北京科技大学实验报告

学院：计算机与通信工程学　专业：信息安全　　班级：信安 201

院

姓名：曾绎哲　　学号：42024058　　日期：2021.10.28

实验名称：Bomblab 二进制炸弹实验

实验目的：

通过拆解给定的二进制炸弹程序，熟悉 Linux 系统的使用，掌握程序反汇编和逆向工程的基本方法，理解汇编语言，学习使用调试器的方法。

实验环境：

- 操作系统：ubuntu 18.04
- GDB 版本：8.1.1

实验前建议与说明：

推荐先大致阅读王爽的《汇编语言》，对拆弹实验中的语法能有大致的了解，对其原理也能有大致的掌握。其次参考《深入理解计算机系统》，拆弹实验的原理便可以较为清晰。

本篇实验报告侧重于讲解如何拆除炸弹，因为对于汇编语言的底层原理并未完全掌握，因此在拆除炸弹的过程中技巧与知识缺一不可，且技巧居多。

实验内容与步骤：

首先，使用 tar -xvf 42024058.tar 指令解压压缩包。

然后使用 objdump -d bomb > bomb.s 对 bomb 文件进行反汇编，并将输出的汇编指令导入 bomb.s 文件当中。通过指令 gdb bomb 进入 gdb 调试阶段，开始拆弹！

Phase_0：字符串的比较

```
0804959b <phase_0>:
 804959b:   f3 0f 1e fb           endbr32
 804959f:   55                    push   %ebp
 80495a0:   89 e5                 mov    %esp,%ebp
 80495a2:   83 ec 08              sub    $0x8,%esp
 80495a5:   83 ec 08              sub    $0x8,%esp   //开辟空间
 80495a8:   68 cc b1 04 08        push   $0x804b1cc  //输入字符串
 80495ad:   ff 75 08              pushl  0x8(%ebp)  //压栈，入原有字符串
 80495b0:   e8 bf 08 00 00        call   8049e74 <strings_not_equal> //调用函数，进行字符串的比较
 80495b5:   83 c4 10              add    $0x10,%esp
 80495b8:   85 c0                 test   %eax,%eax  //相等则标志位ZF=0
 80495ba:   74 0c                 je     80495c8 <phase_0+0x2d> //跳转
 80495bc:   e8 33 0b 00 00        call   804a0f4 <explode_bomb>//不相等则引爆
 80495c1:   b8 00 00 00 00        mov    $0x0,%eax
 80495c6:   eb 05                 jmp    80495cd <phase_0+0x32>
 80495c8:   b8 01 00 00 00        mov    $0x1,%eax
 80495cd:   c9                    leave
 80495ce:   c3                    ret
```

这是一个代码量极小的炸弹，快速浏览完毕后便基本可以知道这是要进行字符串的比较（由函数名称<string_not_equal>亦可知）。因此我们只需要输入一个与地址 0x804b1cc 中存储的字符串相等的字符串即可使得代码跳转从而避免炸弹爆炸。所以我们进入 gdb bomb 调试模式，使用指令 x/s 0x804b1cc 查看该地址的内容：

```
(gdb) x/s 0x804b1cc
0x804b1cc:      "Power is a challenge for integrated circuits."
(gdb)
```

随后只需要在 gdb bomb 中执行指令 run，输入上图中的字符串内容即可，拆弹成功！

```
(gdb) run
Starting program: /home/jovyan/work/bomb
Welcome to my fiendish little bomb. You have 7 phases with
which to blow yourself up. Have a nice day!
Power is a challenge for integrated circuits.
Well done! You seem to have warmed up!
```

Phase_1:浮点数的表示

```
080495cf <phase_1>:
 80495cf:   f3 0f 1e fb             endbr32
 80495d3:   55                      push   %ebp
 80495d4:   89 e5                   mov    %esp,%ebp
 80495d6:   83 ec 38                sub    $0x38,%esp  //开辟空间
 80495d9:   8b 45 08                mov    0x8(%ebp),%eax
 80495dc:   89 45 d4                mov    %eax,-0x2c(%ebp)
 80495df:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
 80495e5:   89 45 f4                mov    %eax,-0xc(%ebp)
 80495e8:   31 c0                   xor    %eax,%eax
 80495ea:   c7 45 e4 ca db a6 2c    movl   $0x2ca6dbca,-0x1c(%ebp)
 80495f1:   db 45 e4                fildl  -0x1c(%ebp)
 80495f4:   dd 5d e8                fstpl  -0x18(%ebp) //浮点数转换
 80495f7:   8d 45 e0                lea    -0x20(%ebp),%eax
 80495fa:   50                      push   %eax
 80495fb:   8d 45 dc                lea    -0x24(%ebp),%eax
 80495fe:   50                      push   %eax
 80495ff:   68 fa b1 04 08          push   $0x804b1fa  //"%d %d", 将输入的两个数入栈
 8049604:   ff 75 d4                pushl  -0x2c(%ebp)
 8049607:   e8 e4 fb ff ff          call   80491f0 <__isoc99_sscanf@plt>  //输入两个数
 804960c:   83 c4 10                add    $0x10,%esp
 804960f:   83 f8 02                cmp    $0x2,%eax //判断输入数字的个数是否为2
 8049612:   74 0c                   je     8049620 <phase_1+0x51>
 8049614:   e8 db 0a 00 00          call   804a0f4 <explode_bomb> //输入数字个数不为2则爆炸
 8049619:   b8 00 00 00 00          mov    $0x0,%eax
 804961e:   eb 2c                   jmp    804964c <phase_1+0x7d>
 8049620:   8d 45 e8                lea    -0x18(%ebp),%eax
 8049623:   8b 10                   mov    (%eax),%edx
 8049625:   8b 45 dc                mov    -0x24(%ebp),%eax
 8049628:   39 c2                   cmp    %eax,%edx
 804962a:   75 0f                   jne    804963b <phase_1+0x6c>
 804962c:   8d 45 e8                lea    -0x18(%ebp),%eax//将答案的地址传给eax
 804962f:   83 c0 04                add    $0x4,%eax //eax地址+4
 8049632:   8b 10                   mov    (%eax),%edx //将eax+4传给edx
 8049634:   8b 45 e0                mov    -0x20(%ebp),%eax//将输入的个数的地址传给eax
 8049637:   39 c2                   cmp    %eax,%edx//比较内容
 8049639:   74 0c                   je     8049647 <phase_1+0x78>
 804963b:   e8 b4 0a 00 00          call   804a0f4 <explode_bomb> //将eax与edx中的值进行比较，不相等则爆炸。因此需要设置断点
 8049640:   b8 00 00 00 00          mov    $0x0,%eax
 8049645:   eb 05                   jmp    804964c <phase_1+0x7d>
 8049647:   b8 01 00 00 00          mov    $0x1,%eax
 804964c:   8b 4d f4                mov    -0xc(%ebp),%ecx
 804964f:   65 33 0d 14 00 00 00    xor    %gs:0x14,%ecx
 8049656:   74 05                   je     804965d <phase_1+0x8e>
 8049658:   e8 33 fb ff ff          call   8049190 <__stack_chk_fail@plt>
 804965d:   c9                      leave
 804965e:   c3                      ret
```

代码量稍大，但正常浏览阅读后基本上清楚代码的目的是什么，边分析边写好注释。我们发现了 push 0x804b1fa，这是一个敏感点，于是我们使用 gdb 调试查看一下：

```
(gdb) x/s 0x804b1fa
0x804b1fa:      "%d %d"
```

结合前两的两个 push 入栈以及后面的压栈和调用函数，以及函数名<sscanf@plt>基本可以判断这是一个输入的含义，需要我们输入两个数。

```
804960f:   83 f8 02                cmp    $0x2,%eax //判断输入数字的个数是否为2
8049612:   74 0c                   je     8049620 <phase_1+0x51>
8049614:   e8 db 0a 00 00          call   804a0f4 <explode_bomb> //输入数字个数不为2则爆炸
```

这是一个判断语句，如果输入的数字个数不为 2，那么炸弹就爆炸，所以确定，我们需要输入两个数。

```
80495f1:    db 45 e4                fildl  -0x1c(%ebp)
80495f4:    dd 5d e8                fstpl  -0x18(%ebp) //浮点数转换
```

这是将数字转为浮点数，查阅资料明白其含义即可。

```
804962c:   8d 45 e8             lea    -0x18(%ebp),%eax//将答案的地址传给eax
804962f:   83 c0 04             add    $0x4,%eax //eax地址+4
8049632:   8b 10                mov    (%eax),%edx //将eax+4传给edx
8049634:   8b 45 e0             mov    -0x20(%ebp),%eax//将输入的个数的地址传给eax
8049637:   39 c2                cmp    %eax,%edx//比较内容
8049639:   74 0c                je     8049647 <phase_1+0x78>
804963b:   e8 b4 0a 00 00       call   804a0f4 <explode_bomb> //将eax与edx中的值进行比较，不相等则爆炸。因此需要设置断点
```

重点在这里，阅读代码我们发现，其实它就是将我们输入的数字与其原本存有的数字浮点数形式进行比较，如果都相等则跳转，否则炸弹爆炸。所以我们只需要知道原本存有的浮点数即可，而他就存在 edx 中，所以我们在炸弹爆炸处设置断点，用 gdb 模式中的 i r edx 指令查看即可：

1）设置断点

```
(gdb) b *0x804963b
Breakpoint 1 at 0x804963b
```

2）查看第一个答案

```
10 20

Breakpoint 1, 0x0804963b in phase_1 ()
(gdb) i r edx
edx            0xe5000000          -452984832
```

3）查看第二个答案

```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/jovyan/work/bomb
Welcome to my fiendish little bomb. You have 7 phases with
which to blow yourself up. Have a nice day!
1

BOOM!!!
The bomb has blown up.
-452984832 10

Breakpoint 1, 0x0804963b in phase_1 ()
(gdb) i r edx
edx            0x41c6536d          1103516525
(gdb)
```

所以输入的两个数字为：-452984832 1103516525

```
-452984832 1103516525
Phase 1 defused. How about the next one?
```

拆弹成功！

Phase_2：循环

```
804965f:   f3 0f 1e fb           endbr32
8049663:   55                    push   %ebp
8049664:   89 e5                 mov    %esp,%ebp
8049666:   83 ec 48              sub    $0x48,%esp
8049669:   8b 45 08              mov    0x8(%ebp),%eax
804966c:   89 45 c4              mov    %eax,-0x3c(%ebp)
804966f:   65 a1 14 00 00 00     mov    %gs:0x14,%eax
8049675:   89 45 f4              mov    %eax,-0xc(%ebp)
8049678:   31 c0                 xor    %eax,%eax
804967a:   83 ec 04              sub    $0x4,%esp //0x44esp
804967d:   6a 07                 push   $0x7 //input 7 numbers
804967f:   8d 45 d8              lea    -0x28(%ebp),%eax
8049682:   50                    push   %eax
8049683:   ff 75 c4              pushl  -0x3c(%ebp)
8049686:   e8 27 07 00 00        call   8049db2 <read_n_numbers>
804968b:   83 c4 10              add    $0x10,%esp
804968e:   85 c0                 test   %eax,%eax
8049690:   75 07                 jne    8049699 <phase_2+0x3a>
8049692:   b8 00 00 00 00        mov    $0x0,%eax
8049697:   eb 54                 jmp    80496ed <phase_2+0x8e>
8049699:   8b 45 d8              mov    -0x28(%ebp),%eax
804969c:   3d ca 00 00 00        cmp    $0xca,%eax//array[7]==202
80496a1:   74 0c                 je     80496af <phase_2+0x50>
80496a3:   e8 4c 0a 00 00        call   804a0f4 <explode_bomb>
80496a8:   b8 00 00 00 00        mov    $0x0,%eax
80496ad:   eb 3e                 jmp    80496ed <phase_2+0x8e>
80496af:   c7 45 d4 01 00 00 00  movl   $0x1,-0x2c(%ebp)//i=1
80496b6:   eb 2a                 jmp    80496e2 <phase_2+0x83>
80496b8:   8b 45 d4              mov    -0x2c(%ebp),%eax
80496bb:   8b 44 85 d8           mov    -0x28(%ebp,%eax,4),%eax //eax=array[i]
80496bf:   8b 55 d4              mov    -0x2c(%ebp),%edx
80496c2:   83 ea 01              sub    $0x1,%edx //edx=i-1
80496c5:   8b 54 95 d8           mov    -0x28(%ebp,%edx,4),%edx//edx=array[i-1]
80496c9:   d1 fa                 sar    %edx//shift arithmetic right == edx/2
80496cb:   83 c2 01              add    $0x1,%edx//edx=edx+1
80496ce:   39 d0                 cmp    %edx,%eax //array[i]=sar(array[i-1])+1
80496d0:   74 0c                 je     80496de <phase_2+0x7f>
80496d2:   e8 1d 0a 00 00        call   804a0f4 <explode_bomb>
80496d7:   b8 00 00 00 00        mov    $0x0,%eax
80496dc:   eb 0f                 jmp    80496ed <phase_2+0x8e>
80496de:   83 45 d4 01           addl   $0x1,-0x2c(%ebp)//i++
80496e2:   83 7d d4 06           cmpl   $0x6,-0x2c(%ebp)//6<=i; cycle 6 times
80496e6:   7e d0                 jle    80496b8 <phase_2+0x59>
80496e8:   b8 01 00 00 00        mov    $0x1,%eax
80496ed:   8b 4d f4              mov    -0xc(%ebp),%ecx
80496f0:   65 33 0d 14 00 00 00  xor    %gs:0x14,%ecx
80496f7:   74 05                 je     80496fe <phase_2+0x9f>
80496f9:   e8 92 fa ff ff        call   8049190 <__stack_chk_fail@plt>
80496fe:   c9                    leave
```

代码量稍大，但总体也不长，正常阅读浏览完成后，基本了解这是一个循环。于是我们从头开始分析代码。

```
8049663:   55                      push   %ebp
8049664:   89 e5                   mov    %esp,%ebp
8049666:   83 ec 48                sub    $0x48,%esp
8049669:   8b 45 08                mov    0x8(%ebp),%eax
804966c:   89 45 c4                mov    %eax,-0x3c(%ebp)
804966f:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
8049675:   89 45 f4                mov    %eax,-0xc(%ebp)
8049678:   31 c0                   xor    %eax,%eax
804967a:   83 ec 04                sub    $0x4,%esp //0x44esp
804967d:   6a 07                   push   $0x7 //input 7 numbers
804967f:   8d 45 d8                lea    -0x28(%ebp),%eax
```

首先是开辟空间，由指令 push $0x7 基本确定我们需要输入 7 个数字。

```
8049699:   8b 45 d8                mov    -0x28(%ebp),%eax
804969c:   3d ca 00 00 00          cmp    $0xca,%eax//array[7]==202
80496a1:   74 0c                   je     80496af <phase_2+0x50>
80496a3:   e8 4c 0a 00 00          call   804a0f4 <explode_bomb>
```

因为要输入七个数字，所以我们猜测是一个数组，由上图代码可知，输入的其中一个数要与 0xca 相等，而这个数的地址是栈尾，所以大胆猜测这个数组的最后一个数为 0xca，即十进制的 202，数组名暂且为 array，否则就爆炸。

```
80496a8:   b8 00 00 00 00          mov    $0x0,%eax
80496ad:   eb 3e                   jmp    80496ed <phase_2+0x8e>
80496af:   c7 45 d4 01 00 00 00    movl   $0x1,-0x2c(%ebp)//i=1
80496b6:   eb 2a                   jmp    80496e2 <phase_2+0x83>
80496b8:   8b 45 d4                mov    -0x2c(%ebp),%eax
80496bb:   8b 44 85 d8             mov    -0x28(%ebp,%eax,4),%eax //eax=array[i]
80496bf:   8b 55 d4                mov    -0x2c(%ebp),%edx
80496c2:   83 ea 01                sub    $0x1,%edx //edx=i-1
80496c5:   8b 54 95 d8             mov    -0x28(%ebp,%edx,4),%edx//edx=array[i-1]
80496c9:   d1 fa                   sar    %edx//shift arithmetic right == edx/2
80496cb:   83 c2 01                add    $0x1,%edx//edx=edx+1
80496ce:   39 d0                   cmp    %edx,%eax //array[i]=sar(array[i-1])+1
80496d0:   74 0c                   je     80496de <phase_2+0x7f>
80496d2:   e8 1d 0a 00 00          call   804a0f4 <explode_bomb>
```

这一部分是重点，这其实就是一个较为简单的算法，类似于 c++中的 for 循环。我们来看一下这个算法：首先，-8x2c（%ebp）存放的是 for 循环中 i 的值，即一个计数器，i 初始值为 1 。然后将数组 array[1]的地址给 edx，然后对 edx 进行逻辑右移操作，查阅资料得知，其实就是 c++中的整型除法，edx=edx/2（注意是 c++中的整型除法!!!),然后 edx 再加 1 ,把这个值传递给 eax,与 array[i-1]比较，如果相等则跳转，否则爆炸。这就意味着，我们输入的数字存放在数组里，必须满足以下规律：array[i]=array[i-1]/2+1。

```
80496de:   83 45 d4 01            addl   $0x1,-0x2c(%ebp)//i++
80496e2:   83 7d d4 06            cmpl   $0x6,-0x2c(%ebp)//6<=i; cycle 6 times
```

由代码知，循环一共 6 次。

而 array[7]=202，总共有七个数字，所以我们可以得到答案：202 102 52 27 14 8 5

```
202 102 52 27 14 8 5
That's number 2.  Keep going!
```

拆弹成功！

Phase_3:条件/分支

```
08049700 <phase_3>:
 8049700:   f3 0f 1e fb           endbr32
 8049704:   55                    push   %ebp
 8049705:   89 e5                 mov    %esp,%ebp
 8049707:   83 ec 38              sub    $0x38,%esp
 804970a:   8b 45 08              mov    0x8(%ebp),%eax
 804970d:   89 45 d4              mov    %eax,-0x2c(%ebp)
 8049710:   65 a1 14 00 00 00     mov    %gs:0x14,%eax
 8049716:   89 45 f4              mov    %eax,-0xc(%ebp)
 8049719:   31 c0                 xor    %eax,%eax
 804971b:   c7 45 f0 00 00 00 00  movl   $0x0,-0x10(%ebp)
 8049722:   83 ec 0c              sub    $0xc,%esp  //开辟空间
 8049725:   8d 45 ec              lea    -0x14(%ebp),%eax
 8049728:   50                    push   %eax
 8049729:   8d 45 e6              lea    -0x1a(%ebp),%eax
 804972c:   50                    push   %eax
 804972d:   8d 45 e8              lea    -0x18(%ebp),%eax
 8049730:   50                    push   %eax  //入栈
 8049731:   68 00 b2 04 08        push   $0x804b200 //"%d %c %d"
 8049736:   ff 75 d4              pushl  -0x2c(%ebp)
 8049739:   e8 b2 fa ff ff        call   80491f0 <__isoc99_sscanf@plt>//输入两个数字一个字符
 804973e:   83 c4 20              add    $0x20,%esp
 8049741:   89 45 f0              mov    %eax,-0x10(%ebp)
 8049744:   83 7d f0 02           cmpl   $0x2,-0x10(%ebp)//判断输入的数字的个数是否为2
 8049748:   7f 0f                 jg     8049759 <phase_3+0x59>
 804974a:   e8 a5 09 00 00        call   804a0f4 <explode_bomb>//输入数字个数不为2则爆炸
 804974f:   b8 00 00 00 00        mov    $0x0,%eax
 8049754:   e9 4a 01 00 00        jmp    80498a3 <phase_3+0x1a3>
 8049759:   8b 45 e8              mov    -0x18(%ebp),%eax
 804975c:   2d 88 00 00 00        sub    $0x88,%eax
 8049761:   83 f8 07              cmp    $0x7,%eax
 8049764:   0f 87 f9 00 00 00     ja     8049863 <phase_3+0x163>//输入的第一个数减去0x88后要小于7
 804976a:   8b 04 85 0c b2 04 08  mov    0x804b20c(,%eax,4),%eax
 8049771:   3e ff e0              notrack jmp *%eax
 8049774:   c6 45 e7 73           movb   $0x73,-0x19(%ebp)
 8049778:   8b 45 ec              mov    -0x14(%ebp),%eax
 804977b:   3d 26 01 00 00        cmp    $0x126,%eax//输入的第三个数必须为0x126==294
 8049780:   0f 84 ed 00 00 00     je     8049873 <phase_3+0x173>
 8049786:   e8 69 09 00 00        call   804a0f4 <explode_bomb>
 804978b:   b8 00 00 00 00        mov    $0x0,%eax
 8049790:   e9 0e 01 00 00        jmp    80498a3 <phase_3+0x1a3>
 8049795:   c6 45 e7 73           movb   $0x73,-0x19(%ebp)//将输入的字符存入-0x19(%ebp)，字符ascii为0x73，所以字符为 s
```

```
8049799:   8b 45 ec              mov    -0x14(%ebp),%eax
804979c:   3d 26 01 00 00        cmp    $0x126,%eax
80497a1:   0f 84 cf 00 00 00     je     8049876 <phase_3+0x176>
80497a7:   e8 48 09 00 00        call   804a0f4 <explode_bomb>
80497ac:   b8 00 00 00 00        mov    $0x0,%eax
80497b1:   e9 ed 00 00 00        jmp    80498a3 <phase_3+0x1a3>
80497b6:   c6 45 e7 73           movb   $0x73,-0x19(%ebp)
80497ba:   8b 45 ec              mov    -0x14(%ebp),%eax
80497bd:   3d 26 01 00 00        cmp    $0x126,%eax
80497c2:   0f 84 b1 00 00 00     je     8049879 <phase_3+0x179>
80497c8:   e8 27 09 00 00        call   804a0f4 <explode_bomb>
80497cd:   b8 00 00 00 00        mov    $0x0,%eax
80497d2:   e9 cc 00 00 00        jmp    80498a3 <phase_3+0x1a3>
80497d7:   c6 45 e7 73           movb   $0x73,-0x19(%ebp)
80497db:   8b 45 ec              mov    -0x14(%ebp),%eax
80497de:   3d 26 01 00 00        cmp    $0x126,%eax
80497e3:   0f 84 93 00 00 00     je     804987c <phase_3+0x17c>
80497e9:   e8 06 09 00 00        call   804a0f4 <explode_bomb>
80497ee:   b8 00 00 00 00        mov    $0x0,%eax
80497f3:   e9 ab 00 00 00        jmp    80498a3 <phase_3+0x1a3>
80497f8:   c6 45 e7 73           movb   $0x73,-0x19(%ebp)
80497fc:   8b 45 ec              mov    -0x14(%ebp),%eax
80497ff:   3d 26 01 00 00        cmp    $0x126,%eax
8049804:   74 79                 je     804987f <phase_3+0x17f>
8049806:   e8 e9 08 00 00        call   804a0f4 <explode_bomb>
804980b:   b8 00 00 00 00        mov    $0x0,%eax
8049810:   e9 8e 00 00 00        jmp    80498a3 <phase_3+0x1a3>
8049815:   c6 45 e7 73           movb   $0x73,-0x19(%ebp)
8049819:   8b 45 ec              mov    -0x14(%ebp),%eax
804981c:   3d 26 01 00 00        cmp    $0x126,%eax
8049821:   74 5f                 je     8049882 <phase_3+0x182>
8049823:   e8 cc 08 00 00        call   804a0f4 <explode_bomb>
8049828:   b8 00 00 00 00        mov    $0x0,%eax
804982d:   eb 74                 jmp    80498a3 <phase_3+0x1a3>
804982f:   c6 45 e7 73           movb   $0x73,-0x19(%ebp)
8049833:   8b 45 ec              mov    -0x14(%ebp),%eax
8049836:   3d 26 01 00 00        cmp    $0x126,%eax
804983b:   74 48                 je     8049885 <phase_3+0x185>
804983d:   e8 b2 08 00 00        call   804a0f4 <explode_bomb>
8049842:   b8 00 00 00 00        mov    $0x0,%eax
8049847:   eb 5a                 jmp    80498a3 <phase_3+0x1a3>
8049849:   c6 45 e7 61           movb   $0x61,-0x19(%ebp)
804984d:   8b 45 ec              mov    -0x14(%ebp),%eax
8049850:   3d d2 03 00 00        cmp    $0x3d2,%eax
8049855:   74 31                 je     8049888 <phase_3+0x188>
8049857:   e8 98 08 00 00        call   804a0f4 <explode_bomb>
```

```
804985c:   b8 00 00 00 00        mov    $0x0,%eax
8049861:   eb 40                 jmp    80498a3 <phase_3+0x1a3>
8049863:   c6 45 e7 73           movb   $0x73,-0x19(%ebp)
8049867:   e8 88 08 00 00        call   804a0f4 <explode_bomb>
804986c:   b8 00 00 00 00        mov    $0x0,%eax
8049871:   eb 30                 jmp    80498a3 <phase_3+0x1a3>
8049873:   90                    nop
8049874:   eb 13                 jmp    8049889 <phase_3+0x189>
8049876:   90                    nop
8049877:   eb 10                 jmp    8049889 <phase_3+0x189>
8049879:   90                    nop
804987a:   eb 0d                 jmp    8049889 <phase_3+0x189>
804987c:   90                    nop
804987d:   eb 0a                 jmp    8049889 <phase_3+0x189>
804987f:   90                    nop
8049880:   eb 07                 jmp    8049889 <phase_3+0x189>
8049882:   90                    nop
8049883:   eb 04                 jmp    8049889 <phase_3+0x189>
8049885:   90                    nop
8049886:   eb 01                 jmp    8049889 <phase_3+0x189>
8049888:   90                    nop
8049889:   0f b6 45 e6           movzbl -0x1a(%ebp),%eax
804988d:   38 45 e7              cmp    %al,-0x19(%ebp)
8049890:   74 0c                 je     804989e <phase_3+0x19e>
8049892:   e8 5d 08 00 00        call   804a0f4 <explode_bomb>
8049897:   b8 00 00 00 00        mov    $0x0,%eax
804989c:   eb 05                 jmp    80498a3 <phase_3+0x1a3>
804989e:   b8 01 00 00 00        mov    $0x1,%eax
80498a3:   8b 55 f4              mov    -0xc(%ebp),%edx
80498a6:   65 33 15 14 00 00 00  xor    %gs:0x14,%edx
80498ad:   74 05                 je     80498b4 <phase_3+0x1b4>
80498af:   e8 dc f8 ff ff        call   8049190 <__stack_chk_fail@plt>
80498b4:   c9                    leave
80498b5:   c3                    ret
```

总体来看，代码很长，可能这题的任务量不小。但仔细认真研读之后会发现，其中有一半以上都是类似于 c++中 switch 的选择语句以及 if 的条件判断语句。首先我们看到代码的首部分：

```
8049700:   f3 0f 1e fb           endbr32
8049704:   55                    push   %ebp
8049705:   89 e5                 mov    %esp,%ebp
8049707:   83 ec 38              sub    $0x38,%esp
804970a:   8b 45 08              mov    0x8(%ebp),%eax
804970d:   89 45 d4              mov    %eax,-0x2c(%ebp)
8049710:   65 a1 14 00 00 00     mov    %gs:0x14,%eax
8049716:   89 45 f4              mov    %eax,-0xc(%ebp)
8049719:   31 c0                 xor    %eax,%eax
804971b:   c7 45 f0 00 00 00 00  movl   $0x0,-0x10(%ebp)
8049722:   83 ec 0c              sub    $0xc,%esp  //开辟空间
8049725:   8d 45 ec              lea    -0x14(%ebp),%eax
8049728:   50                    push   %eax
8049729:   8d 45 e6              lea    -0x1a(%ebp),%eax
804972c:   50                    push   %eax
804972d:   8d 45 e8              lea    -0x18(%ebp),%eax
8049730:   50                    push   %eax  //入栈
8049731:   68 00 b2 04 08        push   $0x804b200 //"%d %c %d"
8049736:   ff 75 d4              pushl  -0x2c(%ebp)
8049739:   e8 b2 fa ff ff        call   80491f0 <__isoc99_sscanf@plt>//输入两个数字一个字符
804973e:   83 c4 20              add    $0x20,%esp
8049741:   89 45 f0              mov    %eax,-0x10(%ebp)
8049744:   83 7d f0 02           cmpl   $0x2,-0x10(%ebp)//判断输入的数字的个数是否为2
8049748:   7f 0f                 jg     8049759 <phase_3+0x59>
804974a:   e8 a5 09 00 00        call   804a0f4 <explode_bomb>//输入数字个数不为2则爆炸
```

这是基础操作，开辟空间，然后查出 0x804b200 地址的内容，发现需要输入两个数字和一个字符，然后判断输入数字的个数是否为 2，是则跳转，否则爆炸。

```
(gdb) x/s 0x804b200
0x804b200:      "%d %c %d"
```

这是查到的 0x804b200 的内容。

```
804974f:   b8 00 00 00 00        mov    $0x0,%eax
8049754:   e9 4a 01 00 00        jmp    80498a3 <phase_3+0x1a3>
8049759:   8b 45 e8              mov    -0x18(%ebp),%eax
804975c:   2d 88 00 00 00        sub    $0x88,%eax
8049761:   83 f8 07              cmp    $0x7,%eax
8049764:   0f 87 f9 00 00 00     ja     8049863 <phase_3+0x163>//输入的第一个数减去0x88后要小于7
```

然后再往下看，我们发现有一个判断语句，将第一个输入的数字减去 0x88 后要小于 0x7 但是大于等于 0，所以我们知道第一个是输入的数字最大为为 0x88+0x7，即最大为 0x8f。所以我们可以选择第一个数字为 0x88，即 136。否则会跳转，而跳转的结果就是炸弹爆炸，所以我们不可以让他跳转。

```
804976a:   8b 04 85 0c b2 04 08    mov    0x804b20c(,%eax,4),%eax
8049771:   3e ff e0                notrack jmp *%eax
8049774:   c6 45 e7 73             movb   $0x73,-0x19(%ebp)
8049778:   8b 45 ec                mov    -0x14(%ebp),%eax
804977b:   3d 26 01 00 00          cmp    $0x126,%eax//输入的第三个数必须为0x126==294
8049780:   0f 84 ed 00 00 00       je     8049873 <phase_3+0x173>
8049786:   e8 69 09 00 00          call   804a0f4 <explode_bomb>
```

再往下看，我们看到第二个输入的数字的地址给了 eax，而 eax 的值必须与 0x126 相等，否则就爆炸，由此可知第二个输入的数字为 0x126，即 294 。

```
804978b:   b8 00 00 00 00          mov    $0x0,%eax
8049790:   e9 0e 01 00 00          jmp    80498a3 <phase_3+0x1a3>
8049795:   c6 45 e7 73             movb   $0x73,-0x19(%ebp)//将输入的字符存入-0x19(%ebp)，字符ascii为0x73，所以字符为 s
```

```
804988d:   38 45 e7                cmp    %al,-0x19(%ebp)
8049890:   74 0c                   je     804989e <phase_3+0x19e>
8049892:   e8 5d 08 00 00          call   804a0f4 <explode_bomb>
```

再往下看，发现这两段对字符的输入做了限定，输入的字符的 ASCII 码值必须为 0x73，即字符 s，所以输入的字符为 s。

到此，我们便知道了答案为：136 s 294

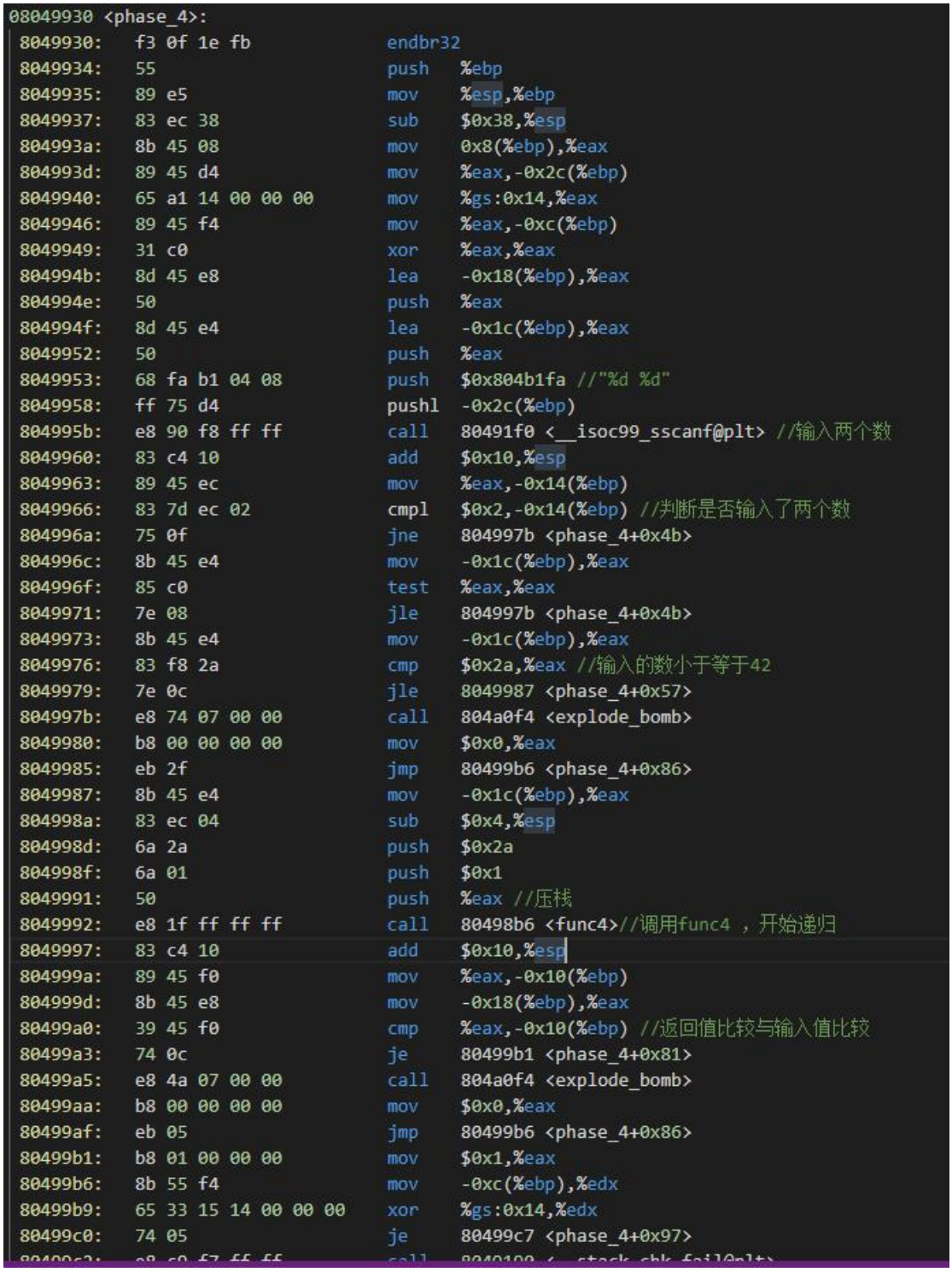
其中第一个数字不一定是 136，只需满足前面的条件即可。

```
136 s 294
Halfway there!
```

拆弹成功！

Phase_4：递归

```
080498b6 <func4>:
 80498b6:   f3 0f 1e fb             endbr32
 80498ba:   55                      push   %ebp
 80498bb:   89 e5                   mov    %esp,%ebp
 80498bd:   83 ec 18                sub    $0x18,%esp
 80498c0:   8b 45 10                mov    0x10(%ebp),%eax
 80498c3:   2b 45 0c                sub    0xc(%ebp),%eax
 80498c6:   89 c2                   mov    %eax,%edx
 80498c8:   c1 ea 1f                shr    $0x1f,%edx
 80498cb:   01 d0                   add    %edx,%eax
 80498cd:   d1 f8                   sar    %eax //逻辑右移31位
 80498cf:   89 c2                   mov    %eax,%edx
 80498d1:   8b 45 0c                mov    0xc(%ebp),%eax
 80498d4:   01 d0                   add    %edx,%eax
 80498d6:   89 45 f4                mov    %eax,-0xc(%ebp)
 80498d9:   8b 45 f4                mov    -0xc(%ebp),%eax
 80498dc:   3b 45 08                cmp    0x8(%ebp),%eax
 80498df:   7e 21                   jle    8049902 <func4+0x4c>
 80498e1:   8b 45 f4                mov    -0xc(%ebp),%eax
 80498e4:   83 e8 01                sub    $0x1,%eax
 80498e7:   83 ec 04                sub    $0x4,%esp
 80498ea:   50                      push   %eax
 80498eb:   ff 75 0c                pushl  0xc(%ebp)
 80498ee:   ff 75 08                pushl  0x8(%ebp)
 80498f1:   e8 c0 ff ff ff          call   80498b6 <func4>
 80498f6:   83 c4 10                add    $0x10,%esp
 80498f9:   8b 55 f4                mov    -0xc(%ebp),%edx
 80498fc:   d1 fa                   sar    %edx
 80498fe:   01 d0                   add    %edx,%eax
 8049900:   eb 2c                   jmp    804992e <func4+0x78>
 8049902:   8b 45 f4                mov    -0xc(%ebp),%eax
 8049905:   3b 45 08                cmp    0x8(%ebp),%eax
 8049908:   7d 21                   jge    804992b <func4+0x75>
 804990a:   8b 45 f4                mov    -0xc(%ebp),%eax
 804990d:   83 c0 01                add    $0x1,%eax
 8049910:   83 ec 04                sub    $0x4,%esp
 8049913:   ff 75 10                pushl  0x10(%ebp)
 8049916:   50                      push   %eax
 8049917:   ff 75 08                pushl  0x8(%ebp)
 804991a:   e8 97 ff ff ff          call   80498b6 <func4>
 804991f:   83 c4 10                add    $0x10,%esp
 8049922:   8b 55 f4                mov    -0xc(%ebp),%edx
 8049925:   01 d2                   add    %edx,%edx
 8049927:   01 d0                   add    %edx,%eax
 8049929:   eb 03                   jmp    804992e <func4+0x78>
 804992b:   8b 45 f4                mov    -0xc(%ebp),%eax
 804992e:   c9                      leave
 804992f:   c3                      ret
```



代码稍长，主要是因为有一个递归函数。先大概浏览一下递归函数，发现逻辑右移 31 位的算法。由于递归函数的代码不太好理解，所以我们直接看 phase4 中的代码。但我知道 func4 中一定是有一个算法的，稍后我们会提到。

```
8049930:   f3 0f 1e fb              endbr32
8049934:   55                       push   %ebp
8049935:   89 e5                    mov    %esp,%ebp
8049937:   83 ec 38                 sub    $0x38,%esp
804993a:   8b 45 08                 mov    0x8(%ebp),%eax
804993d:   89 45 d4                 mov    %eax,-0x2c(%ebp)
8049940:   65 a1 14 00 00 00        mov    %gs:0x14,%eax
8049946:   89 45 f4                 mov    %eax,-0xc(%ebp)
8049949:   31 c0                    xor    %eax,%eax
804994b:   8d 45 e8                 lea    -0x18(%ebp),%eax
804994e:   50                       push   %eax
804994f:   8d 45 e4                 lea    -0x1c(%ebp),%eax
8049952:   50                       push   %eax
8049953:   68 fa b1 04 08           push   $0x804b1fa //"%d %d"
8049958:   ff 75 d4                 pushl  -0x2c(%ebp)
804995b:   e8 90 f8 ff ff           call   80491f0 <__isoc99_sscanf@plt> //输入两个数
8049960:   83 c4 10                 add    $0x10,%esp
8049963:   89 45 ec                 mov    %eax,-0x14(%ebp)
8049966:   83 7d ec 02              cmpl   $0x2,-0x14(%ebp) //判断是否输入了两个数
804996a:   75 0f                    jne    804997b <phase_4+0x4b>
```

前面都是基本操作，开辟空间，输入两个数，判断一下输入的是不是两个数，是则继续，否则就跳转爆炸。（注意，这里的规则跟前面不一样，这里是：输入的是两个数则继续而不跳转，否则就跳转到爆炸函数导致爆炸）

```
804996c:   8b 45 e4                 mov    -0x1c(%ebp),%eax
804996f:   85 c0                    test   %eax,%eax
8049971:   7e 08                    jle    804997b <phase_4+0x4b>
8049973:   8b 45 e4                 mov    -0x1c(%ebp),%eax
8049976:   83 f8 2a                 cmp    $0x2a,%eax //输入的数小于等于42
8049979:   7e 0c                    jle    8049987 <phase_4+0x57>
804997b:   e8 74 07 00 00           call   804a0f4 <explode_bomb>
```

这里有一个很关键的一句话，判断输入的数是否小于等于 0x2a，即 42 。所以我们输入的数应该要小于等于 42 并且大于 0

```
8049980:   b8 00 00 00 00           mov    $0x0,%eax
8049985:   eb 2f                    jmp    80499b6 <phase_4+0x86>
8049987:   8b 45 e4                 mov    -0x1c(%ebp),%eax
804998a:   83 ec 04                 sub    $0x4,%esp
804998d:   6a 2a                    push   $0x2a
804998f:   6a 01                    push   $0x1
8049991:   50                       push   %eax //压栈
8049992:   e8 1f ff ff ff           call   80498b6 <func4>//调用func4 ，开始递归
8049997:   83 c4 10                 add    $0x10,%esp
804999a:   89 45 f0                 mov    %eax,-0x10(%ebp)
804999d:   8b 45 e8                 mov    -0x18(%ebp),%eax
80499a0:   39 45 f0                 cmp    %eax,-0x10(%ebp) //返回值比较与输入值比较
80499a3:   74 0c                    je     80499b1 <phase_4+0x81>
80499a5:   e8 4a 07 00 00           call   804a0f4 <explode_bomb>
```

往下看，这里是重点。压栈并且将输入的第一个数字作为参数传入 func4 中，将最终的返回结果传给 eax 并且与第二个输入的数字进行比较，如果相等就跳转，否则就爆炸。所以我们有一个结论，输入的两个数字要小于等于 42 且第一个输入的数字经过 func4 的处理后与第二个输入的数字相等。接下来的步骤是搞清楚

func4 的内容，明白它的处理机制。根据 func4 中代码我们知道它是一个递归函数，所以我们就可以根据输入的第一个数字计算出第二个数字。
由于第一个数字不能大于 42 且不能小于 0，所以我们随便取一个数字 20，根据 func4 知第二个输入的数字应该是 150，所以，答案为：20 150

```
20 154
So you got that one.  Try this one.
```

拆弹成功！

Phase_5:数组

```
080499c9 <phase_5>:
 80499c9:   f3 0f 1e fb             endbr32
 80499cd:   55                      push   %ebp
 80499ce:   89 e5                   mov    %esp,%ebp
 80499d0:   83 ec 18                sub    $0x18,%esp
 80499d3:   83 ec 0c                sub    $0xc,%esp //开辟空间
 80499d6:   ff 75 08                pushl  0x8(%ebp)
 80499d9:   e8 66 04 00 00          call   8049e44 <string_length> //读取字符串长度
 80499de:   83 c4 10                add    $0x10,%esp
 80499e1:   89 45 f4                mov    %eax,-0xc(%ebp)
 80499e4:   83 7d f4 05             cmpl   $0x5,-0xc(%ebp) //字符串长度应该是5
 80499e8:   74 0c                   je     80499f6 <phase_5+0x2d>
 80499ea:   e8 05 07 00 00          call   804a0f4 <explode_bomb>//判断字符串长度是否为5，是则跳转，否则爆炸
 80499ef:   b8 00 00 00 00          mov    $0x0,%eax
 80499f4:   eb 4c                   jmp    8049a42 <phase_5+0x79>
 80499f6:   c7 45 f0 00 00 00 00    movl   $0x0,-0x10(%ebp)
 80499fd:   c7 45 ec 00 00 00 00    movl   $0x0,-0x14(%ebp)
 8049a04:   eb 1f                   jmp    8049a25 <phase_5+0x5c>
 8049a06:   8b 55 ec                mov    -0x14(%ebp),%edx//开始一个for循环
 8049a09:   8b 45 08                mov    0x8(%ebp),%eax
 8049a0c:   01 d0                   add    %edx,%eax
 8049a0e:   0f b6 00                movzbl (%eax),%eax
 8049a11:   0f be c0                movsbl %al,%eax //每一个字符都对应一个数字
 8049a14:   83 e0 0f                and    $0xf,%eax
 8049a17:   8b 04 85 e0 d1 04 08    mov    0x804d1e0(,%eax,4),%eax
 8049a1e:   01 45 f0                add    %eax,-0x10(%ebp)
 8049a21:   83 45 ec 01             addl   $0x1,-0x14(%ebp)//计算数字之和
 8049a25:   83 7d ec 04             cmpl   $0x4,-0x14(%ebp)
 8049a29:   7e db                   jle    8049a06 <phase_5+0x3d>//for循环结束
 8049a2b:   83 7d f0 26             cmpl   $0x26,-0x10(%ebp)// for循环结束后数字之和应该为0x26，即38
 8049a2f:   74 0c                   je     8049a3d <phase_5+0x74>
 8049a31:   e8 be 06 00 00          call   804a0f4 <explode_bomb>//设置一个断点
 8049a36:   b8 00 00 00 00          mov    $0x0,%eax
 8049a3b:   eb 05                   jmp    8049a42 <phase_5+0x79>
 8049a3d:   b8 01 00 00 00          mov    $0x1,%eax
 8049a42:   c9                      leave
 8049a43:   c3                      ret
```

代码不长，大概浏览得知需要输入一个长度为 5 的字符串进行操作。因此我们从头开始看，逐步分析。

```
80499c9:   f3 0f 1e fb           endbr32
80499cd:   55                    push   %ebp
80499ce:   89 e5                 mov    %esp,%ebp
80499d0:   83 ec 18              sub    $0x18,%esp
80499d3:   83 ec 0c              sub    $0xc,%esp //开辟空间
80499d6:   ff 75 08              pushl  0x8(%ebp)
80499d9:   e8 66 04 00 00        call   8049e44 <string_length> //读取字符串长度
80499de:   83 c4 10              add    $0x10,%esp
80499e1:   89 45 f4              mov    %eax,-0xc(%ebp)
80499e4:   83 7d f4 05           cmpl   $0x5,-0xc(%ebp) //字符串长度应该是5
80499e8:   74 0c                 je     80499f6 <phase_5+0x2d>
80499ea:   e8 05 07 00 00        call   804a0f4 <explode_bomb>//判断字符串长度是否为5，是则跳转，否则爆炸
```

简单的操作，开辟空间，输入一个字符串并返回它的长度，长度若为 5 则继续，否则爆炸，因此我们需要输入一个长度为 5 的字符串。

```
8049a06:   8b 55 ec              mov    -0x14(%ebp),%edx//开始一个for循环
8049a09:   8b 45 08              mov    0x8(%ebp),%eax
8049a0c:   01 d0                 add    %edx,%eax
8049a0e:   0f b6 00              movzbl (%eax),%eax
8049a11:   0f be c0              movsbl %al,%eax //每一个字符都对应一个数字
8049a14:   83 e0 0f              and    $0xf,%eax
8049a17:   8b 04 85 e0 d1 04 08  mov    0x804d1e0(,%eax,4),%eax
8049a1e:   01 45 f0              add    %eax,-0x10(%ebp)
8049a21:   83 45 ec 01           addl   $0x1,-0x14(%ebp)//计算数字之和
8049a25:   83 7d ec 04           cmpl   $0x4,-0x14(%ebp)
8049a29:   7e db                 jle    8049a06 <phase_5+0x3d>//for循环结束
```

开始了一个 for 循环，循环次数为字符串的长度。这其中说明了本题的机制。在这个循环中我们可以看到，每一个字符其实对应了一个数字，类似于 ASCII 码。把每一个字符的值相加，直至循环结束，相当于遍历，然后把这个值存到-0x10(%ebp)中。

```
8049a2b:   83 7d f0 26           cmpl   $0x26,-0x10(%ebp)// for循环结束后数字之和应该为0x26，即38
8049a2f:   74 0c                 je     8049a3d <phase_5+0x74>
8049a31:   e8 be 06 00 00        call   804a0f4 <explode_bomb>//设置一个断点
```

在循环结束之后，我们发现他把那个相加的总和与 0x26，即 38 进行比较，相等则跳转，否则爆炸，所以，五个字符的数值相加应该为 38，但我们如何知道每一个字符对应的数字呢？有一个很简单的办法，就是逐个尝试，看似复杂，其实最为简便，因为我们可以对字符进行排列组合计算。我们首先在爆炸前设置一个断点。

```
(gdb) b *0x8049a31
Breakpoint 1 at 0x8049a31
```

然后开始调试，我们先输入 5 个 a：

```
aaaaa

Breakpoint 1, 0x08049a31 in phase_5 ()
```

用指令 i r ebp 查看 ebp 的地址：

```
(gdb) i r ebp
ebp            0xffffd438          0xffffd438
```

得到了 ebp 的地址，由于那个总和存放在-0x10(%ebp)中，所以我们需要查看-0x10(%ebp)中的内容：

```
(gdb) x/d 0xffffd428
0xffffd428:     40
```

得到 5 个 a 的总和为 40，即字符 a 对应数字 8。用同样的方法，以此类推，我们很快就可以知道字符 b 对应数字 13，字符 s 对应数字 4，所以可以有 13*2+4*3=38 。

```
The bomb has blown up.
bbbbb

Breakpoint 1, 0x08049a31 in phase_5 ()
(gdb) x/d 0xffffd428
0xffffd428:     65
sssss

Breakpoint 1, 0x08049a31 in phase_5 ()
(gdb) x/d 0xffffd428
0xffffd428:     20
```

因此，答案可以为：bbsss（顺序无关紧要）

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

拆弹成功！

Phase_6:链表/二叉树

```
08049a44 <phase_6>:
 8049a44:   f3 0f 1e fb             endbr32
 8049a48:   55                      push   %ebp
 8049a49:   89 e5                   mov    %esp,%ebp
 8049a4b:   83 ec 68                sub    $0x68,%esp
 8049a4e:   8b 45 08                mov    0x8(%ebp),%eax
 8049a51:   89 45 a4                mov    %eax,-0x5c(%ebp)
 8049a54:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
 8049a5a:   89 45 f4                mov    %eax,-0xc(%ebp)
 8049a5d:   31 c0                   xor    %eax,%eax
 8049a5f:   c7 45 c0 08 d1 04 08    movl   $0x804d108,-0x40(%ebp)//这是一个节点
 8049a66:   83 ec 08                sub    $0x8,%esp
 8049a69:   8d 45 c4                lea    -0x3c(%ebp),%eax
 8049a6c:   50                      push   %eax
 8049a6d:   ff 75 a4                pushl  -0x5c(%ebp)
 8049a70:   e8 da 02 00 00          call   8049d4f <read_six_numbers>//读取六个数字
 8049a75:   83 c4 10                add    $0x10,%esp
 8049a78:   85 c0                   test   %eax,%eax
 8049a7a:   75 0a                   jne    8049a86 <phase_6+0x42>
 8049a7c:   b8 00 00 00 00          mov    $0x0,%eax
 8049a81:   e9 37 01 00 00          jmp    8049bbd <phase_6+0x179>
 8049a86:   c7 45 b8 00 00 00 00    movl   $0x0,-0x48(%ebp)
 8049a8d:   eb 60                   jmp    8049aef <phase_6+0xab>
 8049a8f:   8b 45 b8                mov    -0x48(%ebp),%eax
 8049a92:   8b 44 85 c4             mov    -0x3c(%ebp,%eax,4),%eax
 8049a96:   85 c0                   test   %eax,%eax
 8049a98:   7e 0c                   jle    8049aa6 <phase_6+0x62>
 8049a9a:   8b 45 b8                mov    -0x48(%ebp),%eax
 8049a9d:   8b 44 85 c4             mov    -0x3c(%ebp,%eax,4),%eax
 8049aa1:   83 f8 06                cmp    $0x6,%eax
 8049aa4:   7e 0f                   jle    8049ab5 <phase_6+0x71>
 8049aa6:   e8 49 06 00 00          call   804a0f4 <explode_bomb>
 8049aab:   b8 00 00 00 00          mov    $0x0,%eax
 8049ab0:   e9 08 01 00 00          jmp    8049bbd <phase_6+0x179>
 8049ab5:   8b 45 b8                mov    -0x48(%ebp),%eax
 8049ab8:   83 c0 01                add    $0x1,%eax
 8049abb:   89 45 bc                mov    %eax,-0x44(%ebp)
 8049abe:   eb 25                   jmp    8049ae5 <phase_6+0xa1>
 8049ac0:   8b 45 b8                mov    -0x48(%ebp),%eax
 8049ac3:   8b 54 85 c4             mov    -0x3c(%ebp,%eax,4),%edx
 8049ac7:   8b 45 bc                mov    -0x44(%ebp),%eax
 8049aca:   8b 44 85 c4             mov    -0x3c(%ebp,%eax,4),%eax
 8049ace:   39 c2                   cmp    %eax,%edx
 8049ad0:   75 0f                   jne    8049ae1 <phase_6+0x9d>
 8049ad2:   e8 1d 06 00 00          call   804a0f4 <explode_bomb>
 8049ad7:   b8 00 00 00 00          mov    $0x0,%eax
 8049adc:   e9 dc 00 00 00          jmp    8049bbd <phase_6+0x179>
 8049ae1:   83 45 bc 01             addl   $0x1,-0x44(%ebp)
```

对于每个结点内容的处理

```
8049ae5:   83 7d bc 05              cmpl   $0x5,-0x44(%ebp)
8049ae9:   7e d5                    jle    8049ac0 <phase_6+0x7c>
8049aeb:   83 45 b8 01              addl   $0x1,-0x48(%ebp)
8049aef:   83 7d b8 05              cmpl   $0x5,-0x48(%ebp)
8049af3:   7e 9a                    jle    8049a8f <phase_6+0x4b>
8049af5:   c7 45 b8 00 00 00 00     movl   $0x0,-0x48(%ebp)
8049afc:   eb 36                    jmp    8049b34 <phase_6+0xf0>
8049afe:   8b 45 c0                 mov    -0x40(%ebp),%eax
8049b01:   89 45 b4                 mov    %eax,-0x4c(%ebp)
8049b04:   c7 45 bc 01 00 00 00     movl   $0x1,-0x44(%ebp)
8049b0b:   eb 0d                    jmp    8049b1a <phase_6+0xd6>
8049b0d:   8b 45 b4                 mov    -0x4c(%ebp),%eax
8049b10:   8b 40 08                 mov    0x8(%eax),%eax
8049b13:   89 45 b4                 mov    %eax,-0x4c(%ebp)
8049b16:   83 45 bc 01              addl   $0x1,-0x44(%ebp)
8049b1a:   8b 45 b8                 mov    -0x48(%ebp),%eax
8049b1d:   8b 44 85 c4              mov    -0x3c(%ebp,%eax,4),%eax
8049b21:   39 45 bc                 cmp    %eax,-0x44(%ebp)
8049b24:   7c e7                    jl     8049b0d <phase_6+0xc9>
8049b26:   8b 45 b8                 mov    -0x48(%ebp),%eax
8049b29:   8b 55 b4                 mov    -0x4c(%ebp),%edx
8049b2c:   89 54 85 dc              mov    %edx,-0x24(%ebp,%eax,4)
8049b30:   83 45 b8 01              addl   $0x1,-0x48(%ebp)
8049b34:   83 7d b8 05              cmpl   $0x5,-0x48(%ebp)
8049b38:   7e c4                    jle    8049afe <phase_6+0xba>
8049b3a:   8b 45 dc                 mov    -0x24(%ebp),%eax
8049b3d:   89 45 c0                 mov    %eax,-0x40(%ebp)
8049b40:   8b 45 c0                 mov    -0x40(%ebp),%eax
8049b43:   89 45 b4                 mov    %eax,-0x4c(%ebp)
8049b46:   c7 45 b8 01 00 00 00     movl   $0x1,-0x48(%ebp)
8049b4d:   eb 1a                    jmp    8049b69 <phase_6+0x125>
8049b4f:   8b 45 b8                 mov    -0x48(%ebp),%eax
8049b52:   8b 54 85 dc              mov    -0x24(%ebp,%eax,4),%edx
8049b56:   8b 45 b4                 mov    -0x4c(%ebp),%eax
8049b59:   89 50 08                 mov    %edx,0x8(%eax)
8049b5c:   8b 45 b4                 mov    -0x4c(%ebp),%eax
8049b5f:   8b 40 08                 mov    0x8(%eax),%eax
8049b62:   89 45 b4                 mov    %eax,-0x4c(%ebp)
8049b65:   83 45 b8 01              addl   $0x1,-0x48(%ebp)
8049b69:   83 7d b8 05              cmpl   $0x5,-0x48(%ebp)
8049b6d:   7e e0                    jle    8049b4f <phase_6+0x10b>
8049b6f:   8b 45 b4                 mov    -0x4c(%ebp),%eax
8049b72:   c7 40 08 00 00 00 00     movl   $0x0,0x8(%eax)
8049b79:   8b 45 c0                 mov    -0x40(%ebp),%eax
8049b7c:   89 45 b4                 mov    %eax,-0x4c(%ebp)
8049b7f:   c7 45 b8 00 00 00 00     movl   $0x0,-0x48(%ebp)
8049b86:   eb 2a                    jmp    8049bb2 <phase_6+0x16e>
```

对于每个结点内容的处理

```
8049b88:   8b 45 b4              mov    -0x4c(%ebp),%eax
8049b8b:   8b 10                 mov    (%eax),%edx
8049b8d:   8b 45 b4              mov    -0x4c(%ebp),%eax
8049b90:   8b 40 08              mov    0x8(%eax),%eax
8049b93:   8b 00                 mov    (%eax),%eax
8049b95:   39 c2                 cmp    %eax,%edx
8049b97:   7d 0c                 jge    8049ba5 <phase_6+0x161>
8049b99:   e8 56 05 00 00        call   804a0f4 <explode_bomb>
8049b9e:   b8 00 00 00 00        mov    $0x0,%eax
8049ba3:   eb 18                 jmp    8049bbd <phase_6+0x179>
8049ba5:   8b 45 b4              mov    -0x4c(%ebp),%eax
8049ba8:   8b 40 08              mov    0x8(%eax),%eax
8049bab:   89 45 b4              mov    %eax,-0x4c(%ebp)
8049bae:   83 45 b8 01           addl   $0x1,-0x48(%ebp)
8049bb2:   83 7d b8 04           cmpl   $0x4,-0x48(%ebp)
8049bb6:   7e d0                 jle    8049b88 <phase_6+0x144>
8049bb8:   b8 01 00 00 00        mov    $0x1,%eax
8049bbd:   8b 4d f4              mov    -0xc(%ebp),%ecx
8049bc0:   65 33 0d 14 00 00 00  xor    %gs:0x14,%ecx
8049bc7:   74 05                 je     8049bce <phase_6+0x18a>
8049bc9:   e8 c2 f5 ff ff        call   8049190 <__stack_chk_fail@plt>
8049bce:   c9                    leave
8049bcf:   c3                    ret
```

初观代码，挺长的。浏览一遍发现并有取得什么有用的信息，大概知道这是一个链表，并且知道了其中的一个结点可以查阅，然后就是我们需要输入六个数字。

从后面的长串代码中其实我们可以发现他只是实现了一个对于结点内容的大小排序所以题目就变得很简单了，我们只需要从第一个结点开始逐个查阅就可以了，运用指令 p/x *adress@3

```
(gdb) p/x *0x804d108@3
$1 = {0x7, 0x1, 0x804d0fc}
(gdb) p/x *0x804d0fc@3
$2 = {0x2, 0x2, 0x804d0f0}
(gdb) p/x *0x804d0f0@3
$3 = {0x9, 0x3, 0x804d0e4}
(gdb) p/x *0x804d0e4@3
$4 = {0x4, 0x4, 0x804d0d8}
(gdb) p/x *0x804d0d8@3
$5 = {0x3, 0x5, 0x804d0cc}
(gdb) p/x *0x804d0cc@3
$6 = {0x0, 0x6, 0x0}
(gdb)
```

如上图所示就是我们查到的结果。其中第一个为该结点存储的值，第二个为该结点的序号，第三个为下一个结点的地址。由于代码中给出的机制是按照节点中的值从大到小进行排序，所以我们需要对结点序号按照其对应的值进行从大到小的排列，即答案为：3 1 4 5 2 6

```
3 1 4 5 2 6
Congratulations! You've defused the bomb!
```

拆弹成功！

Secret_Phase:二叉树

```
08049bd0 <fun7>:
 8049bd0:   f3 0f 1e fb             endbr32
 8049bd4:   55                      push   %ebp
 8049bd5:   89 e5                   mov    %esp,%ebp
 8049bd7:   83 ec 08                sub    $0x8,%esp
 8049bda:   83 7d 08 00             cmpl   $0x0,0x8(%ebp)
 8049bde:   75 07                   jne    8049be7 <fun7+0x17>
 8049be0:   b8 ff ff ff ff          mov    $0xffffffff,%eax
 8049be5:   eb 4e                   jmp    8049c35 <fun7+0x65>
 8049be7:   8b 45 08                mov    0x8(%ebp),%eax
 8049bea:   8b 00                   mov    (%eax),%eax
 8049bec:   39 45 0c                cmp    %eax,0xc(%ebp)
 8049bef:   7d 19                   jge    8049c0a <fun7+0x3
 8049bf1:   8b 45 08                mov    0x8(%ebp),%eax
 8049bf4:   8b 40 04                mov    0x4(%eax),%eax
 8049bf7:   83 ec 08                sub    $0x8,%esp
 8049bfa:   ff 75 0c                pushl  0xc(%ebp)
 8049bfd:   50                      push   %eax
 8049bfe:   e8 cd ff ff ff          call   8049bd0 <fun7>
 8049c03:   83 c4 10                add    $0x10,%esp
 8049c06:   01 c0                   add    %eax,%eax
 8049c08:   eb 2b                   jmp    8049c35 <fun7+0x65>
 8049c0a:   8b 45 08                mov    0x8(%ebp),%eax
 8049c0d:   8b 00                   mov    (%eax),%eax
 8049c0f:   39 45 0c                cmp    %eax,0xc(%ebp)
 8049c12:   75 07                   jne    8049c1b <fun7+0x4b>
 8049c14:   b8 00 00 00 00          mov    $0x0,%eax
 8049c19:   eb 1a                   jmp    8049c35 <fun7+0x65>
 8049c1b:   8b 45 08                mov    0x8(%ebp),%eax
 8049c1e:   8b 40 08                mov    0x8(%eax),%eax
 8049c21:   83 ec 08                sub    $0x8,%esp
 8049c24:   ff 75 0c                pushl  0xc(%ebp)
 8049c27:   50                      push   %eax
 8049c28:   e8 a3 ff ff ff          call   8049bd0 <fun7>
 8049c2d:   83 c4 10                add    $0x10,%esp
 8049c30:   01 c0                   add    %eax,%eax
 8049c32:   83 c0 01                add    $0x1,%eax
 8049c35:   c9                      leave
 8049c36:   c3                      ret
```

二叉树的实现

```
08049c37 <secret_phase>:
 8049c37:   f3 0f 1e fb             endbr32
 8049c3b:   55                      push   %ebp
 8049c3c:   89 e5                   mov    %esp,%ebp
 8049c3e:   83 ec 18                sub    $0x18,%esp
 8049c41:   e8 67 03 00 00          call   8049fad <read_line>
 8049c46:   89 45 ec                mov    %eax,-0x14(%ebp)
 8049c49:   83 ec 0c                sub    $0xc,%esp
 8049c4c:   ff 75 ec                pushl  -0x14(%ebp)
 8049c4f:   e8 cc f5 ff ff          call   8049220 <atoi@plt>
 8049c54:   83 c4 10                add    $0x10,%esp
 8049c57:   89 45 f0                mov    %eax,-0x10(%ebp)
 8049c5a:   83 7d f0 00             cmpl   $0x0,-0x10(%ebp)
 8049c5e:   7e 09                   jle    8049c69 <secret_phase+0x32>
 8049c60:   81 7d f0 e9 03 00 00    cmpl   $0x3e9,-0x10(%ebp)
 8049c67:   7e 0c                   jle    8049c75 <secret_phase+0x3e>
 8049c69:   e8 86 04 00 00          call   804a0f4 <explode_bomb>
 8049c6e:   b8 00 00 00 00          mov    $0x0,%eax
 8049c73:   eb 42                   jmp    8049cb7 <secret_phase+0x80>
 8049c75:   83 ec 08                sub    $0x8,%esp
 8049c78:   ff 75 f0                pushl  -0x10(%ebp)
 8049c7b:   68 bc d1 04 08          push   $0x804d1bc
 8049c80:   e8 4b ff ff ff          call   8049bd0 <fun7>
 8049c85:   83 c4 10                add    $0x10,%esp
 8049c88:   89 45 f4                mov    %eax,-0xc(%ebp)
 8049c8b:   83 7d f4 06             cmpl   $0x6,-0xc(%ebp)
 8049c8f:   74 0c                   je     8049c9d <secret_phase+0x66>
 8049c91:   e8 5e 04 00 00          call   804a0f4 <explode_bomb>
 8049c96:   b8 00 00 00 00          mov    $0x0,%eax
 8049c9b:   eb 1a                   jmp    8049cb7 <secret_phase+0x80>
 8049c9d:   83 ec 0c                sub    $0xc,%esp
 8049ca0:   68 2c b2 04 08          push   $0x804b22c
 8049ca5:   e8 06 f5 ff ff          call   80491b0 <puts@plt>
 8049caa:   83 c4 10                add    $0x10,%esp
 8049cad:   e8 6f 04 00 00          call   804a121 <phase_defused>
 8049cb2:   b8 01 00 00 00          mov    $0x1,%eax
 8049cb7:   c9                      leave
 8049cb8:   c3                      ret
```

```
804a125:   55                      push   %ebp
804a126:   89 e5                   mov    %esp,%ebp
804a128:   83 ec 68                sub    $0x68,%esp
804a12b:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
804a131:   89 45 f4                mov    %eax,-0xc(%ebp)
804a134:   31 c0                   xor    %eax,%eax
804a136:   a1 2c d2 04 08          mov    0x804d22c,%eax
804a13b:   83 f8 07                cmp    $0x7,%eax
804a13e:   75 77                   jne    804a1b7 <phase_defused+0x96>
804a140:   83 ec 0c                sub    $0xc,%esp
804a143:   8d 45 a4                lea    -0x5c(%ebp),%eax
804a146:   50                      push   %eax
804a147:   8d 45 9c                lea    -0x64(%ebp),%eax
804a14a:   50                      push   %eax
804a14b:   8d 45 98                lea    -0x68(%ebp),%eax
804a14e:   50                      push   %eax
804a14f:   68 2a b3 04 08          push   $0x804b32a
804a154:   68 80 d3 04 08          push   $0x804d380
804a159:   e8 92 f0 ff ff          call   80491f0 <__isoc99_sscanf@plt>
804a15e:   83 c4 20                add    $0x20,%esp
804a161:   89 45 a0                mov    %eax,-0x60(%ebp)
804a164:   83 7d a0 03             cmpl   $0x3,-0x60(%ebp)
804a168:   75 3d                   jne    804a1a7 <phase_defused+0x86>
804a16a:   83 ec 08                sub    $0x8,%esp
804a16d:   68 33 b3 04 08          push   $0x804b333
804a172:   8d 45 a4                lea    -0x5c(%ebp),%eax
804a175:   50                      push   %eax
804a176:   e8 f9 fc ff ff          call   8049e74 <strings_not_equal>
804a17b:   83 c4 10                add    $0x10,%esp
804a17e:   85 c0                   test   %eax,%eax
804a180:   75 25                   jne    804a1a7 <phase_defused+0x86>
804a182:   83 ec 0c                sub    $0xc,%esp
804a185:   68 3c b3 04 08          push   $0x804b33c
804a18a:   e8 21 f0 ff ff          call   80491b0 <puts@plt>
804a18f:   83 c4 10                add    $0x10,%esp
804a192:   83 ec 0c                sub    $0xc,%esp
804a195:   68 64 b3 04 08          push   $0x804b364
804a19a:   e8 11 f0 ff ff          call   80491b0 <puts@plt>
804a19f:   83 c4 10                add    $0x10,%esp
804a1a2:   e8 90 fa ff ff          call   8049c37 <secret_phase>
804a1a7:   83 ec 0c                sub    $0xc,%esp
804a1aa:   68 9c b3 04 08          push   $0x804b39c
804a1af:   e8 fc ef ff ff          call   80491b0 <puts@plt>
804a1b4:   83 c4 10                add    $0x10,%esp
804a1b7:   90                      nop
804a1b8:   8b 45 f4                mov    -0xc(%ebp),%eax
804a1bb:   65 33 05 14 00 00 00    xor    %gs:0x14,%eax
804a1c2:   74 05                   je     804a1c9 <phase_defused+0xa8>
804a1c4:   e8 c7 ef ff ff          call   8049190 <__stack_chk_fail@plt>
```

Phase_defused 详细阐述了如何进入 secret_phase

首先我们需要明确如何才能进入 secret_phase：

我们通过观察阅读 phase_defused 我们会发现，只有当前面答对六道题，且在 phase4 中输入了正确的字符串，才可以进入 secret_phase。

```
804a16d:   68 33 b3 04 08          push   $0x804b333
804a172:   8d 45 a4                lea    -0x5c(%ebp),%eax
804a175:   50                      push   %eax
804a176:   e8 f9 fc ff ff          call   8049e74 <strings_not_equal>
```

我们在这里看到了<strings_not_equal>函数，所以我们判断 phase4 除了要输入两个数字，还应该输入一个正确的字符串，才可以进入隐藏炸弹，所以我们查看地址 0x804b333：

```
gdb) x/s 0x804b333
x804b333:       "MLgYG"
```

然后将之前的答案都输入，会得到：

```
Welcome to my fiendish little bomb. You have 7 phases wit
which to blow yourself up. Have a nice day!
Well done! You seem to have warmed up!
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...
```

我们成功进入了隐藏炸弹，下面分析如何拆解。

通过对于 func7 的阅读，我们知道这其实是一个二叉树的结构，左数为 2k，右数为 2k+1，而最终的答案为 6，因此它的路径为：左-右-右。接下来我们就可以调试了：

```
(gdb) p/x *0x804d1bc@3
$1 = {0x24, 0x804d1b0, 0x804d1a4}
(gdb) p/x *0x804d1b0@3
$2 = {0x8, 0x804d180, 0x804d198}
(gdb) p/x *0x804d198@3
$3 = {0x16, 0x804d12c, 0x804d144}
(gdb) p/x *0x804d144@3
$4 = {0x23, 0x0, 0x0}
(gdb)
```

因此我们需要输入的值就是最开始的结点里存储的值：0x23，即 35

```
But finding it and solving it are quite different...
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```

拆弹完成！

至此，拆弹全部结束！！！