

## 准备工作

解压文件，cd 进入目录。目录下有 bomb, bomb.c 和 README 三个文件。

使用命令 `objdump -d bomb > bomb.txt` 得到汇编指令。

使用 gdb 加载 bomb 文件：`gdb bomb`

```
total 120
drwx----- 6  root 4096 Nov  6 11:31 ./
drwxr-xr-x 685 root 24576 Oct 30 13:28 ../
-rw----- 1  stus 9241 Nov  6 11:31 .bash_history
-rw-r--r-- 1  stus 220 Sep  1 2015 .bash_logout
-rw-r--r-- 1  stus 3771 Sep  1 2015 .bashrc
drwxr-xr-x 2  stus 4096 Nov  6 11:31 bomb560/
-rw-r--r-- 1  root 40960 Nov  2 19:30 bomb560.tar
drwx----- 2  stus 4096 Oct 12 18:50 .cache/
drwxr-xr-x 5  stus 4096 Nov  2 19:21 lab/
-rw-r--r-- 1  stus 655 May 16 2017 .profile
-rw----- 1  stus 6172 Nov  6 11:31 .viminfo
drwxr-xr-x 3  stus 4096 Nov  2 18:58 .vscode-server/

total 144
drwxr-xr-x 2  stus 4096 Nov  6 11:31 ./
drwx----- 6  stus 4096 Nov  6 11:31 ../
-rwxr-xr-x 1  stus 31144 Nov  2 19:30 bomb*
-rw-r--r-- 1  stus 4069 Nov  2 19:30 bomb.c
-rw-r--r-- 1  stus 92878 Nov  6 10:18 bomb.txt
-rw-r--r-- 1  stus 101 Nov  6 11:29 in
-rw-r--r-- 1  stus 62 Nov  2 19:30 README
gdb bomb
```

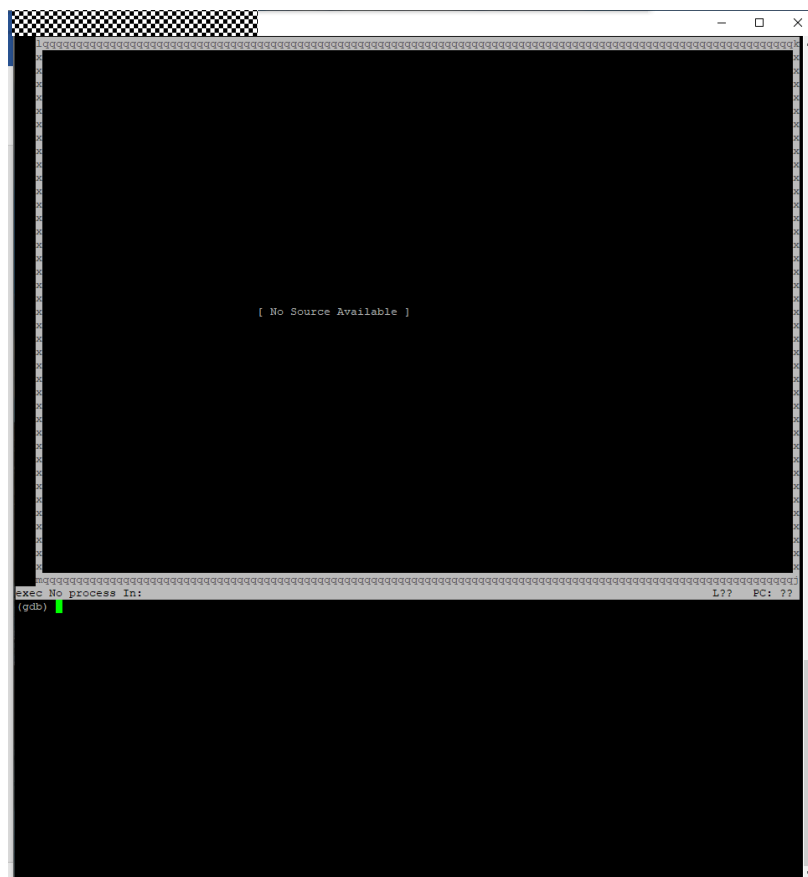
使用 `! 命令` 查看源代码：

```
(gdb) 1 74
69     printf("Welcome to my fiendish little bomb. You have 6 phases with\n");
70     printf("which to blow yourself up. Have a nice day!\n");
71
72     /* Hmm... Six phases must be more secure than one phase! */
73     input = read_line(); /* Get input */
74     phase_1(input);      /* Run the phase */
75     phase_defused();     /* Drat! They figured it out!
76                          * Let me know how they did it. */
77     printf("Phase 1 defused. How about the next one?\n");
78
79 (gdb)
79     /* The second phase is harder. No one will ever figure out
80      * how to defuse this... */
81     input = read_line();
82     phase_2(input);
83     phase_defused();
84     printf("That's number 2. Keep going!\n");
85
86     /* I guess this is too easy so far. Some more complex code will
87      * confuse people. */
88     input = read_line();
89
90 (gdb)
89     phase_3(input);
90     phase_defused();
91     printf("Halfway there!\n");
92
93     /* Oh yeah? Well, how good is your math? Try on this saucy problem! */
94     input = read_line();
95     phase_4(input);
96     phase_defused();
97     printf("So you got that one. Try this one.\n");
98
99 (gdb)
99     /* Round and 'round in memory we go, where we stop, the bomb blows! */
100    input = read_line();
101    phase_5(input);
102    phase_defused();
103    printf("Good work! On to the next...\n");
104
105    /* This phase will never be used, since no one will get past the
106     * earlier ones. But just in case, make this one extra hard. */
107    input = read_line();
108    phase_6(input);
109
110 (gdb)
```

在函数 `phase_1`, `phase_2`, `phase_3`, `phase_4`, `phase_5`, `phase_6`, `explode_bomb` 处分别断点

```
phase_0(input);
(gdb) b explode_bomb
Breakpoint 1 at 0x40172e
(gdb) b phase_1
Breakpoint 2 at 0x400f2d
(gdb) b phase_2
Breakpoint 3 at 0x400f49
(gdb) b phase_3
Breakpoint 4 at 0x400fb5
(gdb) b phase_4
Breakpoint 5 at 0x401156
(gdb) b phase_5
Breakpoint 6 at 0x4011c3
(gdb) b phase_6
Breakpoint 7 at 0x401204
```

使用 ctrl+C+A 开启 tui



## 1. layout asm

开启汇编视窗

```
0x400df6 <main>      push    %rbx
0x400df7 <main+1>     cmp     $0x1,%edi
0x400df8 <main+2>     jne     0x400e0c <main+22>
0x400dfc <main+6>     mov     0x20398d(&trip),%rax
0x400e03 <main+13>    mov     %rax,0x2039a6(&trip)
0x400e0a <main+20>    jmp     0x400e6f <main+121>
0x400e0c <main+22>    mov     %rax,%rbx
0x400e0f <main+25>    cmp     $0x2,%edi
0x400e12 <main+28>    jne     0x400e4e <main+88>
0x400e14 <main+30>    mov     0x2(&res),%rdi
0x400e18 <main+34>    mov     0x4002c4,%esi
0x400e1d <main+39>    callq   0x400c60 <fopen@plt>
0x400e22 <main+44>    mov     %rax,0x203987(&trip)
0x400e29 <main+51>    test   %rax,%rax
0x400e2c <main+54>    jne     0x400e6f <main+121>
0x400e2e <main+56>    mov     0x9(&rbx),%rcx
0x400e32 <main+60>    mov     (%rbx),%rdx
0x400e35 <main+63>    mov     0x4002c4,%esi
0x400e3a <main+68>    mov     $0x1,%edi
0x400e3f <main+73>    callq   0x400c50 <_printf_chk@plt>
0x400e44 <main+78>    mov     $0x0,%edi
0x400e49 <main+83>    callq   0x400c80 <exit@plt>
0x400e4e <main+88>    mov     (%rsi),%rdx
0x400e51 <main+91>    mov     0x4025c3,%esi
0x400e56 <main+96>    mov     $0x1,%edi
0x400e5b <main+101>   mov     $0x0,%eax
0x400e60 <main+106>   callq   0x400c50 <_printf_chk@plt>
0x400e65 <main+111>   mov     $0x0,%edi
0x400e6a <main+116>   callq   0x400c80 <exit@plt>
0x400e6f <main+121>   callq   0x4014c1 <initialize_bomb>
0x400e74 <main+126>   mov     0x402648,%edi
0x400e79 <main+131>   callq   0x400b70 <puts@plt>
0x400e7e <main+136>   mov     0x402688,%edi
0x400e83 <main+141>   callq   0x400b70 <puts@plt>
0x400e88 <main+146>   callq   0x4017a3 <read_line>
0x400e8d <main+151>   mov     %rax,%rdi
0x400e90 <main+154>   callq   0x400f2d <phase_1>
0x400e95 <main+159>   callq   0x4018c9 <phase_defused>
0x400e9a <main+164>   mov     0x4002c4,%edi
;~~~~~
$eax No process in:
(gdb) 
```

## 1. focus cmd

转移焦点到命令行。

使用 r 命令启动程序。

## 第一阶段

首先查看 phase\_1 函数的汇编命令。这一可以用 objdump 查看，也可以用 tui 或者 gdb 的 disas

```
0000000000400f2d <phase_1>:
400f2d: 48 83 c0 08          sub     $0x8,%rsp
400f31: be 10 27 40 00      mov     $0x402710,%esi
400f36: e8 1f 05 00 00      callq  40145a <strings_not_equal>
400f3b: 85 c0               test    %eax,%eax
400f3d: 74 05              je      400f44 <phase_1+0x17>
400f3f: e8 ea 07 00 00      callq  40172e <explode_bomb>
400f44: 48 83 c4 08          add     $0x8,%rsp
400f48: c3                 retq    
```

可以看到调用了个 strings\_not\_equal 函数，函数的两个参数%esi 和%edi 分别是比较的字符串和输入的字符串。比较结束后，判断返回值 eax 是否为 0，不相同爆炸。从逻辑上可以大概判断出，如果两个字符串完全相同，返回的结果一定是 0。

进一步来说，如果仔细去看 strings\_not\_equal 函数，可以发现它其中调用了字符串长度的函数，判断它是否相同，如果相同再逐一进行比较。这样就能确认只要我们的输入和它一样就能通过了。

```

719 000000000040145a <strings_not_equal>:
720 40145a: 41 54          push    %r12
721 40145c: 55            push    %rbp
722 40145d: 53            push    %rbx
723 40145e: 48 89 fb      mov     %rdi,%rbx
724 401461: 48 89 f5      mov     %rsi,%rbp
725 401464: e8 d3 ff ff ff callq   40143c <string_length>
726 401469: 41 89 c4      mov     %eax,%r12d
727 40146c: 48 89 ef      mov     %rbp,%rdi
728 40146f: e8 c8 ff ff ff callq   40143c <string_length>
729 401474: ba 01 00 00 00 mov     $0x1,%edx
730 401479: 41 39 c4      cmp     %eax,%r12d
731 40147c: 75 3c         jne     4014ba <strings_not_equal+0x60>
732 40147e: 0f b6 03      movzbl  (%rbx),%eax
733 401481: 84 c0         test    %al,%al
734 401483: 74 22         je      4014a7 <strings_not_equal+0x4d>
735 401485: 3a 45 00      cmp     0x0(%rbp),%al
736 401488: 74 07         je      401491 <strings_not_equal+0x37>
737 40148a: eb 22         jmp     4014ae <strings_not_equal+0x54>
738 40148c: 3a 45 00      cmp     0x0(%rbp),%al
739 40148f: 75 24         jne     4014b5 <strings_not_equal+0x5b>
740 401491: 48 83 c3 01   add     $0x1,%rbx
741 401495: 48 83 c5 01   add     $0x1,%rbp
742 401499: 0f b6 03      movzbl  (%rbx),%eax
743 40149c: 84 c0         test    %al,%al
744 40149e: 75 ec         jne     40148c <strings_not_equal+0x32>
745 4014a0: ba 00 00 00 00 mov     $0x0,%edx
746 4014a5: eb 13         jmp     4014ba <strings_not_equal+0x60>
747 4014a7: ba 00 00 00 00 mov     $0x0,%edx
748 4014ac: eb 0c         jmp     4014ba <strings_not_equal+0x60>
749 4014ae: ba 01 00 00 00 mov     $0x1,%edx
750 4014b3: eb 05         jmp     4014ba <strings_not_equal+0x60>
751 4014b5: ba 01 00 00 00 mov     $0x1,%edx
752 4014ba: 89 d0         mov     %edx,%eax
753 4014bc: 5b           pop     %rbx

```

使用 x/s 命令可以查看内存中的值。

#### 1. layout \$esi

```

(gdb) x/s $esi
0x402710:      "Houses will begat jobs, jobs will begat houses."
(gdb) x/s $edi
0x6047c0 <input_strings>:      "Houses will begat jobs, jobs will begat houses."

```

因此第一个的答案就是上面的字符串。

## 第二阶段

```

0000000000400f49 <phase_2>:
 400f49: 55                push    %rbp
 400f4a: 53                push    %rbx
 400f4b: 48 83 ec 28       sub     $0x28,%rsp
 400f4f: 64 48 8b 04 25 28 00 mov     %fs:0x28,%rax
 400f56: 00 00
 400f58: 48 89 44 24 18     mov     %rax,0x18(%rsp)
 400f5d: 31 c0             xor     %eax,%eax
 400f5f: 48 89 e6          mov     %rsp,%rsi
 400f62: e8 fd 07 00 00     callq   401764 <read_six_numbers>
 400f67: 83 3c 24 00       cmpl    $0x0,(%rsp)
 400f6b: 75 07             jne     400f74 <phase_2+0x2b>
 400f6d: 83 7c 24 04 01     cmpl    $0x1,0x4(%rsp)
 400f72: 74 05             je      400f79 <phase_2+0x30>
 400f74: e8 b5 07 00 00     callq   40172e <explode_bomb>
 400f79: 48 89 e3          mov     %rsp,%rbx
 400f7c: 48 8d 6c 24 10     lea     0x10(%rsp),%rbp
 400f81: 8b 43 04          mov     0x4(%rbx),%eax
 400f84: 03 03             add     (%rbx),%eax
 400f86: 39 43 08          cmp     %eax,0x8(%rbx)
 400f89: 74 05             je      400f90 <phase_2+0x47>
 400f8b: e8 9e 07 00 00     callq   40172e <explode_bomb>
 400f90: 48 83 c3 04       add     $0x4,%rbx
 400f94: 48 39 eb          cmp     %rbp,%rbx
 400f97: 75 e8             jne     400f81 <phase_2+0x38>
 400f99: 48 8b 44 24 18     mov     0x18(%rsp),%rax
 400f9e: 64 48 33 04 25 28 00 xor     %fs:0x28,%rax
 400fa5: 00 00
 400fa7: 74 05             je      400fae <phase_2+0x65>
 400fa9: e8 e2 fb ff ff     callq   400b90 <__stack_chk_fail@plt>
 400fae: 48 83 c4 28       add     $0x28,%rsp
 400fb2: 5b                pop     %rbx
 400fb3: 5d                pop     %rbp
 400fb4: c3                retq

```

首先看输入部分的处理。函数调用了 read\_six\_numbers 的函数，说明需要输入 6 个数字。然后再看输入参数的寄存器。这边只传了%rsp 栈指针作为参数进入，说明读入的数据肯定是存在栈上的。

下面看一下 read\_six\_numbers 函数

```

0x401764 <read_six_numbers> sub     $0x8,%rsp
0x401768 <read_six_numbers+4> mov     %rsi,%rdx
0x40176b <read_six_numbers+7> lea     0x4(%rsi),%rcx
0x40176f <read_six_numbers+11> lea     0x14(%rsi),%rax
0x401773 <read_six_numbers+15> push    %rax
0x401774 <read_six_numbers+16> lea     0x10(%rsi),%rax
0x401778 <read_six_numbers+20> push    %rax
0x401779 <read_six_numbers+21> lea     0xc(%rsi),%r9
0x40177d <read_six_numbers+25> lea     0x8(%rsi),%r8
0x401781 <read_six_numbers+29> mov     $0x402a41,%esi
0x401786 <read_six_numbers+34> mov     $0x0,%eax
0x40178b <read_six_numbers+39> callq   0x400c40 <__isoc99_sscanf@plt>
0x401790 <read_six_numbers+44> add     $0x10,%rsp
0x401794 <read_six_numbers+48> cmp     $0x5,%eax
0x401797 <read_six_numbers+51> jg      0x40179e <read_six_numbers+58>
0x401799 <read_six_numbers+53> callq   0x40172e <explode_bomb>
0x40179e <read_six_numbers+58> add     $0x8,%rsp
0x4017a2 <read_six_numbers+62> retq

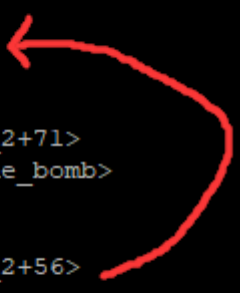
```

这里用了 sscanf 函数。从函数声明可知，sscanf 的第一个参数是字符串，第二个参数是格式，后面是不定长参数地址。很明显，地址 0x402a41 存的就是格式：

```
(gdb) x/s 0x402a41
0x402a41: "%d %d %d %d %d %d"
```

可见的确是读六个 int。六个参数分别存在 %rsp,%rsp+0x4,%rsp+0x8,%rsp+0xc,%rsp+0x10,%rsp+0x14。下面看判断的部分。

```
0x400f67 <phase_2+30>  cmpl    $0x0, (%rsp)
0x400f6b <phase_2+34>  jne     0x400f74 <phase_2+43>
0x400f6d <phase_2+36>  cmpl    $0x1, 0x4(%rsp)
0x400f72 <phase_2+41>  je      0x400f79 <phase_2+48>
0x400f74 <phase_2+43>  callq   0x40172e <explode_bomb>
0x400f79 <phase_2+48>  mov     %rsp, %rbx
0x400f7c <phase_2+51>  lea     0x10(%rsp), %rbp
0x400f81 <phase_2+56>  mov     0x4(%rbx), %eax
0x400f84 <phase_2+59>  add     (%rbx), %eax
0x400f86 <phase_2+61>  cmp     %eax, 0x8(%rbx)
0x400f89 <phase_2+64>  je      0x400f90 <phase_2+71>
0x400f8b <phase_2+66>  callq   0x40172e <explode_bomb>
0x400f90 <phase_2+71>  add     $0x4, %rbx
0x400f94 <phase_2+75>  cmp     %rbp, %rbx
0x400f97 <phase_2+78>  jne     0x400f81 <phase_2+56>
0x400f99 <phase_2+80>  mov     0x18(%rsp), %rax
0x400f9e <phase_2+85>  xor     %fs:0x28, %rax
0x400fa7 <phase_2+94>  je      0x400fae <phase_2+101>
0x400fa9 <phase_2+96>  callq   0x400b90 <__stack_chk_fail@plt>
0x400fae <phase_2+101> add     $0x28, %rsp
0x400fb2 <phase_2+105> pop     %rbx
0x400fb3 <phase_2+106> pop     %rbp
0x400fb4 <phase_2+107> retq
```



不难看出，这是一个循环。首先判断第一个数是否为 0，第二个数是否为 1。然后在循环里判断第 n 个数是否等于第 n-1 个数加上第 n-2 个数。因此，这是一个斐波那契数列。可以推导出，结果为 0,1,1,2,3,5。

### 第三阶段

这一部分代码比较长。首先看输入部分的处理：

```
0x400fc7 <phase_3+18>  xor     %eax, %eax
0x400fc9 <phase_3+20>  lea     0x14(%rsp), %r8
0x400fce <phase_3+25>  lea     0xf(%rsp), %rcx
0x400fd3 <phase_3+30>  lea     0x10(%rsp), %rdx
0x400fd8 <phase_3+35>  mov     $0x402766, %esi
0x400fdd <phase_3+40>  callq   0x400c40 <__isoc99_sscanf@plt>
```

调用函数 sscanf(input\_strings,%esi,%rsp+0x10,%rsp+0xf,%rsp+0x14)。

下面不妨假设三个参数分别为 a(%rsp+0x10),b(%rsp+0xf),c(%rsp+0x14)

看一下输入格式：

```
(gdb) x/s 0x402766
0x402766: "%d %c %d"
```

是一个数字，一个字符和一个数字。

读入后，首先比较了一下读入变量个数，然后判断了一下参数 a 是否属于 0~7。接着，使用 jmpq 指令根据参数 a 进行跳转。这是一个间接跳转，跳转的地址是根据 0x402780 的跳转表来的。

```

0x400fe2 <phase_3+45>    cmp     $0x2,%eax
0x400fe5 <phase_3+48>    jg      0x400fec <phase_3+55>
0x400fe7 <phase_3+50>    callq   0x40172e <explode_bomb>
0x400fec <phase_3+55>    cmpl    $0x7,0x10(%rsp)
0x400ff1 <phase_3+60>    ja      0x4010ec <phase_3+311>
0x400ff7 <phase_3+66>    mov     0x10(%rsp),%eax
0x400ffb <phase_3+70>    jmpq    *0x402780(,%rax,8)

```

跳转表:

```

(gdb) x/lxg 0x402780
0x402780: 0x0000000000401002
0x402788: 0x0000000000401024
0x402790: 0x0000000000401046
0x402798: 0x0000000000401068
0x4027a0: 0x0000000000401083
0x4027a8: 0x000000000040109b
0x4027b0: 0x00000000004010b6
0x4027b8: 0x00000000004010d1

```

分别对应下面几个入口

```

401002: b8 64 00 00 00    mov     $0x64,%eax
401007: 81 7c 24 14 cd 00 00    cmpl    $0xcd,0x14(%rsp)
40100e: 00
40100f: 0f 84 e1 00 00 00    je      4010f6 <phase_3+0x141>
401015: e8 14 07 00 00    callq   40172e <explode_bomb>
40101a: b8 64 00 00 00    mov     $0x64,%eax
40101f: e9 d2 00 00 00    jmpq    4010f6 <phase_3+0x141>
401024: b8 61 00 00 00    mov     $0x61,%eax
401029: 81 7c 24 14 6a 03 00    cmpl    $0x36a,0x14(%rsp)
401030: 00
401031: 0f 84 bf 00 00 00    je      4010f6 <phase_3+0x141>
401037: e8 f2 06 00 00    callq   40172e <explode_bomb>
40103c: b8 61 00 00 00    mov     $0x61,%eax
401041: e9 b0 00 00 00    jmpq    4010f6 <phase_3+0x141>
401046: b8 67 00 00 00    mov     $0x67,%eax
40104b: 81 7c 24 14 17 01 00    cmpl    $0x117,0x14(%rsp)
401052: 00
401053: 0f 84 9d 00 00 00    je      4010f6 <phase_3+0x141>
401059: e8 d0 06 00 00    callq   40172e <explode_bomb>
40105e: b8 67 00 00 00    mov     $0x67,%eax
401063: e9 8e 00 00 00    jmpq    4010f6 <phase_3+0x141>
401068: b8 6a 00 00 00    mov     $0x6a,%eax
40106d: 81 7c 24 14 1b 03 00    cmpl    $0x31b,0x14(%rsp)
401074: 00
401075: 74 7f             je      4010f6 <phase_3+0x141>
401077: e8 b2 06 00 00    callq   40172e <explode_bomb>
40107c: b8 6a 00 00 00    mov     $0x6a,%eax
401081: eb 73             jmp     4010f6 <phase_3+0x141>
401083: b8 68 00 00 00    mov     $0x68,%eax
401088: 83 7c 24 14 54    cmpl    $0x54,0x14(%rsp)
40108d: 74 67             je      4010f6 <phase_3+0x141>
40108f: e8 9a 06 00 00    callq   40172e <explode_bomb>
401094: b8 68 00 00 00    mov     $0x68,%eax
401099: eb 5b             jmp     4010f6 <phase_3+0x141>
40109b: b8 63 00 00 00    mov     $0x63,%eax
4010a0: 81 7c 24 14 9f 02 00    cmpl    $0x29f,0x14(%rsp)
4010a7: 00
4010a8: 74 4c             je      4010f6 <phase_3+0x141>
4010aa: e8 7f 06 00 00    callq   40172e <explode_bomb>
4010af: b8 63 00 00 00    mov     $0x63,%eax
4010b4: eb 40             jmp     4010f6 <phase_3+0x141>
4010b6: b8 67 00 00 00    mov     $0x67,%eax
4010bb: 81 7c 24 14 1f 03 00    cmpl    $0x31f,0x14(%rsp)
4010c2: 00
4010c3: 74 31             je      4010f6 <phase_3+0x141>
4010c5: e8 64 06 00 00    callq   40172e <explode_bomb>
4010ca: b8 67 00 00 00    mov     $0x67,%eax
4010cf: eb 25             jmp     4010f6 <phase_3+0x141>
4010d1: b8 66 00 00 00    mov     $0x66,%eax
4010d6: 81 7c 24 14 87 01 00    cmpl    $0x187,0x14(%rsp)
4010dd: 00
4010de: 74 16             je      4010f6 <phase_3+0x141>
4010e0: e8 49 06 00 00    callq   40172e <explode_bomb>
4010e5: b8 66 00 00 00    mov     $0x66,%eax
4010ea: eb 0a             jmp     4010f6 <phase_3+0x141>
4010ec: e8 3d 06 00 00    callq   40172e <explode_bomb>
4010f1: b8 64 00 00 00    mov     $0x64,%eax
4010f6: 3a 44 24 0f       cmp     0xf(%rsp),%al
4010fa: 74 05             je      401101 <phase_3+0x14c>
4010fc: e8 2d 06 00 00    callq   40172e <explode_bomb>
401101: 48 8b 44 24 18    mov     0x18(%rsp),%rax

```



下面只说明当第一个参数为 2，入口为 0x401046 时的情况：

```
0x401046 <phase_3+145> mov $0x67,%eax
0x40104b <phase_3+150> cmpl $0x117,0x14(%rsp)
0x401053 <phase_3+158> je 0x4010f6 <phase_3+321>
0x401059 <phase_3+164> callq 0x40172e <explode_bomb>
0x40105e <phase_3+169> mov $0x67,%eax
0x401063 <phase_3+174> jmpq 0x4010f6 <phase_3+321>
0x401069 <phase_3+179> mov $0x67,%eax
0x4010f6 <phase_3+321> cmp 0xf(%rsp),%al
0x4010fa <phase_3+325> je 0x401101 <phase_3+332>
0x4010fc <phase_3+327> callq 0x40172e <explode_bomb>
```

首先给 eax 赋值，判断第三个参数是否为 0x117，然后再次进行跳转。比较第二个参数的 ascii 码是否为 0x67。最后结束。

一种可能的结果是: 2 g 279

## 第四阶段

```
0000000000401156 <phase_4>:
401156: 48 83 ec 18          sub $0x18,%rsp
40115a: 64 48 8b 04 25 28 00 mov %fs:0x28,%rax
401161: 00 00
401163: 48 89 44 24 08       mov %rax,0x8(%rsp)
401168: 31 c0                xor %eax,%eax
40116a: 48 89 e1             mov %rsp,%rcx
40116d: 48 8d 54 24 04       lea 0x4(%rsp),%rdx
401172: be 4d 2a 40 00       mov $0x402a4d,%esi
401177: e8 c4 fa ff ff      callq 400c40 <__isoc99_sscanf@plt>
40117c: 83 f8 02             cmp $0x2,%eax
40117f: 75 0b               jne 40118c <phase_4+0x36>
401181: 8b 04 24             mov (%rsp),%eax
401184: 83 e8 02             sub $0x2,%eax
401187: 83 f8 02             cmp $0x2,%eax
40118a: 76 05               jbe 401191 <phase_4+0x3b>
40118c: e8 9d 05 00 00      callq 40172e <explode_bomb>
401191: 8b 34 24             mov (%rsp),%esi
401194: bf 09 00 00 00       mov $0x9,%edi
401199: e8 7d ff ff ff      callq 40111b <func4>
40119e: 3b 44 24 04          cmp 0x4(%rsp),%eax
4011a2: 74 05               je 4011a9 <phase_4+0x53>
4011a4: e8 85 05 00 00      callq 40172e <explode_bomb>
4011a9: 48 8b 44 24 08       mov 0x8(%rsp),%rax
4011ae: 64 48 33 04 25 28 00 xor %fs:0x28,%rax
4011b5: 00 00
4011b7: 74 05               je 4011be <phase_4+0x68>
4011b9: e8 d2 f9 ff ff      callq 400b90 <__stack_chk_fail@plt>
4011be: 48 83 c4 18          add $0x18,%rsp
4011c2: c3                  retq
```

首先看输入部分：

调用函数 `sscanf(input_strings,%esi,%rsp+0x4,%rsp)`

读入的参数是两个 int

```
(gdb) x/s 0x402a4d
0x402a4d: "%d %d"
```



输入处理结束后，判断读入参数是否多于两个，然后判断第二个参数是否小于等于 4（属于 0~4）。

```
0x40117c <phase_4+38>    cmp     $0x2,%eax
0x40117f <phase_4+41>    jne     0x40118c <phase_4+54>
0x401181 <phase_4+43>    mov     (%rsp),%eax
0x401184 <phase_4+46>    sub     $0x2,%eax
0x401187 <phase_4+49>    cmp     $0x2,%eax
0x40118a <phase_4+52>    jbe     0x401191 <phase_4+59>
0x40118c <phase_4+54>    callq   0x40172e <explode_bomb>
```

接着调用函数 func4(第二个参数,9)。判断结果是否等于第一个参数。

```
0x401191 <phase_4+59>    mov     (%rsp),%esi
0x401194 <phase_4+62>    mov     $0x9,%edi
0x401199 <phase_4+67>    callq   0x40111b <func4>
0x40119e <phase_4+72>    cmp     0x4(%rsp),%eax
0x4011a2 <phase_4+76>    je      0x4011a9 <phase_4+83>
0x4011a4 <phase_4+78>    callq   0x40172e <explode_bomb>
```

下面来看 func4 函数。这是一个递归函数，因此用 c 语言对它进行翻译能够更好的得到结果：

```
000000000040111b <func4>:
40111b: 85 ff          test    %edi,%edi
40111d: 7e 2b          jle     40114a <func4+0x2f>
40111f: 89 f0          mov     %esi,%eax
401121: 83 ff 01       cmp     $0x1,%edi
401124: 74 2e          je      401154 <func4+0x39>
401126: 41 54          push    %r12
401128: 55            push    %rbp
401129: 53            push    %rbx
40112a: 89 f5          mov     %esi,%ebp
40112c: 89 fb          mov     %edi,%ebx
40112e: 8d 7f ff       lea     -0x1(%rdi),%edi
401131: e8 e5 ff ff ff callq   40111b <func4>
401136: 44 8d 64 05 00 lea     0x0(%rbp,%rax,1),%r12d
40113b: 8d 7b fe       lea     -0x2(%rbx),%edi
40113e: 89 ee          mov     %ebp,%esi
401140: e8 d6 ff ff ff callq   40111b <func4>
401145: 44 01 e0       add     %r12d,%eax
401148: eb 06          jmp     401150 <func4+0x35>
40114a: b8 00 00 00 00 mov     $0x0,%eax
40114f: c3            retq
401150: 5b            pop     %rbx
401151: 5d            pop     %rbp
401152: 41 5c          pop     %r12
401154: f3 c3          repz retq
```

```
1. int func4(int x,int y){
2.     if (y<=0){
3.         return 0;
4.     }else if (y==1){
5.         return x;
6.     }else{
7.         return x+func4(x,y-1)+func4(x,y-2);
8.     }
9. }
```

需要注意的是第一个判断语句。

然后可以列出递归运算表

fun(x,0)	0	fun(x,5)	12x
fun(x,1)	x	fun(x,6)	20x
fun(x,2)	2x	fun(x,7)	33x
fun(x,3)	4x	fun(x,8)	54x
fun(x,4)	7x	fun(x,9)	88x

因此结果就为 88x。

可能的结果为 176 2。

当然，因为是在 gdb 上运行，也可以使用 finish 命令，直接获得运算的结果。

## 第五阶段

```

00000000004011c3 <phase_5>:
 4011c3: 53                push    %rbx
 4011c4: 48 89 fb          mov     %rdi,%rbx
 4011c7: e8 70 02 00 00    callq  40143c <string_length>
 4011cc: 83 f8 06          cmp     $0x6,%eax
 4011cf: 74 05             je      4011d6 <phase_5+0x13>
 4011d1: e8 58 05 00 00    callq  40172e <explode_bomb>
 4011d6: 48 89 d8          mov     %rbx,%rax
 4011d9: 48 8d 7b 06       lea     0x6(%rbx),%rdi
 4011dd: b9 00 00 00 00    mov     $0x0,%ecx
 4011e2: 0f b6 10          movzbl  (%rax),%edx
 4011e5: 83 e2 0f          and     $0xf,%edx
 4011e8: 03 0c 95 c0 27 40 00 add     0x4027c0(,%rdx,4),%ecx
 4011ef: 48 83 c0 01       add     $0x1,%rax
 4011f3: 48 39 f8          cmp     %rdi,%rax
 4011f6: 75 ea            jne     4011e2 <phase_5+0x1f>
 4011f8: 83 f9 30          cmp     $0x30,%ecx
 4011fb: 74 05             je      401202 <phase_5+0x3f>
 4011fd: e8 2c 05 00 00    callq  40172e <explode_bomb>
 401202: 5b                pop     %rbx
 401203: c3                retq

```

输入部分要求输入 6 个字符。

再看运行部分

```

0x4011d6 <phase_5+19> mov     %rbx,%rax
0x4011d9 <phase_5+22> lea     0x6(%rbx),%rdi
0x4011dd <phase_5+26> mov     $0x0,%ecx
0x4011e2 <phase_5+31> movzbl  (%rax),%edx
0x4011e5 <phase_5+34> and     $0xf,%edx
0x4011e8 <phase_5+37> add     0x4027c0(,%rdx,4),%ecx
0x4011ef <phase_5+44> add     $0x1,%rax
0x4011f3 <phase_5+48> cmp     %rdi,%rax
0x4011f6 <phase_5+51> jne     0x4011e2 <phase_5+31>
0x4011f8 <phase_5+53> cmp     $0x30,%ecx
0x4011fb <phase_5+56> je      0x401202 <phase_5+63>
0x4011fd <phase_5+58> callq   0x40172e <explode_bomb>

```

首先初始化：%rax 指向第一个元素，%rdi 指向最后 1 个元素。循环计数 ecx 等于 0。

然后开始循环。

把当前指向的字母的 ascii 码提取最低半字节（0xF 掩码）保存在 %edx 中。然后把它作为索引，在 0x4027c0 处取值。可以猜出 0x4027c0 是一个 16 个元素的 int 数组。下面看一下 0x4027c0 下的内容：

1. x/16xw 0x4027c0

```
(gdb) x/16xw 0x4027c0
0x4027c0 <array.3600>: 0x00000002      0x0000000a      0x00000006      0x00000001
0x4027d0 <array.3600+16>: 0x0000000c      0x00000010      0x00000009      0x00000003
0x4027e0 <array.3600+32>: 0x00000004      0x00000007      0x0000000e      0x00000005
0x4027f0 <array.3600+48>: 0x0000000b      0x00000008      0x0000000f      0x0000000d
```

16 个元素的数组存的数恰好也是 0 到 15。

最后，把六个字母索引对应数组元素加在一起，结果应该是 0x30(48)。48=5+6+7+8+9+13（一种可能的结果）。因此，需要选择 0x2,0x6,0x9,0xb,0xd,0xf。这里只能输入一个字符，并且取的是它的低位。参考 ascii 码表，可以考虑用 2(0x32),6(0x36),9(0x39),K(0x4B),M(0x4D),O(0x4F)来作为答案。

一个可能的答案是：269KMO。