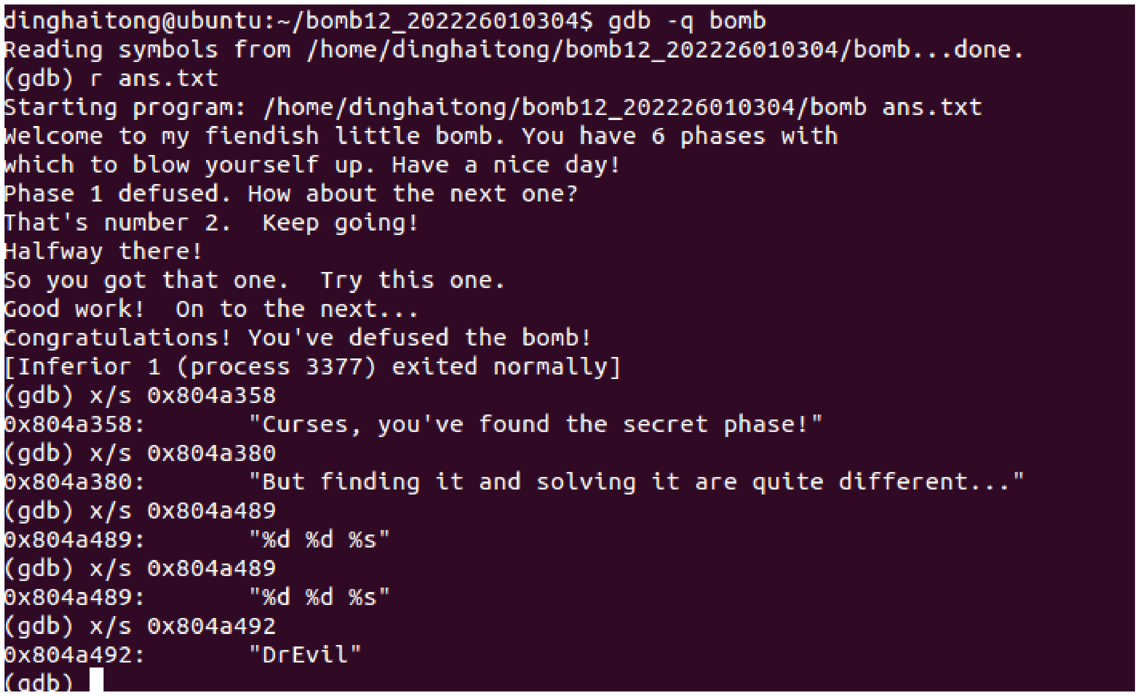
接着在汇编文件中查找哪里 call <secret_phase>，发现是在<phase_defuse>，而它在前面 6 个阶段都会调用，所以这个彩蛋就在这里开始。



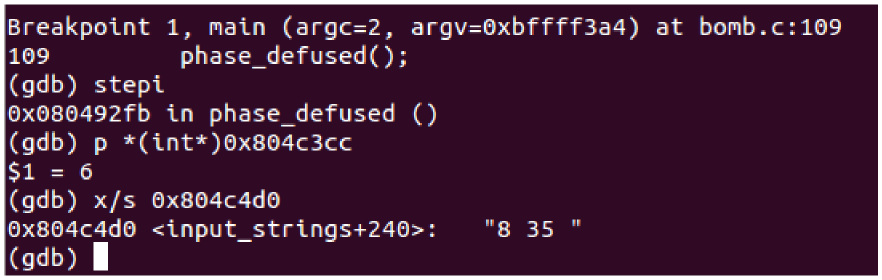
```
080492fb <phase_defused>:
 80492fb:	81 ec 8c 00 00 00    	sub    $0x8c,%esp
 8049301:	65 a1 14 00 00 00    	mov    %gs:0x14,%eax
 8049307:	89 44 24 7c          	mov    %eax,0x7c(%esp)
 804930b:	31 c0                	xor    %eax,%eax
 804930d:	83 3d cc c3 04 08 06 	cmpl   $0x6,0x804c3cc
 8049314:	75 72                	jne    8049388 <phase_defused+0x8d>
 8049316:	8d 44 24 2c          	lea    0x2c(%esp),%eax
 804931a:	89 44 24 10          	mov    %eax,0x10(%esp)
 804931e:	8d 44 24 28          	lea    0x28(%esp),%eax
 8049322:	89 44 24 0c          	mov    %eax,0xc(%esp)
 8049326:	8d 44 24 24          	lea    0x24(%esp),%eax
 804932a:	89 44 24 08          	mov    %eax,0x8(%esp)
 804932e:	c7 44 24 04 89 a4 04 	movl   $0x804a489,0x4(%esp)
 8049335:	08 
 8049336:	c7 04 24 d0 c4 04 08 	movl   $0x804c4d0,(%esp)
 804933d:	e8 2e f5 ff ff       	call   8048870 <__isoc99_sscanf@plt>
 8049342:	83 f8 03             	cmp    $0x3,%eax
 8049345:	75 35                	jne    804937c <phase_defused+0x81>
 8049347:	c7 44 24 04 92 a4 04 	movl   $0x804a492,0x4(%esp)
 804934e:	08 
 804934f:	8d 44 24 2c          	lea    0x2c(%esp),%eax
 8049353:	89 04 24             	mov    %eax,(%esp)
 8049356:	e8 09 fd ff ff       	call   8049064 <strings_not_equal>
 804935b:	85 c0                	test   %eax,%eax
 804935d:	75 1d                	jne    804937c <phase_defused+0x81>
 804935f:	c7 04 24 58 a3 04 08 	movl   $0x804a358,(%esp)
 8049366:	e8 95 f4 ff ff       	call   8048800 <puts@plt>
 804936b:	c7 04 24 80 a3 04 08 	movl   $0x804a380,(%esp)
 8049372:	e8 89 f4 ff ff       	call   8048800 <puts@plt>
 8049377:	e8 e0 fb ff ff       	call   8048f5c <secret_phase>
 804937c:	c7 04 24 b8 a3 04 08 	movl   $0x804a3b8,(%esp)
 8049383:	e8 78 f4 ff ff       	call   8048800 <puts@plt>
 8049388:	8b 44 24 7c          	mov    0x7c(%esp),%eax
```

```
080492fb <phase_defused>:
 80492fb:	81 ec 8c 00 00 00    	sub    $0x8c,%esp
 8049301:	65 a1 14 00 00 00    	mov    %gs:0x14,%eax
 8049307:	89 44 24 7c          	mov    %eax,0x7c(%esp)
 804930b:	31 c0                	xor    %eax,%eax
 804930d:	83 3d cc c3 04 08 06 	cmpl   $0x6,0x804c3cc      //0x804c3cc中存放的是通过的卡关数，在<readline>中递增
 8049314:	75 72                	jne    8049388 <phase_defused+0x8d>
 8049316:	8d 44 24 2c          	lea    0x2c(%esp),%eax
 804931a:	89 44 24 10          	mov    %eax,0x10(%esp)
 804931e:	8d 44 24 28          	lea    0x28(%esp),%eax
 8049322:	89 44 24 0c          	mov    %eax,0xc(%esp)
 8049326:	8d 44 24 24          	lea    0x24(%esp),%eax
 804932a:	89 44 24 08          	mov    %eax,0x8(%esp)
 804932e:	c7 44 24 04 89 a4 04 	movl   $0x804a489,0x4(%esp) //0x804a489:	"%d %d %s"
 8049335:	08 
 8049336:	c7 04 24 d0 c4 04 08 	movl   $0x804c4d0,(%esp)     //0x804c4d0 <input_strings+240>:	"8 35 "
 804933d:	e8 2e f5 ff ff       	call   8048870 <__isoc99_sscanf@plt>
 8049342:	83 f8 03             	cmp    $0x3,%eax            //sscanf返回的参数个数
 8049345:	75 35                	jne    804937c <phase_defused+0x81> //如果不是3就没法进入secret_phase
 8049347:       c7 44 24 04 92 a4 04    movl   $0x804a492,0x4(%esp) //0x804a492:
"DrEvil"
 804934e:       08
 804934f:       8d 44 24 2c             lea    0x2c(%esp),%eax
 8049353:       89 04 24                mov    %eax,(%esp)          //(%esp) = phase_4中写入的第
三个%s
 8049356:       e8 09 fd ff ff          call   8049064 <strings_not_equal> //此处%s = "DrEvil"
 804935b:       85 c0                   test   %eax,%eax
 804935d:       75 1d                   jne    804937c <phase_defused+0x81>
 804935f:       c7 04 24 58 a3 04 08    movl   $0x804a358,(%esp)  //0x804a358:   "Curses,
you've found the secret phase!"
 8049366:       e8 95 f4 ff ff          call   8048800 <puts@plt>
 804936b:       c7 04 24 80 a3 04 08    movl   $0x804a380,(%esp)  //0x804a380:   "But finding
it and solving it are quite different..."
 8049372:       e8 89 f4 ff ff          call   8048800 <puts@plt>
 8049377:       e8 e0 fb ff ff          call   8048f5c <secret_phase>
 804937c:       c7 04 24 b8 a3 04 08    movl   $0x804a3b8,(%esp)
 8049383:       e8 78 f4 ff ff          call   8048800 <puts@plt>
 8049388:       8b 44 24 7c             mov    0x7c(%esp),%eax
 804938c:       65 33 05 14 00 00 00    xor    %gs:0x14,%eax
 8049393:       74 05                   je     804939a <phase_defused+0x9f>
 8049395:       e8 36 f4 ff ff          call   80487d0 <__stack_chk_fail@plt>
 804939a:       81 c4 8c 00 00 00       add    $0x8c,%esp
 80493a0:       c3                      ret
 80493a1:       90                      nop
 80493a2:       90                      nop
 80493a3:       90                      nop
 80493a4:       90                      nop
 80493a5:       90                      nop
 80493a6:       90                      nop
 80493a7:       90                      nop
 80493a8:       90                      nop
 80493a9:       90                      nop
 80493aa:       90                      nop
 80493ab:       90                      nop
 80493ac:       90                      nop
 80493ad:       90                      nop
 80493ae:       90                      nop
 80493af:       90                      nop
```

要进入`<secret_phase>`需要保证 6 个部分全部完成，并且在第 4 部分的输入中再输入字符串`DrEvil`，运行之后就会提示你发现了新东西。这个时候会读入你在 `ans. txt` 中的第 7 行输入。



```
dinghaitong@ubuntu:~/bomb12_202226010304$ gdb -q bomb
Reading symbols from /home/dinghaitong/bomb12_202226010304/bomb...done.
(gdb) r ans.txt
Starting program: /home/dinghaitong/bomb12_202226010304/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...
Congratulations! You've defused the bomb!
[Inferior 1 (process 3377) exited normally]
(gdb) x/s 0x804a358
0x804a358:       "Curses, you've found the secret phase!"
(gdb) x/s 0x804a380
0x804a380:       "But finding it and solving it are quite different..."
(gdb) x/s 0x804a489
0x804a489:       "%d %d %s"
(gdb) x/s 0x804a489
0x804a489:       "%d %d %s"
(gdb) x/s 0x804a492
0x804a492:       "DrEvil"
(gdb)
```



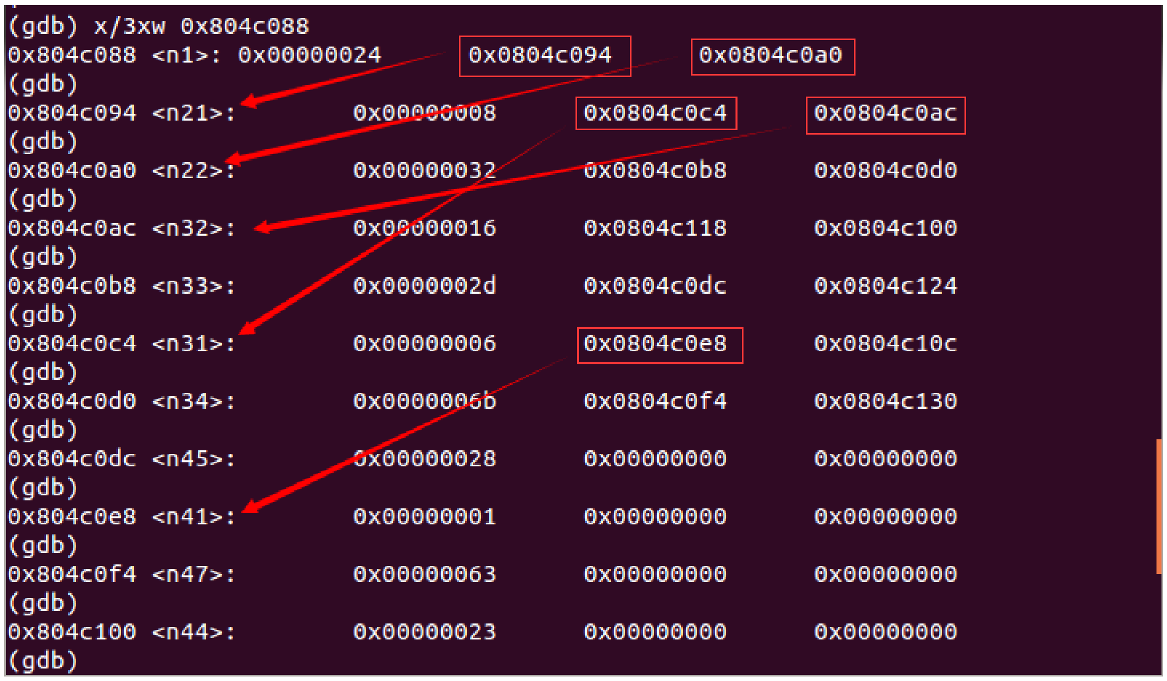
```
Breakpoint 1, main (argc=2, argv=0xbffff3a4) at bomb.c:109
109          phase_defused();
(gdb) stepi
0x080492fb in phase_defused ()
(gdb) p *(int*)0x804c3cc
$1 = 6
(gdb) x/s 0x804c4d0
0x804c4d0 <input_strings+240>:   "8 35 "
(gdb)
```

```
08048f5c <secret_phase>:
 8048f5c:	53                   	push   %ebx
 8048f5d:	83 ec 18             	sub    $0x18,%esp
 8048f60:	e8 38 02 00 00       	call   804919d <read_line> //读第7行
 8048f65:	c7 44 24 08 0a 00 00 	movl   $0xa,0x8(%esp)      // (%esp + 0x8) = 0xa 十进制
 8048f6c:	00 
 8048f6d:	c7 44 24 04 00 00 00 	movl   $0x0,0x4(%esp)      // (%esp + 0x4) = 0x0
 8048f74:	00 
 8048f75:	89 04 24             	mov    %eax,(%esp)         // (%esp) = 第七行
 8048f78:	e8 63 f9 ff ff       	call   80488e0 <strtol@plt> //将字符串转化成十进制数,返回值为%eax
 8048f7d:	89 c3                	mov    %eax,%ebx           // %ebx = %eax
 8048f7f:	8d 40 ff             	lea    -0x1(%eax),%eax     // %eax -= 1
 8048f82:	3d e8 03 00 00       	cmp    $0x3e8,%eax         // %eax ? 0x3e8
 8048f87:	76 05                	jbe    8048f8e <secret_phase+0x32> //  0 < 输入值ln7 <= 0x3e9 = 1001
 8048f89:	e8 e8 01 00 00       	call   8049176 <explode_bomb>
 8048f8e:	89 5c 24 04          	mov    %ebx,0x4(%esp)      // (%esp + 0x4) = ln7
 8048f92:	c7 04 24 88 c0 04 08 	movl   $0x804c088,(%esp)   // (%esp) = $0x804c088
 8048f99:	e8 6d ff ff ff       	call   8048f0b <fun7>   fun(ln7, $0x804c088)
 8048f9e:	83 f8 03             	cmp    $0x3,%eax
 8048fa1:	74 05                	je     8048fa8 <secret_phase+0x4c>  //fun7返回值必须为3
 8048fa3:	e8 ce 01 00 00       	call   8049176 <explode_bomb>
 8048fa8:	c7 04 24 7c a2 04 08 	movl   $0x804a27c,(%esp)
 8048faf:       e8 4c f8 ff ff          call   8048800 <puts@plt>
 8048fb4:       e8 42 03 00 00          call   80492fb <phase_defused>
 8048fb9:       83 c4 18                add    $0x18,%esp
 8048fbc:       5b                      pop    %ebx
 8048fbd:       c3                      ret
 8048fbe:       90                      nop
 8048fbf:       90                      nop
```

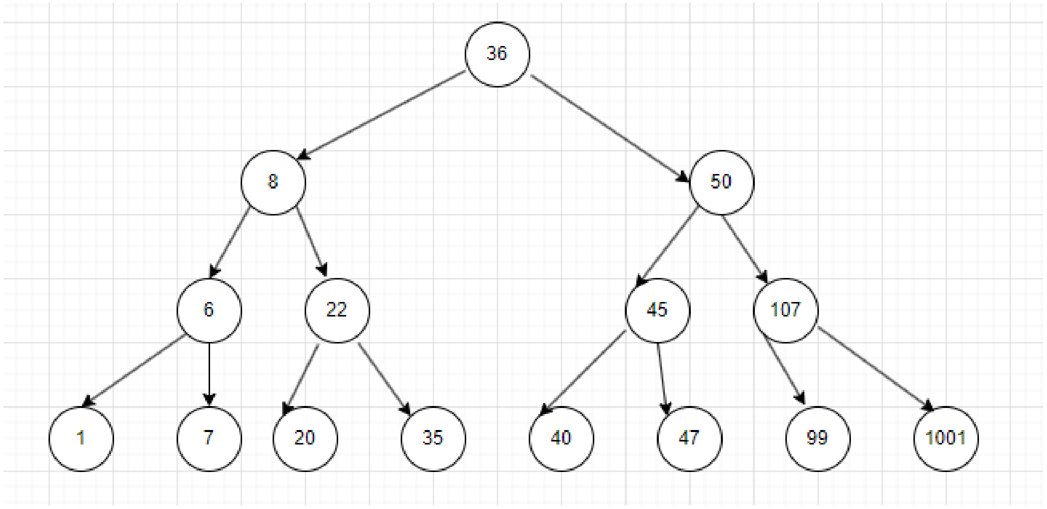
将输入转换成数值，同时确保是在 `[1, 1001]` 中的整数，之后将数值 `ln7`，和地址 `0x804c088` 当作参数传入函数 `<func7>`，返回值必须为 3。

```
08048f0b <fun7>:
 8048f0b:       53                      push   %ebx
 8048f0c:       83 ec 18                sub    $0x18,%esp          //init
 8048f0f:       8b 54 24 20             mov    0x20(%esp),%edx     //%edx = $0x804c088
 8048f13:       8b 4c 24 24             mov    0x24(%esp),%ecx     //%ecx = ln7
 8048f17:       85 d2                   test   %edx,%edx
 8048f19:       74 37                   je     8048f52 <fun7+0x47> //if %edx == 0, return -1
 8048f1b:       8b 1a                   mov    (%edx),%ebx          // %ebx = 当前节点的值 =
node.val
 8048f1d:       39 cb                   cmp    %ecx,%ebx
 8048f1f:       7e 13                   jle    8048f34 <fun7+0x29> //  node.val ? ln7
 8048f21:       89 4c 24 04             mov    %ecx,0x4(%esp) // if node.val > ln7
 8048f25:       8b 42 04                mov    0x4(%edx),%eax
 8048f28:       89 04 24                mov    %eax,(%esp)
 8048f2b:       e8 db ff ff ff          call   8048f0b <fun7>       // fun7(ln7, node.left)
 8048f30:       01 c0                   add    %eax,%eax            // return 2*fun7(ln7,
node.left)
 8048f32:       eb 23                   jmp    8048f57 <fun7+0x4c>
 8048f34:       b8 00 00 00 00          mov    $0x0,%eax       //  if node.val <= ln7, %eax = 0
 8048f39:       39 cb                   cmp    %ecx,%ebx            // node.val ? ln7
 8048f3b:       74 1a                   je     8048f57 <fun7+0x4c> // if node.val == ln7,
return 0
 8048f3d:       89 4c 24 04             mov    %ecx,0x4(%esp)       // if node.val < ln7
 8048f41:       8b 42 08                mov    0x8(%edx),%eax
 8048f44:       89 04 24                mov    %eax,(%esp)
 8048f47:       e8 bf ff ff ff          call   8048f0b <fun7>       // fun7(ln7, node.right)
 8048f4c:       8d 44 00 01             lea    0x1(%eax,%eax,1),%eax
 8048f50:       eb 05                   jmp    8048f57 <fun7+0x4c> // return 2*fun7(ln7,
node.right) + 1
 8048f52:       b8 ff ff ff ff          mov    $0xffffffff,%eax
 8048f57:       83 c4 18                add    $0x18,%esp
 8048f5a:       5b                      pop    %ebx
 8048f5b:       c3                      ret
```

`<func7>` 是一个递归函数，传入的参数 `0x804c088` 是一棵二叉搜索树的首地址，每个节点 12 字节，`{data, left, rigt}`：



```
(gdb) x/3xw 0x804c088
0x804c088 <n1>: 0x00000024      0x0804c094      0x0804c0a0
(gdb)
0x804c094 <n21>:        0x00000008      0x0804c0c4      0x0804c0ac
(gdb)
0x804c0a0 <n22>:        0x00000032      0x0804c0b8      0x0804c0d0
(gdb)
0x804c0ac <n32>:        0x00000016      0x0804c118      0x0804c100
(gdb)
0x804c0b8 <n33>:        0x0000002d      0x0804c0dc      0x0804c124
(gdb)
0x804c0c4 <n31>:        0x00000006      0x0804c0e8      0x0804c10c
(gdb)
0x804c0d0 <n34>:        0x0000006b      0x0804c0f4      0x0804c130
(gdb)
0x804c0dc <n45>:        0x00000028      0x00000000      0x00000000
(gdb)
0x804c0e8 <n41>:        0x00000001      0x00000000      0x00000000
(gdb)
0x804c0f4 <n47>:        0x00000063      0x00000000      0x00000000
(gdb)
0x804c100 <n44>:        0x00000023      0x00000000      0x00000000
(gdb)
```



反推出 `<func7>` 的 C 代码如下：

```
int fun7(int tar, int addr)
{
        int l = 2 * addr;
        int r = 2 * addr + 1;
        if(arr_Node[addr].val == tar) return 0;
        if(arr_Node[addr].val < tar) return 2 * fun7(tar, r) + 1;
        if(arr_Node[addr].val > tar) return 2 * fun7(tar, l);
}
```

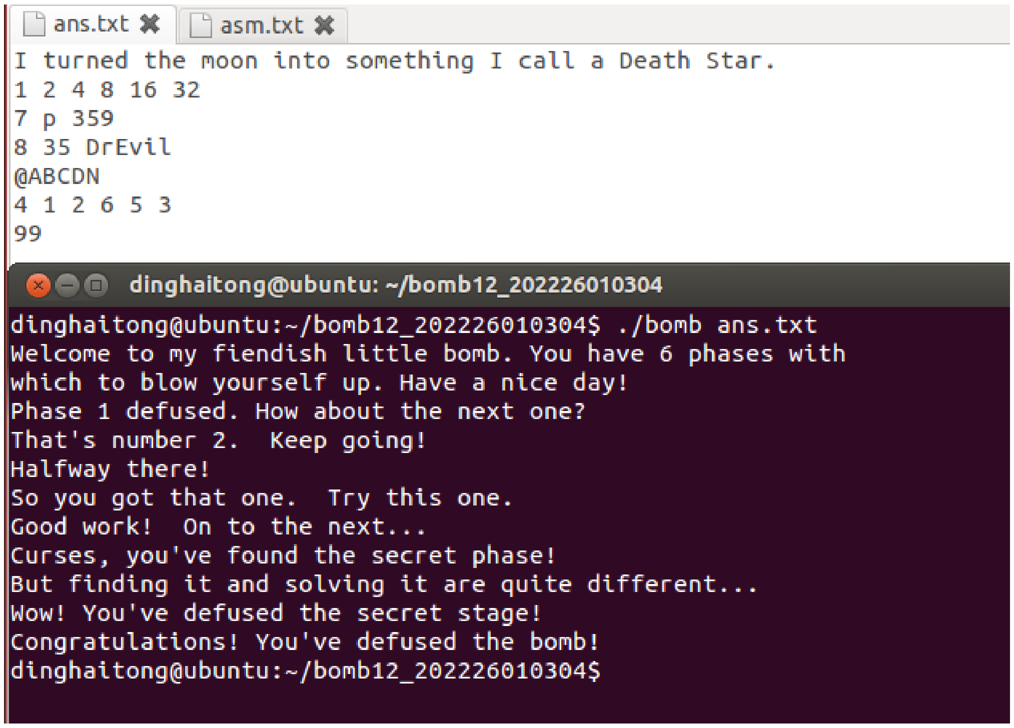
此时我们要求解 `func7(tar, 0x804c088) == 3`，可以通过以下程序：

```
#include <iostream>

using namespace std;

struct Node
```

```c
{
        int val;
        int left;
        int right;
};

int arr_val[16] = {
        0,
        36,
        8,
        50,
        6,
        22,
        45,
        107,
        1,
        7,
        20,
        35,
        40,
        47,
        99,
        1001
};
Node arr_Node[16];

void insert()
{
        for(int i = 1; i <= 15; i++)
        {
                arr_Node[i] = {arr_val[i], 2*i, (2*i + 1)};
        }

}
void print_all_node()
{
        int pau = 1;
        for(int i = 1; i <= 15; i++)
        {
                printf("%d ", arr_Node[i].val);
                if(i == pau)
                {
                        printf("\n");
                        pau = 2 * pau + 1;
                }



        }
}

int fun7(int tar, int addr)
```

```
{
        int l = 2 * addr;
        int r = 2 * addr + 1;
        if(arr_Node[addr].val == tar) return 0;
        if(arr_Node[addr].val < tar) return 2 * fun7(tar, r) + 1;
        if(arr_Node[addr].val > tar) return 2 * fun7(tar, l);
}

int main()
{
        insert();
//      print_all_node();

        for(int i = 1; i <= 15; i++)
        {
                if(fun7(arr_val[i], 1) == 3)
                        cout << arr_val[i] << endl;
        }
        return 0;
}
```

运行之后可以得到，ans = 99 or 107
验证正确：



```
I turned the moon into something I call a Death Star.
1 2 4 8 16 32
7 p 359
8 35 DrEvil
@ABCDN
4 1 2 6 5 3
99
```

```
dinghaitong@ubuntu:~/bomb12_202226010304$ ./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!
dinghaitong@ubuntu:~/bomb12_202226010304$
```

> ***the ans of secret_phase： 99

三、总结

1. 实验过程中出现的问题

- 对于调用函数过程中栈帧空间不熟悉
- GDB 工具不熟练，许多指令不清楚
- 多重循环所对应的含义不能总结到位
- 对于地址和地址指向的数据经常搞混

2. 收获

- 7 个阶段全部完成的成就感是无可比拟的
- 对于汇编代码不再恐惧
- 对于调用函数、循环、数组等概念理解更加深刻