CSAPP：bomblab 实验报告

姓名：施楚峰

学号：PB18000335

背景故事

- Dr. Evil’s Insidious Bomb, Version 1.1
- Copyright 2011, Dr. Evil Incorporated. All rights reserved.
  *
- LICENSE:
  *
- Dr. Evil Incorporated (the PERPETRATOR) hereby grants you (the
- VICTIM) explicit permission to use this bomb (the BOMB). This is a
- time limited license, which expires on the death of the VICTIM.
- The PERPETRATOR takes no responsibility for damage, frustration,
- insanity, bug-eyes, carpal-tunnel syndrome, loss of sleep, or other
- harm to the VICTIM. Unless the PERPETRATOR wants to take credit,
- that is. The VICTIM may not distribute this bomb source code to
- any enemies of the PERPETRATOR. No VICTIM may debug,
- reverse-engineer, run “strings” on, decompile, decrypt, or use any
- other technique to gain knowledge of and defuse the BOMB. BOMB
- proof clothing may not be worn when handling this program. The
- PERPETRATOR will not apologize for the PERPETRATOR’s poor sense of
- humor. This license is null and void where the BOMB is prohibited
- by law.

> 实验目标就是通过在终端中输入特定的 数值 或 字符串 ， 来拆除炸弹，进入下一关卡或完成挑战。

实验准备

先执行 `objdump -d bomb > asm.txt` 将 `bomb` 文件反编译并存入 `asm.txt` 文件中。经过粗略地观察，我们发现与实验相关的函数有

- main
- phase_1
- phase_2
- phase_3
- fuc_4
- phase_4
- phase_5
- phase_6
- fuc_7
- secret_phase
- string_length
- strings_not_equal
- explode_bomb
- read_six_numbers
- read_line
- phase_defused

read_line 和 sscanf

通过搜索我们可以知道

```
int sscanf(const char *buffer, const char *format, [ argument ] ... );
```

因此任何对 `sscanf` 的调用都必须含有三个以上参数，再结合编译器使用寄存器的顺序可以知道，寄存器 `%edi` 对应于 `buffer` 指针， `%esi` 对应于 `format` 指针。因而从函数名以及指令中我们可以推测出，程序先通过 `read_line` 函数读出一行字符串并将指针存入 `%edi` 中，再通过调用 `sscanf` 函数从字符串中读取对应格式的数据。

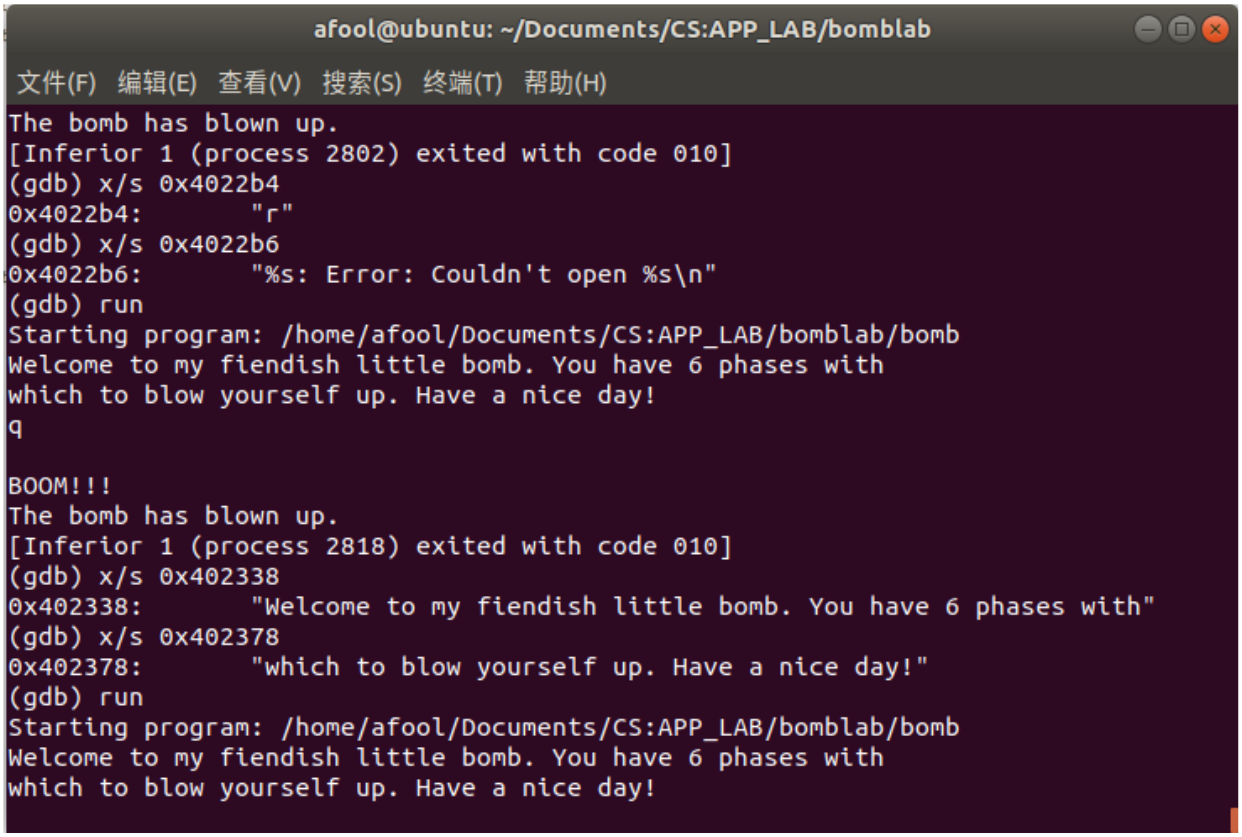
main

下为精简代码

```
0000000000400da0 <main>:

  400e1e:   bf 38 23 40 00          mov    $0x402338,%edi
  400e23:   e8 e8 fc ff ff          callq  400b10 <puts@plt>
  400e28:   bf 78 23 40 00          mov    $0x402378,%edi
  400e2d:   e8 de fc ff ff          callq  400b10 <puts@plt>

  400e32:   e8 67 06 00 00          callq  40149e <read_line>
  400e37:   48 89 c7                mov    %rax,%rdi
  400e3a:   e8 a1 00 00 00          callq  400ee0 <phase_1>
  400e3f:   e8 80 07 00 00          callq  4015c4 <phase_defused>
  400e44:   bf a8 23 40 00          mov    $0x4023a8,%edi
```

```
400e49:   e8 c2 fc ff ff          callq  400b10 <puts@plt>

400e4e:   e8 4b 06 00 00          callq  40149e <read_line>
400e53:   48 89 c7                mov    %rax,%rdi
400e56:   e8 a1 00 00 00          callq  400efc <phase_2>
400e5b:   e8 64 07 00 00          callq  4015c4 <phase_defused>
400e60:   bf ed 22 40 00          mov    $0x4022ed,%edi
400e65:   e8 a6 fc ff ff          callq  400b10 <puts@plt>

400e6a:   e8 2f 06 00 00          callq  40149e <read_line>
400e6f:   48 89 c7                mov    %rax,%rdi
400e72:   e8 cc 00 00 00          callq  400f43 <phase_3>
400e77:   e8 48 07 00 00          callq  4015c4 <phase_defused>
400e7c:   bf 0b 23 40 00          mov    $0x40230b,%edi
400e81:   e8 8a fc ff ff          callq  400b10 <puts@plt>

400e86:   e8 13 06 00 00          callq  40149e <read_line>
400e8b:   48 89 c7                mov    %rax,%rdi
400e8e:   e8 79 01 00 00          callq  40100c <phase_4>
400e93:   e8 2c 07 00 00          callq  4015c4 <phase_defused>
400e98:   bf d8 23 40 00          mov    $0x4023d8,%edi
400e9d:   e8 6e fc ff ff          callq  400b10 <puts@plt>

400ea2:   e8 f7 05 00 00          callq  40149e <read_line>
400ea7:   48 89 c7                mov    %rax,%rdi
400eaa:   e8 b3 01 00 00          callq  401062 <phase_5>
400eaf:   e8 10 07 00 00          callq  4015c4 <phase_defused>
400eb4:   bf 1a 23 40 00          mov    $0x40231a,%edi
400eb9:   e8 52 fc ff ff          callq  400b10 <puts@plt>

400ebe:   e8 db 05 00 00          callq  40149e <read_line>
400ec3:   48 89 c7                mov    %rax,%rdi
400ec6:   e8 29 02 00 00          callq  4010f4 <phase_6>
400ecb:   e8 f4 06 00 00          callq  4015c4 <phase_defused>
```

先对 `main` 函数观察，发现函数开始时输出一些字符串，`x/s` 命令查看相应内存中存储的字符串发现就是程序开始运行时的输出。



继续观察发现 `main` 函数调用了 `phase_X` `phase_defused` 等函数，可见就是在 `main` 中进行关卡的调用。而每次开启新关卡前，都先读入一行字符串，在关卡中对其进行操作。

string_length / strings_not_equal / read_six_numbers

从函数名称中我们可以推测出函数功能分别为为 求字符串长度 / 判断字符串是否相同 / 读取六个数字。并且我们可以更加详细地知道 `string_not_equal` 在字符串相同时反回 `1`。

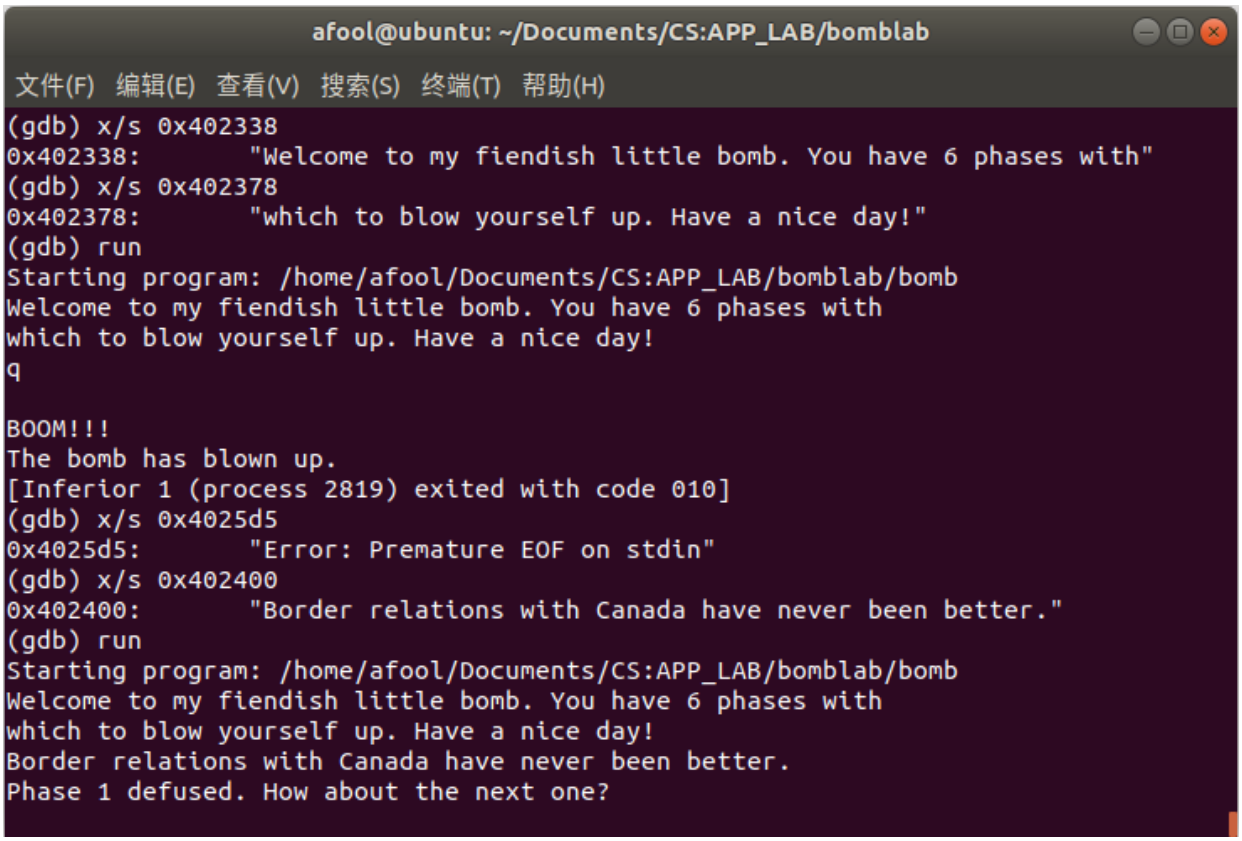
实验过程

phase_1

```
0000000000400ee0 <phase_1>:

 400ee0:   48 83 ec 08             sub    $0x8,%rsp
 400ee4:   be 00 24 40 00          mov    $0x402400,%esi
 400ee9:   e8 4a 04 00 00          callq  401338 <strings_not_equal>
 400eee:   85 c0                   test   %eax,%eax
 400ef0:   74 05                   je     400ef7 <phase_1+0x17>
```

```
 400ef2:   e8 43 05 00 00          callq  40143a <explode_bomb>
 400ef7:   48 83 c4 08             add    $0x8,%rsp
 400efb:   c3                      retq
```

程序简单的将输入与内存中值进行了比对，查看 `0x402400` 处内存即可得到答案。



phase_2

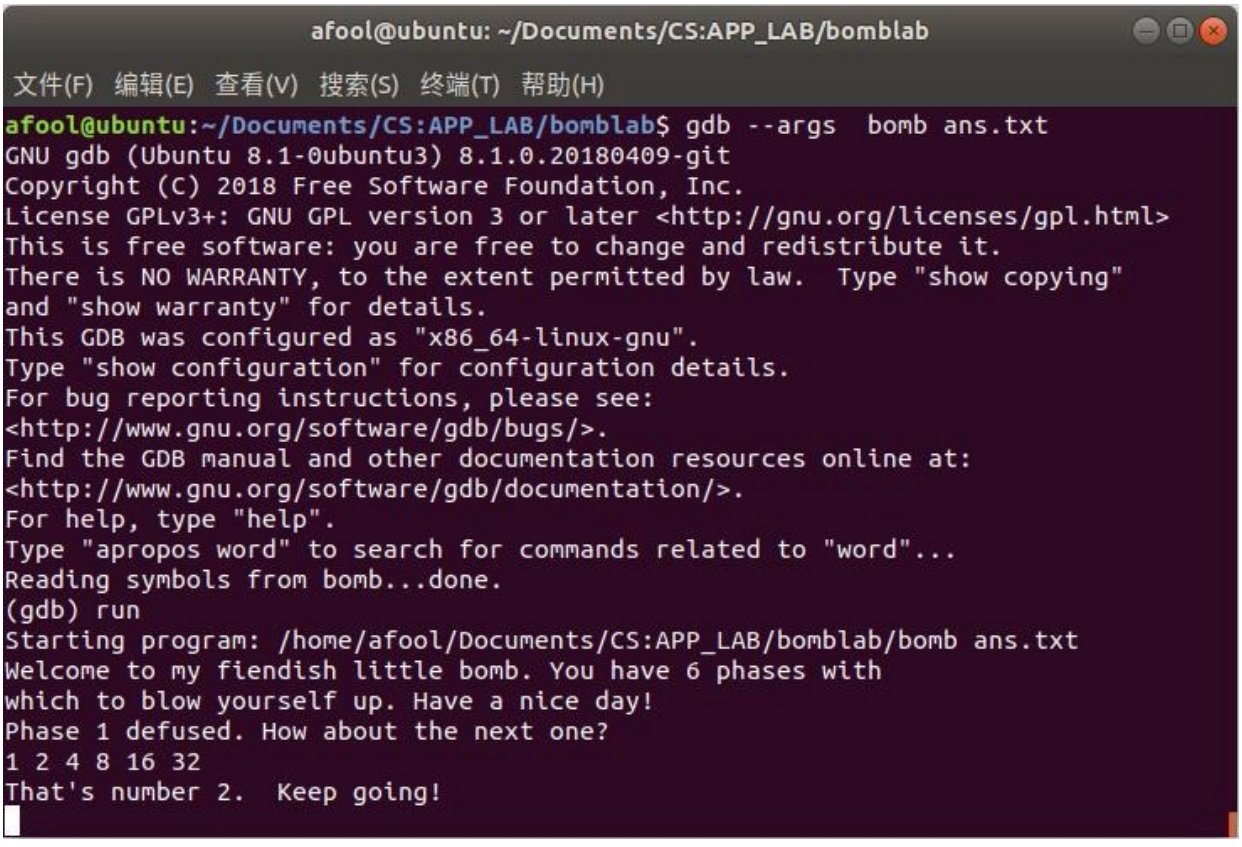
```
0000000000400efc <phase_2>:

 400efc:   55                      push   %rbp
 400efd:   53                      push   %rbx
 400efe:   48 83 ec 28             sub    $0x28,%rsp
 400f02:   48 89 e6                mov    %rsp,%rsi
 400f05:   e8 52 05 00 00          callq  40145c <read_six_numbers>
 400f0a:   83 3c 24 01             cmpl   $0x1,(%rsp)
 400f0e:   74 20                   je     400f30 <phase_2+0x34>
 400f10:   e8 25 05 00 00          callq  40143a <explode_bomb>

 400f15:   eb 19                   jmp    400f30 <phase_2+0x34>
 400f17:   8b 43 fc                mov    -0x4(%rbx),%eax
 400f1a:   01 c0                   add    %eax,%eax
 400f1c:   39 03                   cmp    %eax,(%rbx)
 400f1e:   74 05                   je     400f25 <phase_2+0x29>
 400f20:   e8 15 05 00 00          callq  40143a <explode_bomb>
 400f25:   48 83 c3 04             add    $0x4,%rbx
 400f29:   48 39 eb                cmp    %rbp,%rbx
```

```
  400f2c:   75 e9                    jne    400f17 <phase_2+0x1b>
  400f2e:   eb 0c                    jmp    400f3c <phase_2+0x40>
  400f30:   48 8d 5c 24 04           lea    0x4(%rsp),%rbx
  400f35:   48 8d 6c 24 18           lea    0x18(%rsp),%rbp
  400f3a:   eb db                    jmp    400f17 <phase_2+0x1b>
  400f3c:   48 83 c4 28              add    $0x28,%rsp
  400f40:   5b                       pop    %rbx
  400f41:   5d                       pop    %rbp
  400f42:   c3                       retq
```

函数先读入六个数字，全部存入栈中。

```
  400f0a:   83 3c 24 01              cmpl   $0x1,(%rsp)
  400f0e:   74 20                    je     400f30 <phase_2+0x34>
  400f10:   e8 25 05 00 00           callq  40143a <explode_bomb>
```

可知读入的第一个数应该为 `1` 否则炸弹将被引爆。再接下去是一个循环，进一行分析可以知道指令功能为判断 $a[i] = a[i-1] << 1$ 是否对每一位都成立，那么本题答案就应该是 `1 2 4 8 16 32`



phase_3

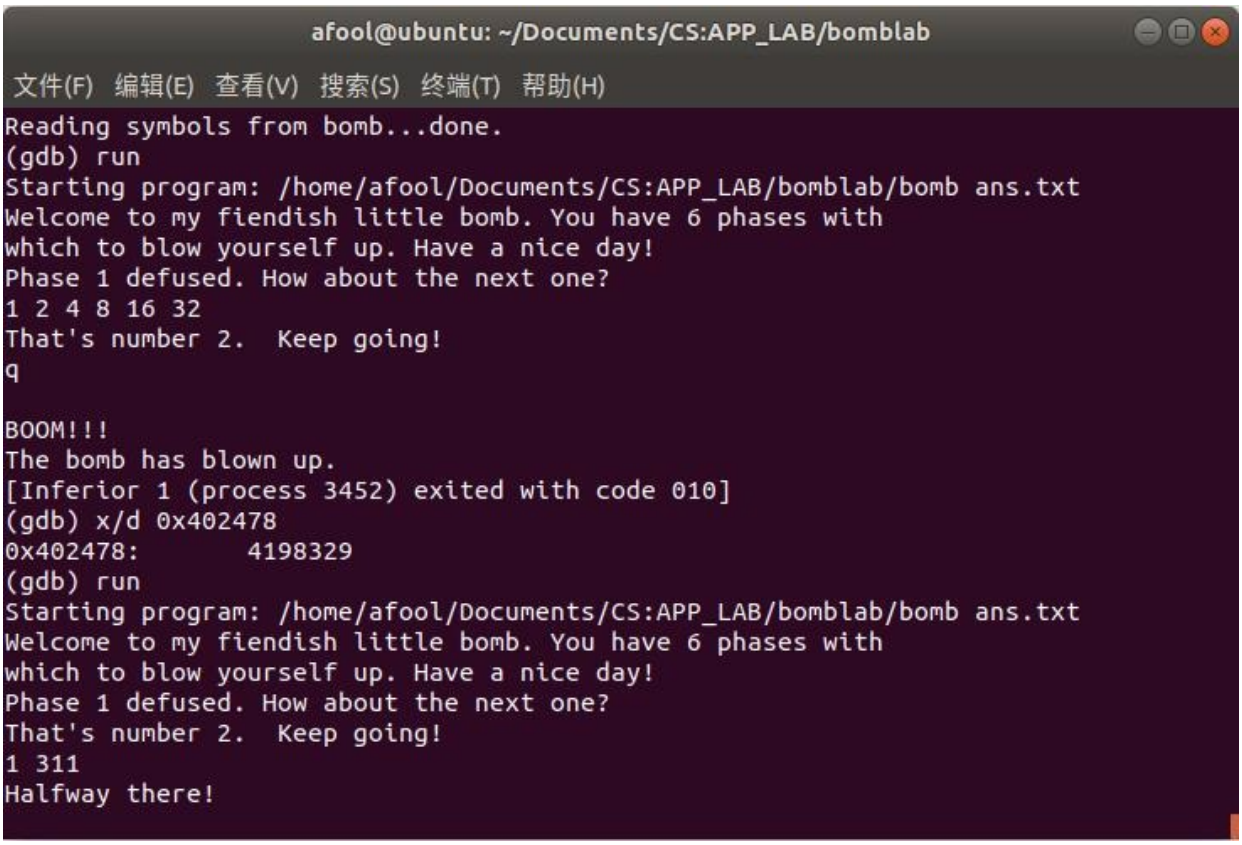
```
0000000000400f43 <phase_3>:

  400f43:   48 83 ec 18              sub    $0x18,%rsp
```

```
400f47:   48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
400f4c:   48 8d 54 24 08          lea    0x8(%rsp),%rdx
400f51:   be cf 25 40 00          mov    $0x4025cf,%esi
400f56:   b8 00 00 00 00          mov    $0x0,%eax
400f5b:   e8 90 fc ff ff          callq  400bf0 <__isoc99_sscanf@plt>

400f60:   83 f8 01                cmp    $0x1,%eax
400f63:   7f 05                   jg     400f6a <phase_3+0x27>
400f65:   e8 d0 04 00 00          callq  40143a <explode_bomb>
400f6a:   83 7c 24 08 07          cmpl   $0x7,0x8(%rsp)
400f6f:   77 3c                   ja     400fad <phase_3+0x6a>

400f71:   8b 44 24 08             mov    0x8(%rsp),%eax
400f75:   ff 24 c5 70 24 40 00    jmpq   *0x402470(,%rax,8)
400f7c:   b8 cf 00 00 00          mov    $0xcf,%eax
400f81:   eb 3b                   jmp    400fbe <phase_3+0x7b>
400f83:   b8 c3 02 00 00          mov    $0x2c3,%eax
400f88:   eb 34                   jmp    400fbe <phase_3+0x7b>
400f8a:   b8 00 01 00 00          mov    $0x100,%eax
400f8f:   eb 2d                   jmp    400fbe <phase_3+0x7b>
400f91:   b8 85 01 00 00          mov    $0x185,%eax
400f96:   eb 26                   jmp    400fbe <phase_3+0x7b>
400f98:   b8 ce 00 00 00          mov    $0xce,%eax
400f9d:   eb 1f                   jmp    400fbe <phase_3+0x7b>
400f9f:   b8 aa 02 00 00          mov    $0x2aa,%eax
400fa4:   eb 18                   jmp    400fbe <phase_3+0x7b>
400fa6:   b8 47 01 00 00          mov    $0x147,%eax
400fab:   eb 11                   jmp    400fbe <phase_3+0x7b>

400fad:   e8 88 04 00 00          callq  40143a <explode_bomb>

400fb2:   b8 00 00 00 00          mov    $0x0,%eax
400fb7:   eb 05                   jmp    400fbe <phase_3+0x7b>
400fb9:   b8 37 01 00 00          mov    $0x137,%eax
400fbe:   3b 44 24 0c             cmp    0xc(%rsp),%eax
400fc2:   74 05                   je     400fc9 <phase_3+0x86>
400fc4:   e8 71 04 00 00          callq  40143a <explode_bomb>
400fc9:   48 83 c4 18             add    $0x18,%rsp
400fcd:   c3                      retq
```

函数先读取两个数字，通过查看内存可以发现读取格式为 `"%d %d"`

```
400f60:   83 f8 01                cmp    $0x1,%eax
400f63:   7f 05                   jg     400f6a <phase_3+0x27>
400f65:   e8 d0 04 00 00          callq  40143a <explode_bomb>
400f6a:   83 7c 24 08 07          cmpl   $0x7,0x8(%rsp)
400f6f:   77 3c                   ja     400fad <phase_3+0x6a>
...
00fad:    e8 88 04 00 00          callq  40143a <explode_bomb>
```

可知对第一个数 `a` 的限制为 `1 <= a <= 7` ，再对接下来的逻辑快进行分析，发现程序跳转到相对 `0x402470` 的地方偏移 `8 * a` 内存处存储的地址上，对所有的 `1` 到 `7` 的数进行枚举，发现当 `a = 1` 时，地址 `0x402478` 处值为 `4198329` ，而又有$4198329D = 0X400FB9H$，
当 `a` 为 `1` 时，直接跳转到了下一个语块内，稍做分析就可以推测出 `b` 应为 `311` 。



phase_4

```
0000000000400fce <func4>:

  400fce:   48 83 ec 08             sub    $0x8,%rsp
  400fd2:   89 d0                   mov    %edx,%eax
  400fd4:   29 f0                   sub    %esi,%eax
  400fd6:   89 c1                   mov    %eax,%ecx
  400fd8:   c1 e9 1f                shr    $0x1f,%ecx
  400fdb:   01 c8                   add    %ecx,%eax
  400fdd:   d1 f8                   sar    %eax
  400fdf:   8d 0c 30                lea    (%rax,%rsi,1),%ecx
  400fe2:   39 f9                   cmp    %edi,%ecx
  400fe4:   7e 0c                   jle    400ff2 <func4+0x24>
  400fe6:   8d 51 ff                lea    -0x1(%rcx),%edx
  400fe9:   e8 e0 ff ff ff          callq  400fce <func4>
  400fee:   01 c0                   add    %eax,%eax
  400ff0:   eb 15                   jmp    401007 <func4+0x39>
  400ff2:   b8 00 00 00 00          mov    $0x0,%eax
  400ff7:   39 f9                   cmp    %edi,%ecx
  400ff9:   7d 0c                   jge    401007 <func4+0x39>
  400ffb:   8d 71 01                lea    0x1(%rcx),%esi
```

```
  400ffe:   e8 cb ff ff ff          callq  400fce <func4>
  401003:   8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
  401007:   48 83 c4 08             add    $0x8,%rsp
  40100b:   c3                      retq

000000000040100c <phase_4>:

  40100c:   48 83 ec 18             sub    $0x18,%rsp
  401010:   48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
  401015:   48 8d 54 24 08          lea    0x8(%rsp),%rdx
  40101a:   be cf 25 40 00          mov    $0x4025cf,%esi
  40101f:   b8 00 00 00 00          mov    $0x0,%eax
  401024:   e8 c7 fb ff ff          callq  400bf0 <__isoc99_sscanf@plt>
  401029:   83 f8 02                cmp    $0x2,%eax
  40102c:   75 07                   jne    401035 <phase_4+0x29>
  40102e:   83 7c 24 08 0e          cmpl   $0xe,0x8(%rsp)
  401033:   76 05                   jbe    40103a <phase_4+0x2e>
  401035:   e8 00 04 00 00          callq  40143a <explode_bomb>

  40103a:   ba 0e 00 00 00          mov    $0xe,%edx
  40103f:   be 00 00 00 00          mov    $0x0,%esi
  401044:   8b 7c 24 08             mov    0x8(%rsp),%edi
  401048:   e8 81 ff ff ff          callq  400fce <func4>
  40104d:   85 c0                   test   %eax,%eax
  40104f:   75 07                   jne    401058 <phase_4+0x4c>
  401051:   83 7c 24 0c 00          cmpl   $0x0,0xc(%rsp)
  401056:   74 05                   je     40105d <phase_4+0x51>
  401058:   e8 dd 03 00 00          callq  40143a <explode_bomb>
  40105d:   48 83 c4 18             add    $0x18,%rsp
  401061:   c3                      retq
```

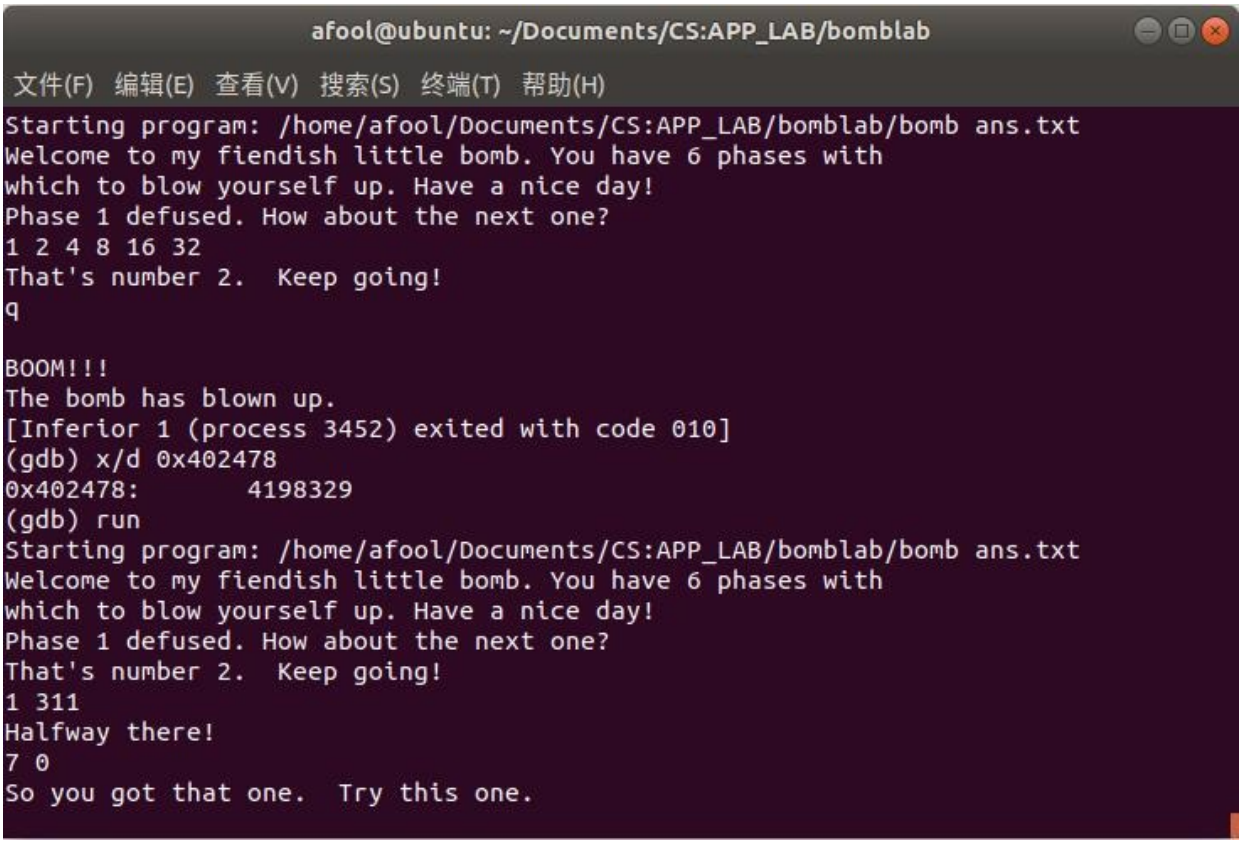
首先分析 `phase_4` 函数，发现它输入了两个数字，其中对第一个数字有一个取值范围的要求 $[0, 14]$。在调用 `fuc4` 函数后，判断函数返回值是否为 `0`，是则成功拆弹。继续对 `fuc4` 进行分析，发现函数在执行类似二分法的操作，在对代码仔细分析后，可以得出类似的函数模型。

> 由于输入范围限制，可以忽略一些细节

```
int fuc4(int k, int l, int r)
{
    int mid = l + r >> 1;
    if (k < mid)
    {
        r = mid - 1;
        return fuc4(k, l, r) << 1;
    }
    else {
        if (k == mid) return 0;
        return (func4(mid + 1, r) << 1) | 1;
    }
```

```
}
```

要想使其反回 `0` ,最简单的办法就是输入 `a` 为 `7` ， 通过后续可以判断 'b' 应该为 `0` 。

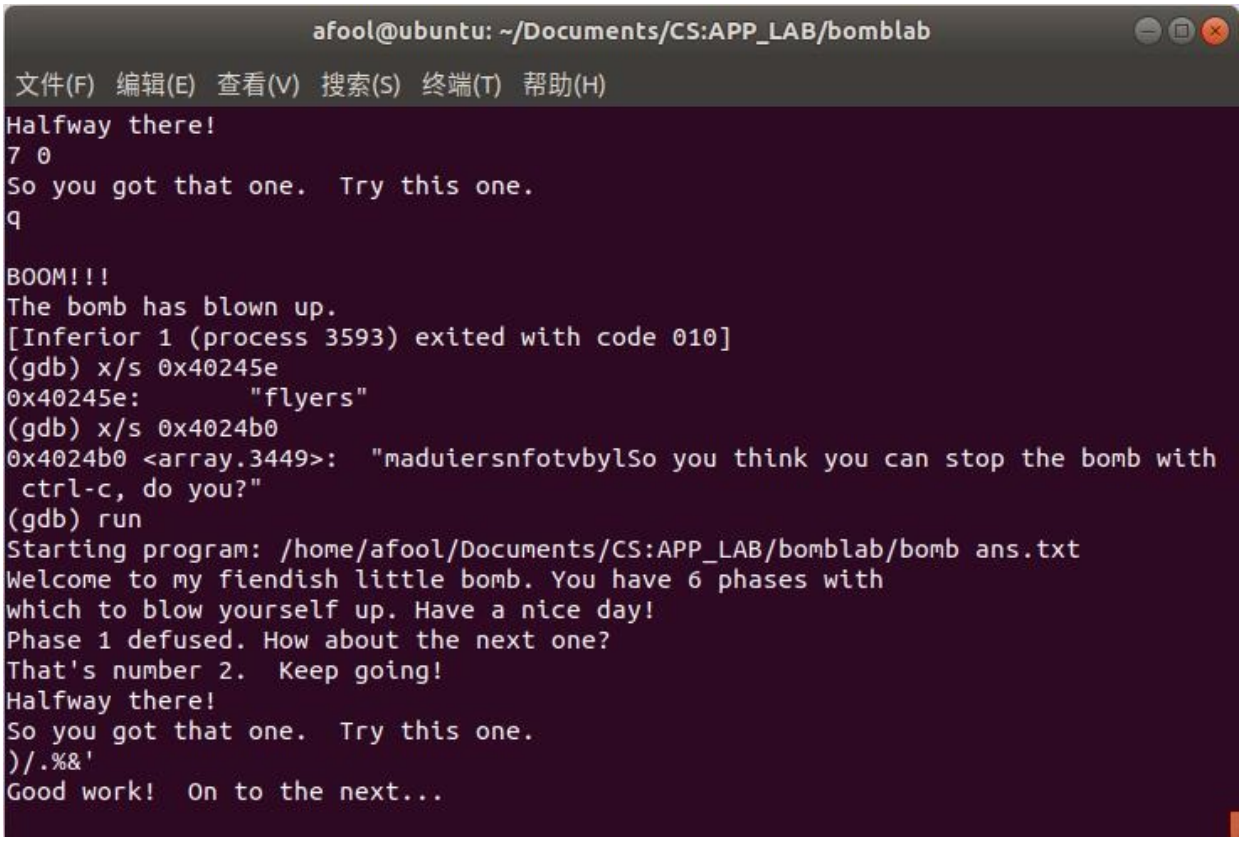


phase_5

```
0000000000401062 <phase_5>:

  401062:   53                      push   %rbx
  401063:   48 83 ec 20             sub    $0x20,%rsp
  401067:   48 89 fb                mov    %rdi,%rbx
  40106a:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  401071:   00 00
  401073:   48 89 44 24 18          mov    %rax,0x18(%rsp)
  401078:   31 c0                   xor    %eax,%eax
  40107a:   e8 9c 02 00 00          callq  40131b <string_length>
  40107f:   83 f8 06                cmp    $0x6,%eax
  401082:   74 4e                   je     4010d2 <phase_5+0x70>
  401084:   e8 b1 03 00 00          callq  40143a <explode_bomb>

  401089:   eb 47                   jmp    4010d2 <phase_5+0x70>
  40108b:   0f b6 0c 03             movzbl (%rbx,%rax,1),%ecx
  40108f:   88 0c 24                mov    %cl,(%rsp)
  401092:   48 8b 14 24             mov    (%rsp),%rdx
  401096:   83 e2 0f                and    $0xf,%edx
  401099:   0f b6 92 b0 24 40 00    movzbl 0x4024b0(%rdx),%edx
  4010a0:   88 54 04 10             mov    %dl,0x10(%rsp,%rax,1)
```

```
4010a4:   48 83 c0 01              add    $0x1,%rax
4010a8:   48 83 f8 06              cmp    $0x6,%rax
4010ac:   75 dd                    jne    40108b <phase_5+0x29>
4010ae:   c6 44 24 16 00           movb   $0x0,0x16(%rsp)

4010b3:   be 5e 24 40 00           mov    $0x40245e,%esi
4010b8:   48 8d 7c 24 10           lea    0x10(%rsp),%rdi
4010bd:   e8 76 02 00 00           callq  401338 <strings_not_equal>
4010c2:   85 c0                    test   %eax,%eax
4010c4:   74 13                    je     4010d9 <phase_5+0x77>
4010c6:   e8 6f 03 00 00           callq  40143a <explode_bomb>
4010cb:   0f 1f 44 00 00           nopl   0x0(%rax,%rax,1)
4010d0:   eb 07                    jmp    4010d9 <phase_5+0x77>
4010d2:   b8 00 00 00 00           mov    $0x0,%eax
4010d7:   eb b2                    jmp    40108b <phase_5+0x29>
4010d9:   48 8b 44 24 18           mov    0x18(%rsp),%rax
4010de:   64 48 33 04 25 28 00     xor    %fs:0x28,%rax
4010e5:   00 00
4010e7:   74 05                    je     4010ee <phase_5+0x8c>
4010e9:   e8 42 fa ff ff           callq  400b30 <__stack_chk_fail@plt>
4010ee:   48 83 c4 20              add    $0x20,%rsp
4010f2:   5b                       pop    %rbx
4010f3:   c3                       retq
```

先把程序分段，观察第一段，发现程序读了长度为 `6` 的一个字符串，第二段比较复杂先略过直接看第三段。发现第三段程序对两个字符串进行了比较，通过查看内存我们发现，其中一个字符串是 `flyers` ，而仅有当另一个字符串与之相同时才拆弹成功，接下来看另一个字符串是怎么得来的。第二部分中，我们发现程序对我们输入的字符串进行了一个映射，是取每个字符 `Ascii` 码的低 `16` 位相对内存 `0x4024b0` 的偏移，通过查看发现该处存储了 `"maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"` ,只需要构造一个字符串进行相应映射能构成 `flyers` 即可。



phase_6

下为精简代码

```
00000000004010f4 <phase_6>:

  401106:   e8 51 03 00 00          callq  40145c <read_six_numbers>
  40110b:   49 89 e6                mov    %rsp,%r14
  40110e:   41 bc 00 00 00 00       mov    $0x0,%r12d

  401114:   4c 89 ed                mov    %r13,%rbp
  401117:   41 8b 45 00             mov    0x0(%r13),%eax
  40111b:   83 e8 01                sub    $0x1,%eax
  40111e:   83 f8 05                cmp    $0x5,%eax
  401121:   76 05                   jbe    401128 <phase_6+0x34>
  401123:   e8 12 03 00 00          callq  40143a <explode_bomb>
  401128:   41 83 c4 01             add    $0x1,%r12d
  40112c:   41 83 fc 06             cmp    $0x6,%r12d
  401130:   74 21                   je     401153 <phase_6+0x5f>
  401132:   44 89 e3                mov    %r12d,%ebx
  401135:   48 63 c3                movslq %ebx,%rax
  401138:   8b 04 84                mov    (%rsp,%rax,4),%eax
  40113b:   39 45 00                cmp    %eax,0x0(%rbp)
  40113e:   75 05                   jne    401145 <phase_6+0x51>
  401140:   e8 f5 02 00 00          callq  40143a <explode_bomb>
  401145:   83 c3 01                add    $0x1,%ebx
  401148:   83 fb 05                cmp    $0x5,%ebx
  40114b:   7e e8                   jle    401135 <phase_6+0x41>
```

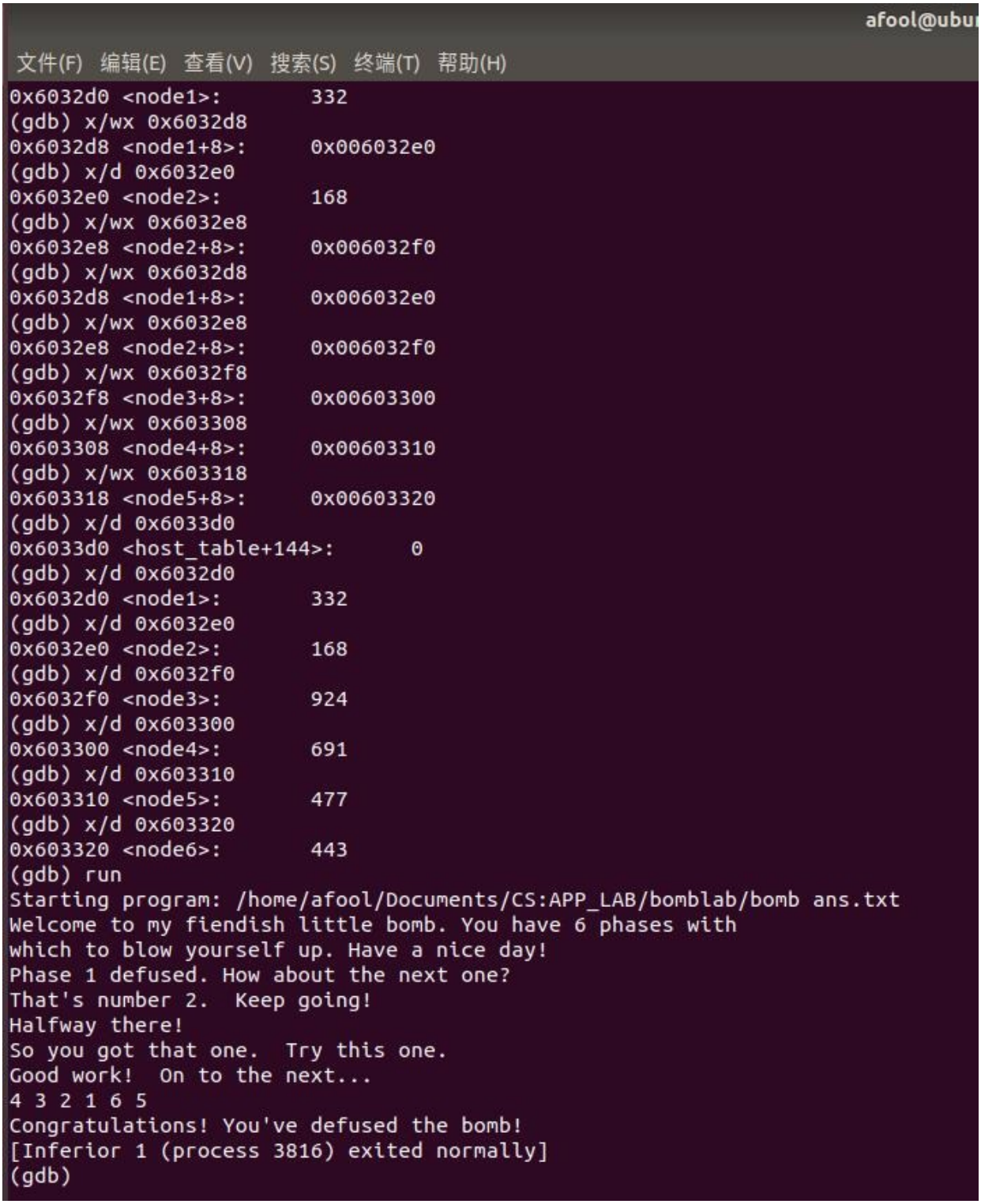
```
40114d:   49 83 c5 04            add    $0x4,%r13
401151:   eb c1                  jmp    401114 <phase_6+0x20>

401153:   48 8d 74 24 18         lea    0x18(%rsp),%rsi
401158:   4c 89 f0               mov    %r14,%rax
40115b:   b9 07 00 00 00         mov    $0x7,%ecx

401160:   89 ca                  mov    %ecx,%edx
401162:   2b 10                  sub    (%rax),%edx
401164:   89 10                  mov    %edx,(%rax)
401166:   48 83 c0 04            add    $0x4,%rax
40116a:   48 39 f0               cmp    %rsi,%rax
40116d:   75 f1                  jne    401160 <phase_6+0x6c>

40116f:   be 00 00 00 00         mov    $0x0,%esi
401174:   eb 21                  jmp    401197 <phase_6+0xa3>
401176:   48 8b 52 08            mov    0x8(%rdx),%rdx
40117a:   83 c0 01               add    $0x1,%eax
40117d:   39 c8                  cmp    %ecx,%eax
40117f:   75 f5                  jne    401176 <phase_6+0x82>
401181:   eb 05                  jmp    401188 <phase_6+0x94>
401183:   ba d0 32 60 00         mov    $0x6032d0,%edx
401188:   48 89 54 74 20         mov    %rdx,0x20(%rsp,%rsi,2)
40118d:   48 83 c6 04            add    $0x4,%rsi
401191:   48 83 fe 18            cmp    $0x18,%rsi
401195:   74 14                  je     4011ab <phase_6+0xb7>
401197:   8b 0c 34               mov    (%rsp,%rsi,1),%ecx
40119a:   83 f9 01               cmp    $0x1,%ecx
40119d:   7e e4                  jle    401183 <phase_6+0x8f>
40119f:   b8 01 00 00 00         mov    $0x1,%eax
4011a4:   ba d0 32 60 00         mov    $0x6032d0,%edx
4011a9:   eb cb                  jmp    401176 <phase_6+0x82>

4011ab:   48 8b 5c 24 20         mov    0x20(%rsp),%rbx
4011b0:   48 8d 44 24 28         lea    0x28(%rsp),%rax
4011b5:   48 8d 74 24 50         lea    0x50(%rsp),%rsi
4011ba:   48 89 d9               mov    %rbx,%rcx
4011bd:   48 8b 10               mov    (%rax),%rdx
4011c0:   48 89 51 08            mov    %rdx,0x8(%rcx)
4011c4:   48 83 c0 08            add    $0x8,%rax
4011c8:   48 39 f0               cmp    %rsi,%rax
4011cb:   74 05                  je     4011d2 <phase_6+0xde>
4011cd:   48 89 d1               mov    %rdx,%rcx
4011d0:   eb eb                  jmp    4011bd <phase_6+0xc9>
4011d2:   48 c7 42 08 00 00 00   movq   $0x0,0x8(%rdx)
4011d9:   00
4011da:   bd 05 00 00 00         mov    $0x5,%ebp
4011df:   48 8b 43 08            mov    0x8(%rbx),%rax
4011e3:   8b 00                  mov    (%rax),%eax
4011e5:   39 03                  cmp    %eax,(%rbx)
4011e7:   7d 05                  jge    4011ee <phase_6+0xfa>
4011e9:   e8 4c 02 00 00         callq  40143a <explode_bomb>
```

```
4011ee:   48 8b 5b 08              mov    0x8(%rbx),%rbx
4011f2:   83 ed 01                 sub    $0x1,%ebp
4011f5:   75 e8                    jne    4011df <phase_6+0xeb>
```

分段分析可以推断出，函数读入了 `6` 个数字保存进栈中，并且输入应当保证是 `1-6` 的一个全排列，之后每个数字各自赋值为 `7` 减去自身。在之后的一个段比较难以理解，不过仔细分析之后，可以发现是一个链表一样的工作方式，类似于结构体

```
struct P{
    int num;
    P *next;
};
```

而判断炸弹是否爆炸的逻辑是要求满足 `p[a[i]].num >= p[a[i + 1]].num` ，其中 `p` 数组由 `next` 遍历一遍后存储在栈中。通过将各个值输出再构造数据即可拆除炸弹。



secret_phase

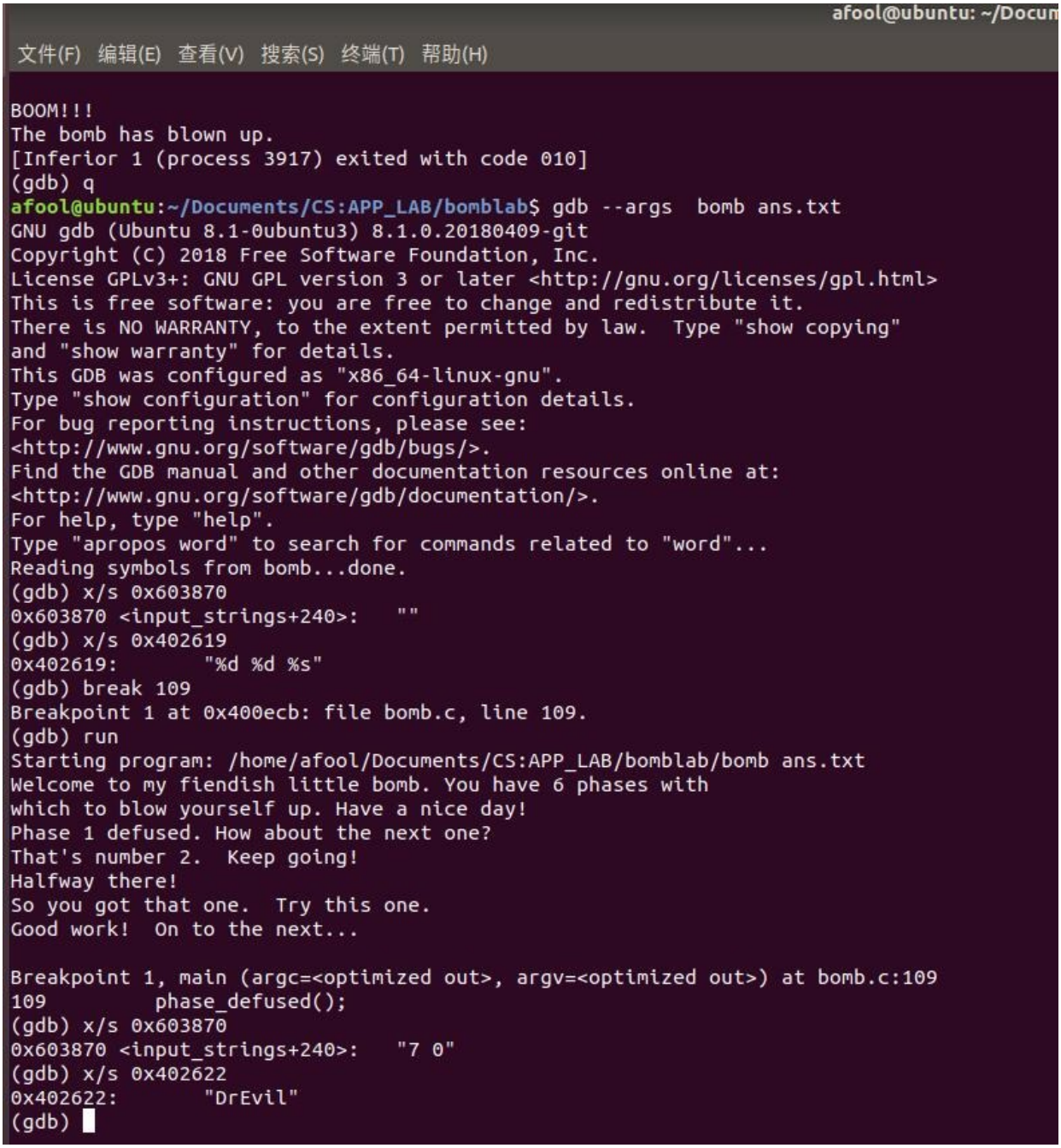
一开始我们便知道程序还有一个隐藏关卡，但是完成六个关卡后程序就直接结束了，那到底该如何进入呢?

从 `phase_defused` 函数中可以找到对 `secret_phase` 函数的调用。

```
  4015d8:   83 3d 81 21 20 00 06    cmpl   $0x6,0x202181(%rip)        # 603760 <
num_input_strings>
  4015df:   75 5e                   jne    40163f <phase_defused+0x7b>
```

```
4015e1:   4c 8d 44 24 10          lea    0x10(%rsp),%r8
4015e6:   48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
4015eb:   48 8d 54 24 08          lea    0x8(%rsp),%rdx
4015f0:   be 19 26 40 00          mov    $0x402619,%esi
4015f5:   bf 70 38 60 00          mov    $0x603870,%edi
4015fa:   e8 f1 f5 ff ff          callq  400bf0 <__isoc99_sscanf@plt>
4015ff:   83 f8 03                cmp    $0x3,%eax
401602:   75 31                   jne    401635 <phase_defused+0x71>
401604:   be 22 26 40 00          mov    $0x402622,%esi
401609:   48 8d 7c 24 10          lea    0x10(%rsp),%rdi
40160e:   e8 25 fd ff ff          callq  401338 <strings_not_equal>
401613:   85 c0                   test   %eax,%eax
401615:   75 1e                   jne    401635 <phase_defused+0x71>
401617:   bf f8 24 40 00          mov    $0x4024f8,%edi
40161c:   e8 ef f4 ff ff          callq  400b10 <puts@plt>
401621:   bf 20 25 40 00          mov    $0x402520,%edi
401626:   e8 e5 f4 ff ff          callq  400b10 <puts@plt>
40162b:   b8 00 00 00 00          mov    $0x0,%eax
401630:   e8 0d fc ff ff          callq  401242 <secret_phase>
401635:   bf 58 25 40 00          mov    $0x402558,%edi
40163a:   e8 d1 f4 ff ff          callq  400b10 <puts@plt>
```

发现函数在 `0x603760` 处有一个计数器，当完成六个关卡后开始判断是否启动隐藏关卡。观察后发现，程序调用了 `sscanf` 函数，以 `0x402619` `0x603870` 地址为参数，尝试性查看内存后发现，读取格式为 `"%d %d %s"` 但是 `0x603870` 处却为空！继续执行程序后会将读出的字符串与 `0x402622` 处值进行比较，相同才能开启隐藏关。思考后我们断定，`0x603870` 处的值是前面过程中存放的，只要在最后一个 `bomb_defused` 函数前设置断点，再进行查看就行。查看后发现 `0x603870` 处值为 `"7 0"` ，正是 `phase_4` 的输入，那么要进入隐藏关只需要在输入后继续添加 `"DrEvil"` 就行了。



下面分析 `secret_phase` 部分

```
0000000000401204 <fun7>:

  401204:   48 83 ec 08             sub    $0x8,%rsp
  401208:   48 85 ff                test   %rdi,%rdi
  40120b:   74 2b                   je     401238 <fun7+0x34>
  40120d:   8b 17                   mov    (%rdi),%edx
  40120f:   39 f2                   cmp    %esi,%edx
  401211:   7e 0d                   jle    401220 <fun7+0x1c>
  401213:   48 8b 7f 08             mov    0x8(%rdi),%rdi
  401217:   e8 e8 ff ff ff          callq  401204 <fun7>
  40121c:   01 c0                   add    %eax,%eax
  40121e:   eb 1d                   jmp    40123d <fun7+0x39>
  401220:   b8 00 00 00 00          mov    $0x0,%eax
  401225:   39 f2                   cmp    %esi,%edx
  401227:   74 14                   je     40123d <fun7+0x39>
```

```
  401229:   48 8b 7f 10             mov    0x10(%rdi),%rdi
  40122d:   e8 d2 ff ff ff          callq  401204 <fun7>
  401232:   8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
  401236:   eb 05                   jmp    40123d <fun7+0x39>
  401238:   b8 ff ff ff ff          mov    $0xffffffff,%eax
  40123d:   48 83 c4 08             add    $0x8,%rsp
  401241:   c3                      retq

0000000000401242 <secret_phase>:

  401242:   53                      push   %rbx
  401243:   e8 56 02 00 00          callq  40149e <read_line>
  401248:   ba 0a 00 00 00          mov    $0xa,%edx
  40124d:   be 00 00 00 00          mov    $0x0,%esi
  401252:   48 89 c7                mov    %rax,%rdi
  401255:   e8 76 f9 ff ff          callq  400bd0 <strtol@plt>
  40125a:   48 89 c3                mov    %rax,%rbx
  40125d:   8d 40 ff                lea    -0x1(%rax),%eax
  401260:   3d e8 03 00 00          cmp    $0x3e8,%eax
  401265:   76 05                   jbe    40126c <secret_phase+0x2a>
  401267:   e8 ce 01 00 00          callq  40143a <explode_bomb>

  40126c:   89 de                   mov    %ebx,%esi
  40126e:   bf f0 30 60 00          mov    $0x6030f0,%edi
  401273:   e8 8c ff ff ff          callq  401204 <fun7>
  401278:   83 f8 02                cmp    $0x2,%eax
  40127b:   74 05                   je     401282 <secret_phase+0x40>
  40127d:   e8 b8 01 00 00          callq  40143a <explode_bomb>
  401282:   bf 38 24 40 00          mov    $0x402438,%edi
  401287:   e8 84 f8 ff ff          callq  400b10 <puts@plt>
  40128c:   e8 33 03 00 00          callq  4015c4 <phase_defused>
  401291:   5b                      pop    %rbx
  401292:   c3                      retq
```

分析代码可知，程序读入一个数字 `a` ，满足 `1 <= a <= 0x3e8` 。接着以 `a` 和地址 `0x6030f0` 为参数，调用 `fuc7` 函数，函数返回值为 `2` 时拆弹成功。根据对 `fun7` 函数的分析，可以得到如下近似代码。

```
int fun7(int *next, int a)
{
    if (next == NULL) return 0xffffffff;

    int tem = *next;
    if (tem > a)
    {
        next = *(next + 8);
        return fun7(next, a) << 1;
    }
    else {
      if (tem == a) return 0;
```

```
        next = *(next + 10);
            return (fun7(next, a) << 1) | 1;
        }

}
```

根据 `f(x) = 2` 可推得 `f(next(x)) = 1` `f(next(next(x))) = 0`，要求递归第三层时 `a` 与 `tem` 值相等，可推得应输入 `22`。

实验心得

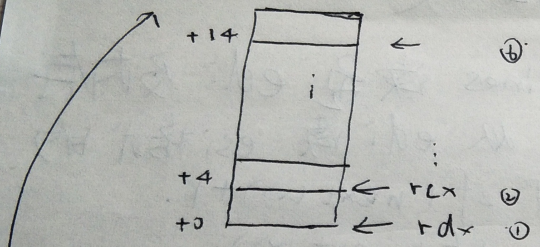
通过这个实验将理论学到的知识真正的运用到了实践当中，琢磨了好久终于拆除一个炸弹的感觉不要太开心。除此之外通过实验也让我接触到了 `GDB` `objdump` 等工具，让我对反编译工作有了更深入的了解。

其它

本实验所有资源在 https://github.com/Afool1999/CSAPP-Labs/tree/master/bomblab

phase_1 x/s 0x402400

phase_2 read_six_numbers.



a[i] = a[i-1] <<1;

x/s 0x4025c3 "%d %d %d %d %d %d"

phase_3

x/s 0x4025cf "%d %d".

(rsp+8) ← a (rsp+c) ← b.

a <= 7..

一开始认为 a=-669 b=-669.

直接跳转到 400f88 直接成功.

但是比较是 ja 无符号.

考虑查看内存中的值.

x/wx 0x24 0x470 ....

四字一起.

发现 a=1, 0x402478

值为

0x00400f69.

再 0x1 与 b 比较,

相等就过关.

phase_4.

a∈[0,14].

func4(a, 0, 14)

可以发现 func4 其实是类似
分的程序. 最后返回 mid
我们提供的 a 要使返
为0. 而最简单的 a 就
同时要求 b 为 0.
只要 a 一直小于 mid
或 a 恰等于 mid
a 不可以大于 mid