一、 实验目标与要求：

1. 理解程序（控制语句、函数、返回值、堆栈结构）是如何运行的
2. 掌握 GDB 调试工具和 objdump 反汇编工具

二、实验环境：

1. 计算机（Intel CPU）
2. Linux64 位操作系统（Ubuntu 17）
3. GDB 调试工具
4. objdump 反汇编工具

三、实验方法与步骤：

本实验设计为一个黑客拆解二进制炸弹的游戏。我们仅给黑客（同学）提供一个二进制可执行文件 bomb_64 和主函数所在的源程序 bomb_64.c，不提供每个关卡的源代码。程序运行中有 6 个关卡（6 个 phase），每个关卡需要用户输入正确的字符串或数字才能通关，否则会引爆炸弹（打印出一条错误信息，并导致评分下降）!

要求同学运用 **GDB 调试工具和 objdump 反汇编工具**，通过分析汇编代码，找到在每个 phase 程序段中，引导程序跳转到“explode_bomb”程序段的地方，并分析其成功跳转的条件，以此为突破口寻找应该在命令行输入何种字符串来通关。

本实验需解决 Phase_1(**15 分**)、Phase_2(**15 分**)、Phase_3(**15 分**)、Phase_4(**15 分**)、Phase_5(**15 分**)、Phase_6(**10 分**)。通过**截图+文字**的形式把实验过程写在实验报告上，最后并撰写**实验结论与心得(15 分)**。

```
dongyunhao_2019284073@ubuntu:~/expirement3$ ./bomb_64
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Science isn't about why, it's about why not?
Phase 1 defused. How about the next one?
1 1 1 1 1 1
That's number 2.  Keep going!
1 926
Halfway there!
9
So you got that one.  Try this one.
7 93
Congratulations! You've (mostly) defused the bomb!
Hit Control-C to escape phase 6 (for free!), but if you want to
try phase 6 for extra credit, you can continue.  Just beware!
673
Congratulations! You've defused the bomb! Again!
```

四、实验过程及内容：

1、进行反汇编：

首先需要对二进制文件进行反汇编。在*terminal*中输入如下代码：

$$objdump\ -d\ bomb\_64\ >\ 1.txt$$

将事先编译好的二进制文件进行反汇编，并将汇编代码重定向到 1.txt 文件中。

```
dongyunhao_2019284073@ubuntu:~/expirement3$ objdump -d bomb_64 > 1.txt
dongyunhao_2019284073@ubuntu:~/expirement3$
```

此时只需对反汇编出来的代码进行分析，并完成六个关卡即可。

2、对六个关卡进行一一破解：

（1）phase1:（string，函数调用，栈）

①获取汇编代码：

```
339 0000000000400e70 <phase_1>:
340   400e70:       48 83 ec 08             sub    $0x8,%rsp
341   400e74:       be f8 1a 40 00          mov    $0x401af8,%esi
342   400e79:       e8 bf 03 00 00          callq  40123d <strings_not_equal>
343   400e7e:       85 c0                   test   %eax,%eax
344   400e80:       74 05                   je     400e87 <phase_1+0x17>
345   400e82:       e8 b6 07 00 00          callq  40163d <explode_bomb>
346   400e87:       48 83 c4 08             add    $0x8,%rsp
347   400e8b:       c3                      retq
```

②分析代码逻辑：

```
01. 0000000000400e70 <phase_1>:
02.   400e70: 48 83 ec 08         sub    $0x8,%rsp            // 开栈
03.   400e74: be f8 1a 40 00      mov    $0x401af8,%esi       // 函数的第二个参数是$0x401af8，猜测是常量字符串
04.   400e79: e8 bf 03 00 00      callq  40123d <strings_not_equal>
05.   400e7e: 85 c0               test   %eax,%eax            // 返回值等于0，即两个字符串相等，则结束否则bomb
06.   400e80: 74 05               je     400e87 <phase_1+0x17>
07.   400e82: e8 b6 07 00 00      callq  40163d <explode_bomb>
08.   400e87: 48 83 c4 08         add    $0x8,%rsp            // 退栈
09.   400e8b: c3
```

- 首先，第 340 行为函数调用进行开栈。第 341 行将 0x401af8 地址中的内容复制到%esi 寄存器。第 342 行调用函数，判断字符串是否相等，若返回值为零，则跳转到 400e87 地址处；若不为零，则跳转到 explode_bomb 函数。

- 因此可以此处即为，判断输入是否与 0x401af8 中的字符串相等，相等则成功，不相等则炸弹爆炸，因此只需找到即可。

- 接下来，使用 gdb 进行获取 0x401af8 地址中的字符串。在 gdb 中输入如下代码：

$$p(char\ *)\ 0x401af8$$

```
(gdb) p (char *) 0x401af8
$2 = 0x401af8 "Science isn't about why, it's about why not?"
```

- 因此，只要输入*Science isn't about why, it's about why not?*即可

③进行测试：

输入*Science isn't about why, it's about why not?*

```
(gdb) r
Starting program: /home/dongyunhao_2019284073/expirement3/bomb_64
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Science isn't about why, it's about why not?
Phase 1 defused. How about the next one?
```

答案正确！

（2）phase2:（循环语句，数组）

①获取汇编代码：

```
0000000000400e8c <phase_2>:
  400e8c:       48 89 5c 24 e0          mov    %rbx,-0x20(%rsp)
  400e91:       48 89 6c 24 e8          mov    %rbp,-0x18(%rsp)
  400e96:       4c 89 64 24 f0          mov    %r12,-0x10(%rsp)
  400e9b:       4c 89 6c 24 f8          mov    %r13,-0x8(%rsp)
  400ea0:       48 83 ec 48             sub    $0x48,%rsp
  400ea4:       48 89 e6                mov    %rsp,%rsi
  400ea7:       e8 97 08 00 00          callq  401743 <read_six_numbers>
  400eac:       48 89 e5                mov    %rsp,%rbp
  400eaf:       4c 8d 6c 24 0c          lea    0xc(%rsp),%r13
  400eb4:       41 bc 00 00 00 00       mov    $0x0,%r12d
  400eba:       48 89 eb                mov    %rbp,%rbx
  400ebd:       8b 45 0c                mov    0xc(%rbp),%eax
  400ec0:       39 45 00                cmp    %eax,0x0(%rbp)
  400ec3:       74 05                   je     400eca <phase_2+0x3e>
  400ec5:       e8 73 07 00 00          callq  40163d <explode_bomb>
  400eca:       44 03 23                add    (%rbx),%r12d
  400ecd:       48 83 c5 04             add    $0x4,%rbp
  400ed1:       4c 39 ed                cmp    %r13,%rbp
  400ed4:       75 e4                   jne    400eba <phase_2+0x2e>
  400ed6:       45 85 e4                test   %r12d,%r12d
  400ed9:       75 05                   jne    400ee0 <phase_2+0x54>
  400edb:       e8 5d 07 00 00          callq  40163d <explode_bomb>
  400ee0:       48 8b 5c 24 28          mov    0x28(%rsp),%rbx
  400ee5:       48 8b 6c 24 30          mov    0x30(%rsp),%rbp
  400eea:       4c 8b 64 24 38          mov    0x38(%rsp),%r12
  400eef:       4c 8b 6c 24 40          mov    0x40(%rsp),%r13
  400ef4:       48 83 c4 48             add    $0x48,%rsp
  400ef8:       c3                      retq
```

```
0000000000401743 <read_six_numbers>:
  401743:       48 83 ec 18             sub    $0x18,%rsp
  401747:       48 89 f2                mov    %rsi,%rdx
  40174a:       48 8d 4e 04             lea    0x4(%rsi),%rcx
  40174e:       48 8d 46 14             lea    0x14(%rsi),%rax
  401752:       48 89 44 24 08          mov    %rax,0x8(%rsp)
  401757:       48 8d 46 10             lea    0x10(%rsi),%rax
  40175b:       48 89 04 24             mov    %rax,(%rsp)
  40175f:       4c 8d 4e 0c             lea    0xc(%rsi),%r9
  401763:       4c 8d 46 08             lea    0x8(%rsi),%r8
  401767:       be b2 1e 40 00          mov    $0x401eb2,%esi
  40176c:       b8 00 00 00 00          mov    $0x0,%eax
  401771:       e8 3a f3 ff ff          callq  400ab0 <__isoc99_sscanf@plt>
  401776:       83 f8 05                cmp    $0x5,%eax
  401779:       7f 05                   jg     401780 <read_six_numbers+0x3d>
  40177b:       e8 bd fe ff ff          callq  40163d <explode_bomb>
  401780:       48 83 c4 18             add    $0x18,%rsp
  401784:       c3                      retq
```

②分析代码逻辑：

```
0000000000400e8c <phase_2>:
  400e8c: 48 89 5c 24 e0        mov    %rbx,-0x20(%rsp)       // 保存父函数数据，压栈
  400e91: 48 89 6c 24 e8        mov    %rbp,-0x18(%rsp)
  400e96: 4c 89 64 24 f0        mov    %r12,-0x10(%rsp)
  400e9b: 4c 89 6c 24 f8        mov    %r13,-0x8(%rsp)
  400ea0: 48 83 ec 48           sub    $0x48,%rsp             // 开栈
  400ea4: 48 89 e6              mov    %rsp,%rsi              // 栈顶指针给到第二个参数
  400ea7: e8 97 08 00 00        callq  401743 <read_six_numbers> // 调用函数读数据
  400eac: 48 89 e5              mov    %rsp,%rbp              // 保存栈指针到rbp
  400eaf: 4c 8d 6c 24 0c        lea    0xc(%rsp),%r13         // 栈指针+12保存到r13
  400eb4: 41 bc 00 00 00 00     mov    $0x0,%r12d             // r12低32位清零
  // ------------------------------------------------- 循环 ------------------------------------------ //
  400eba: 48 89 eb              mov    %rbp,%rbx              // rbp的值保存到rbx
  400ebd: 8b 45 0c              mov    0xc(%rbp),%eax         // M[rbp+12]保存到eax
  400ec0: 39 45 00              cmp    %eax,0x0(%rbp)         // 看 M[rbp] 和 M[rbp+12] 是否相等
  400ec3: 74 05                 je     400eca <phase_2+0x3e>  // 如果M[rbp]和M[rbp+12]不等则bomb
  400ec5: e8 73 07 00 00        callq  40163d <explode_bomb>
  400eca: 44 03 23              add    (%rbx),%r12d           // r12 += M[rbx]
  400ecd: 48 83 c5 04           add    $0x4,%rbp              // rbp += 4, rbp为迭代指针
  400ed1: 4c 39 ed              cmp    %r13,%rbp              // r13和rbp如果不等则跳转
  400ed4: 75 e4                 jne    400eba <phase_2+0x2e>
  // ------------------------------------------------- 循环结束 ------------------------------------- //
  400ed6: 45 85 e4              test   %r12d,%r12d            // r12d是累加值，累加值为0则bomb
  400ed9: 75 05                 jne    400ee0 <phase_2+0x54>
  400edb: e8 5d 07 00 00        callq  40163d <explode_bomb>
  400ee0: 48 8b 5c 24 28        mov    0x28(%rsp),%rbx        // 退栈
  400ee5: 48 8b 6c 24 30        mov    0x30(%rsp),%rbp
  400eea: 4c 8b 64 24 38        mov    0x38(%rsp),%r12
  400eef: 4c 8b 6c 24 40        mov    0x40(%rsp),%r13
  400ef4: 48 83 c4 48           add    $0x48,%rsp
  400ef8: c3                    retq
```

- 从代码的第 356 行，可以看到该函数首先调用了函数读入了 6 个数字，因此该关卡的答案由 6 个数组构成。并且可以看到，该读入 6 个数字的函数存放在内存地址为 401743 的地方，可以进入该地址查看*read_six_numbers*函数。
- 在主调函数*phase_2*中，可以发现，读入的 6 个数字将被存放在从%rsp 指向的地址开始向上的位置。

```
361   400ebd:       8b 45 0c                mov    0xc(%rbp),%eax
362   400ec0:       39 45 00                cmp    %eax,0x0(%rbp)
363   400ec3:       74 05                   je     400eca <phase_2+0x3e>
364   400ec5:       e8 73 07 00 00          callq  40163d <explode_bomb>
```

- 如上的代码中用 mov 将 0xc(%rbp)存入%eax 中，并通过 cmp 比较两个值，如果两个值相等得到结果 0，跳转到 0x400eca，否则将执行<explode_bomb>函数引爆炸弹。

```
365   400eca:       44 03 23                add    (%rbx),%r12d
366   400ecd:       48 83 c5 04             add    $0x4,%rbp
367   400ed1:       4c 39 ed                cmp    %r13,%rbp
368   400ed4:       75 e4                   jne    400eba <phase_2+0x2e>
369   400ed6:       45 85 e4                test   %r12d,%r12d
370   400ed9:       75 05                   jne    400ee0 <phase_2+0x54>
371   400edb:       e8 5d 07 00 00          callq  40163d <explode_bomb>
```

- 然后使用 rbp 作为迭代指针，rbp+12 为迭代终点，一共迭代三次，而每次我们都将 M[rbp] 和 M[rbp+12] 处的数据比对，如果不等则 bomb。而对于 int 型，刚好是 4 字节，三个即为 12 字节。因此，这段函数正在判断读入的 6 个数字中前三个数字是否与后三个数字对应相等。

```
369   400ed6:       45 85 e4                test   %r12d,%r12d
370   400ed9:       75 05                   jne    400ee0 <phase_2+0x54>
371   400edb:       e8 5d 07 00 00          callq  40163d <explode_bomb>
```

- r12 寄存器在运行前被之前清零，每次运行又 += M[rbx]，即在计算累加和。又根据后文 test $r12d $r12d，可得，输入的 6 个数要满足累加和不等于 0，否则将引爆炸弹。

- 综上，本题不引爆炸弹安全拆弹的条件是输入 6 个累加和不为零的数字，并满足前三个数字与后三个数字对应相等。

③进行测试：

根据上面的分析，可知1 1 1 1 1 1为一种合法的情况，进行输出并测试。

```
1 1 1 1 1 1
That's number 2.  Keep going!
```

答案正确！

（3）phase3:（switch 语句）

①获取汇编代码：

```
379 0000000000400ef9 <phase_3>:
380   400ef9:       48 83 ec 18             sub    $0x18,%rsp
381   400efd:       48 8d 4c 24 08          lea    0x8(%rsp),%rcx
382   400f02:       48 8d 54 24 0c          lea    0xc(%rsp),%rdx
383   400f07:       be be 1e 40 00          mov    $0x401ebe,%esi
384   400f0c:       b8 00 00 00 00          mov    $0x0,%eax
385   400f11:       e8 9a fb ff ff          callq  400ab0 <__isoc99_sscanf@plt>
386   400f16:       83 f8 01                cmp    $0x1,%eax
387   400f19:       7f 05                   jg     400f20 <phase_3+0x27>
388   400f1b:       e8 1d 07 00 00          callq  40163d <explode_bomb>
389   400f20:       83 7c 24 0c 07          cmpl   $0x7,0xc(%rsp)
390   400f25:       77 3c                   ja     400f63 <phase_3+0x6a>
391   400f27:       8b 44 24 0c             mov    0xc(%rsp),%eax
392   400f2b:       ff 24 c5 60 1b 40 00    jmpq   *0x401b60(,%rax,8)
393   400f32:       b8 17 02 00 00          mov    $0x217,%eax
394   400f37:       eb 3b                   jmp    400f74 <phase_3+0x7b>
395   400f39:       b8 d6 00 00 00          mov    $0xd6,%eax
396   400f3e:       eb 34                   jmp    400f74 <phase_3+0x7b>
397   400f40:       b8 53 01 00 00          mov    $0x153,%eax
398   400f45:       eb 2d                   jmp    400f74 <phase_3+0x7b>
399   400f47:       b8 77 00 00 00          mov    $0x77,%eax
400   400f4c:       eb 26                   jmp    400f74 <phase_3+0x7b>
401   400f4e:       b8 60 01 00 00          mov    $0x160,%eax
402   400f53:       eb 1f                   jmp    400f74 <phase_3+0x7b>
403   400f55:       b8 97 03 00 00          mov    $0x397,%eax
404   400f5a:       eb 18                   jmp    400f74 <phase_3+0x7b>
405   400f5c:       b8 9c 01 00 00          mov    $0x19c,%eax
406   400f61:       eb 11                   jmp    400f74 <phase_3+0x7b>
407   400f63:       e8 d5 06 00 00          callq  40163d <explode_bomb>
408   400f68:       b8 00 00 00 00          mov    $0x0,%eax
409   400f6d:       eb 05                   jmp    400f74 <phase_3+0x7b>
410   400f6f:       b8 9e 03 00 00          mov    $0x39e,%eax
411   400f74:       3b 44 24 08             cmp    0x8(%rsp),%eax
412   400f78:       74 05                   je     400f7f <phase_3+0x86>
413   400f7a:       e8 be 06 00 00          callq  40163d <explode_bomb>
414   400f7f:       48 83 c4 18             add    $0x18,%rsp
415   400f83:       c3                      retq
```

②分析代码逻辑：

```
0000000000400ef9 <phase_3>:
  400ef9: 48 83 ec 18           sub    $0x18,%rsp           // 开栈
  400efd: 48 8d 4c 24 08        lea    0x8(%rsp),%rcx       // rcx = rsp+8  第四个参数
  400f02: 48 8d 54 24 0c        lea    0xc(%rsp),%rdx       // rdx = rsp+12 第三个参数
  400f07: be be 1e 40 00        mov    $0x401ebe,%esi       // 第二个参数 gdb查找这个字符串的值是 "%d %d"
  400f0c: b8 00 00 00 00        mov    $0x0,%eax            // 返回值低32字节置0
  400f11: e8 9a fb ff ff        callq  400ab0 <__isoc99_sscanf@plt> // 读数据
  400f16: 83 f8 01              cmp    $0x1,%eax            // scanf返回不为1则bomb
  400f19: 7f 05                 jg     400f20 <phase_3+0x27>
  400f1b: e8 1d 07 00 00        callq  40163d <explode_bomb>
  400f20: 83 7c 24 0c 07        cmpl   $0x7,0xc(%rsp)       // 7 < M[rsp+12]（第一个%d） 则跳转到bomb
  400f25: 77 3c                 ja     400f63 <phase_3+0x6a>
  400f27: 8b 44 24 0c           mov    0xc(%rsp),%eax       // M[rsp+12]（第一个%d）存到eax
  400f2b: ff 24 c5 60 1b 40 00  jmpq   *0x401b60(,%rax,8)   // 跳转到 M[0x401b60+ 8*rax] 地址
  400f32: b8 17 02 00 00        mov    $0x217,%eax          // if rax=0则到这里，eax=535
  400f37: eb 3b                 jmp    400f74 <phase_3+0x7b>
  400f39: b8 d6 00 00 00        mov    $0xd6,%eax           // if rax=2则到这里，eax=214
  400f3e: eb 34                 jmp    400f74 <phase_3+0x7b>
  400f40: b8 53 01 00 00        mov    $0x153,%eax          // if rax=3则到这里，eax=339
  400f45: eb 2d                 jmp    400f74 <phase_3+0x7b>
  400f47: b8 77 00 00 00        mov    $0x77,%eax           // if rax=4则到这里，eax=119
  400f4c: eb 26                 jmp    400f74 <phase_3+0x7b>
  400f4e: b8 60 01 00 00        mov    $0x160,%eax          // if rax=5则到这里，eax=352
  400f53: eb 1f                 jmp    400f74 <phase_3+0x7b>
  400f55: b8 97 03 00 00        mov    $0x397,%eax          // if rax=6则到这里，eax=919
  400f5a: eb 18                 jmp    400f74 <phase_3+0x7b>
  400f5c: b8 9c 01 00 00        mov    $0x19c,%eax          // if rax=7则到这里，eax=412
  400f61: eb 11                 jmp    400f74 <phase_3+0x7b>
  400f63: e8 d5 06 00 00        callq  40163d <explode_bomb>// bomb
  400f68: b8 00 00 00 00        mov    $0x0,%eax
  400f6d: eb 05                 jmp    400f74 <phase_3+0x7b>
  400f6f: b8 9e 03 00 00        mov    $0x39e,%eax          // if rax=1则到这里，eax=926
  400f74: 3b 44 24 08           cmp    0x8(%rsp),%eax       // switch转到这里，如果第二个%d不等于eax，则bomb
  400f78: 74 05                 je     400f7f <phase_3+0x86>
  400f7a: e8 be 06 00 00        callq  40163d <explode_bomb>
  400f7f: 48 83 c4 18           add    $0x18,%rsp           // 退栈
  400f83: c3                    retq
```

- 首先，在 scanf 调用前，传入第三第四个参数了，并存在 rsp+8 和 rsp+12 位置，从间距来看应该是 int，然后又向 esi 传入常量，推测是 scanf 格式确定的字符串。因此可以利用 gdb 查看 0x401ebe 内存下的值。

```
(gdb) p(char*)0x401ebe
$4 = 0x401ebe "%d %d"
```

  这说明，本任务的输入为两个整数。

- 第 11 行中判断输入的第一个整数是否比 7 小，否则直接 bomb。

- 第 14 行进行跳转，目的地址是 M[0x401b60 + rax*8]，其中 rax 是输入的第一个整数。当跳转到对应的地址后，将 eax 赋对应值，并跳转到第 33 行判断是否与第二个输入整数相等，如果不等则 bomb。因此，程序大致为一 switch 型程序。现在我们只需找出 switch 对应的 8 个值即可。

- 利用 gdb 分别查看对应第一个输入为 0~7 时跳转的目的地址值

```
(gdb) p/x *(int*) (0x401b60 + 8 * 0)
$8 = 0x400f32
(gdb) p/x *(int*) (0x401b60 + 8 * 1)
$9 = 0x400f6f
(gdb) p/x *(int*) (0x401b60 + 8 * 2)
$10 = 0x400f39
(gdb) p/x *(int*) (0x401b60 + 8 * 3)
$11 = 0x400f40
(gdb) p/x *(int*) (0x401b60 + 8 * 4)
$12 = 0x400f47
(gdb) p/x *(int*) (0x401b60 + 8 * 5)
$13 = 0x400f4e
(gdb) p/x *(int*) (0x401b60 + 8 * 6)
$14 = 0x400f55
(gdb) p/x *(int*) (0x401b60 + 8 * 7)
$15 = 0x400f5c
```

- 因此，本题即为一个*switch*型函数，只要输入满足*switch*对应的 8 组数值即可。因此，合法输入如下：

$$\begin{cases} 0\ 535 \\ 1\ 926 \\ 2\ 214 \\ 3\ 339 \\ 4\ 119 \\ 5\ 352 \\ 6\ 919 \\ 7\ 412 \end{cases}$$

③进行测试：
根据分析，测试输入 0 535

```
0 535
Halfway there!
```

答案正确！

（4）phase4:（递归）
①获取汇编代码：

```
0000000000400fc1 <phase_4>:
  400fc1:       48 83 ec 18             sub    $0x18,%rsp
  400fc5:       48 8d 54 24 0c          lea    0xc(%rsp),%rdx
  400fca:       be c1 1e 40 00          mov    $0x401ec1,%esi
  400fcf:       b8 00 00 00 00          mov    $0x0,%eax
  400fd4:       e8 d7 fa ff ff          callq  400ab0 <__isoc99_sscanf@plt>
  400fd9:       83 f8 01                cmp    $0x1,%eax
  400fdc:       75 07                   jne    400fe5 <phase_4+0x24>
  400fde:       83 7c 24 0c 00          cmpl   $0x0,0xc(%rsp)
  400fe3:       7f 05                   jg     400fea <phase_4+0x29>
  400fe5:       e8 53 06 00 00          callq  40163d <explode_bomb>
  400fea:       8b 7c 24 0c             mov    0xc(%rsp),%edi
  400fee:       e8 91 ff ff ff          callq  400f84 <func4>
  400ff3:       83 f8 37                cmp    $0x37,%eax
  400ff6:       74 05                   je     400ffd <phase_4+0x3c>
  400ff8:       e8 40 06 00 00          callq  40163d <explode_bomb>
  400ffd:       48 83 c4 18             add    $0x18,%rsp
  401001:       c3                      retq
```

```
0000000000400f84 <func4>:
  400f84:       48 89 5c 24 f0          mov    %rbx,-0x10(%rsp)
  400f89:       48 89 6c 24 f8          mov    %rbp,-0x8(%rsp)
  400f8e:       48 83 ec 18             sub    $0x18,%rsp
  400f92:       89 fb                   mov    %edi,%ebx
  400f94:       b8 01 00 00 00          mov    $0x1,%eax
  400f99:       83 ff 01                cmp    $0x1,%edi
  400f9c:       7e 14                   jle    400fb2 <func4+0x2e>
  400f9e:       8d 7b ff                lea    -0x1(%rbx),%edi
  400fa1:       e8 de ff ff ff          callq  400f84 <func4>
  400fa6:       89 c5                   mov    %eax,%ebp
  400fa8:       8d 7b fe                lea    -0x2(%rbx),%edi
  400fab:       e8 d4 ff ff ff          callq  400f84 <func4>
  400fb0:       01 e8                   add    %ebp,%eax
  400fb2:       48 8b 5c 24 08          mov    0x8(%rsp),%rbx
  400fb7:       48 8b 6c 24 10          mov    0x10(%rsp),%rbp
  400fbc:       48 83 c4 18             add    $0x18,%rsp
  400fc0:       c3                      retq
```

②分析代码逻辑：

```
0000000000400fc1 <phase_4>:
  400fc1: 48 83 ec 18           sub    $0x18,%rsp          // 开栈
  400fc5: 48 8d 54 24 0c        lea    0xc(%rsp),%rdx      // rdx=rsp+12，设置第三个参数为rsp+12，即存放输入
  400fca: be c1 1e 40 00        mov    $0x401ec1,%esi      // 根据上面几题的经验应该是scanf的格式字串，为%d
  400fcf: b8 00 00 00 00        mov    $0x0,%eax           // eax置零
  400fd4: e8 d7 fa ff ff        callq  400ab0 <__isoc99_sscanf@plt>
  400fd9: 83 f8 01              cmp    $0x1,%eax           // scanf返回值是否为1？不为1则bomb
  400fdc: 75 07                 jne    400fe5 <phase_4+0x24>
  400fde: 83 7c 24 0c 00        cmpl   $0x0,0xc(%rsp)      // 进一步比较输入的int是否大于0
  400fe3: 7f 05                 jg     400fea <phase_4+0x29>
  400fe5: e8 53 06 00 00        callq  40163d <explode_bomb>  // 输入的int为0也bomb
  400fea: 8b 7c 24 0c           mov    0xc(%rsp),%edi      // 如果输入int大于0则到这里，设置fun4第一个参数为输入的int
  400fee: e8 91 ff ff ff        callq  400f84 <func4>
  400ff3: 83 f8 37              cmp    $0x37,%eax          // 返回值是否为55？
  400ff6: 74 05                 je     400ffd <phase_4+0x3c>
  400ff8: e8 40 06 00 00        callq  40163d <explode_bomb>  // 返回值不为55则bomb
  400ffd: 48 83 c4 18           add    $0x18,%rsp          // 返回值为55则退栈
  401001: c3                    retq
```

```
0000000000400f84 <func4>:
  400f84: 48 89 5c 24 f0        mov    %rbx,-0x10(%rsp)
  400f89: 48 89 6c 24 f8        mov    %rbp,-0x8(%rsp)
  400f8e: 48 83 ec 18           sub    $0x18,%rsp      // 开栈
  400f92: 89 fb                 mov    %edi,%ebx
  400f94: b8 01 00 00 00        mov    $0x1,%eax       // 返回值置1
  400f99: 83 ff 01              cmp    $0x1,%edi       // 如果第一个参数为1，直接返回（此时返回值是1）
  400f9c: 7e 14                 jle    400fb2 <func4+0x2e>
  400f9e: 8d 7b ff              lea    -0x1(%rbx),%edi // 设置第一个参数 edi = rbx -1，再递归调用fun4
  400fa1: e8 de ff ff ff        callq  400f84 <func4>
  400fa6: 89 c5                 mov    %eax,%ebp       // 返回值保存到 ebp
  400fa8: 8d 7b fe              lea    -0x2(%rbx),%edi // 设置第一个参数 edi = rbx -2，再递归调用fun4
  400fab: e8 d4 ff ff ff        callq  400f84 <func4>
  400fb0: 01 e8                 add    %ebp,%eax       // ebp += 返回值
  400fb2: 48 8b 5c 24 08        mov    0x8(%rsp),%rbx  // 退栈
  400fb7: 48 8b 6c 24 10        mov    0x10(%rsp),%rbp
  400fbc: 48 83 c4 18           add    $0x18,%rsp
  400fc0: c3                    retq
```

- 同样首先观察参数，发现这次 scanf 只读了一个参数，根据前面的经验不难看出，0x401ec1 存储的是 scanf 的常字符串，使用 gdb 查看 0x401ec1 中的值

```
(gdb) p (char *) 0x401ec1
$17 = 0x401ec1 "%d"
```

  这说明，本题的输入仅有一个整数

- 主调函数的第 14 行判断fun_4的返回值是否为 55，如果不是则 bomb

- 接下来观察被调函数fun_4。首先是压栈保存数据，再将传入的参数与 1 对比，小于等于 1 则跳转到 0x400fb2，退出函数并返回。否则将传入的参数减一后再次调用 func4 函数，再将返回值存放在%ebp 里，然后将原始参数减二后再次递归调用 func4 函数，将返回值与上一次递归得到的返回值%ebp 相加存在%eax 中。结合数学分析，该函数是求斐波那契数列第n项的值。

- 因此，只需找出斐波那契数列第几项的值为 55 即可，通过数学计算，是第 9 项。因此答案为 9.

③进行测试：
通过上面的分析，输入 9 即可

```
9
So you got that one.  Try this one.
```

答案正确！

（5）phase5:（字串变换，ascii 转换，寻址）
①获取汇编代码：

```
0000000000401002 <phase_5>:
  401002:       48 83 ec 18             sub    $0x18,%rsp
  401006:       48 8d 4c 24 08          lea    0x8(%rsp),%rcx
  40100b:       48 8d 54 24 0c          lea    0xc(%rsp),%rdx
  401010:       be be 1e 40 00          mov    $0x401ebe,%esi
  401015:       b8 00 00 00 00          mov    $0x0,%eax
  40101a:       e8 91 fa ff ff          callq  400ab0 <__isoc99_sscanf@plt>
  40101f:       83 f8 01                cmp    $0x1,%eax
  401022:       7f 05                   jg     401029 <phase_5+0x27>
  401024:       e8 14 06 00 00          callq  40163d <explode_bomb>
  401029:       8b 44 24 0c             mov    0xc(%rsp),%eax
  40102d:       83 e0 0f                and    $0xf,%eax
  401030:       89 44 24 0c             mov    %eax,0xc(%rsp)
  401034:       83 f8 0f                cmp    $0xf,%eax
  401037:       74 2c                   je     401065 <phase_5+0x63>
  401039:       b9 00 00 00 00          mov    $0x0,%ecx
  40103e:       ba 00 00 00 00          mov    $0x0,%edx
  401043:       83 c2 01                add    $0x1,%edx
  401046:       48 98                   cltq
  401048:       8b 04 85 a0 1b 40 00    mov    0x401ba0(,%rax,4),%eax
  40104f:       01 c1                   add    %eax,%ecx
  401051:       83 f8 0f                cmp    $0xf,%eax
  401054:       75 ed                   jne    401043 <phase_5+0x41>
  401056:       89 44 24 0c             mov    %eax,0xc(%rsp)
  40105a:       83 fa 0c                cmp    $0xc,%edx
  40105d:       75 06                   jne    401065 <phase_5+0x63>
  40105f:       3b 4c 24 08             cmp    0x8(%rsp),%ecx
  401063:       74 05                   je     40106a <phase_5+0x68>
  401065:       e8 d3 05 00 00          callq  40163d <explode_bomb>
  40106a:       48 83 c4 18             add    $0x18,%rsp
  40106e:       c3                      retq
```

②分析代码逻辑：

```
0000000000401002 <phase_5>:
  401002: 48 83 ec 18           sub    $0x18,%rsp            // 开栈
  401006: 48 8d 4c 24 08        lea    0x8(%rsp),%rcx        // 第四个参数 += M[rsp+8]
  40100b: 48 8d 54 24 0c        lea    0xc(%rsp),%rdx        // 第三个参数 += M[rsp+12]
  401010: be be 1e 40 00        mov    $0x401ebe,%esi        // 第二个参数，gdb查看内容为"%d %d"
  401015: b8 00 00 00 00        mov    $0x0,%eax             // eax清零
  40101a: e8 91 fa ff ff        callq  400ab0 <__isoc99_sscanf@plt>
  40101f: 83 f8 01              cmp    $0x1,%eax             // 如果eax<=1那么bomb
  401022: 7f 05                 jg     401029 <phase_5+0x27>
  401024: e8 14 06 00 00        callq  40163d <explode_bomb>
  401029: 8b 44 24 0c           mov    0xc(%rsp),%eax        // eax = M[rsp+12]，即输入的第一个int
  40102d: 83 e0 0f              and    $0xf,%eax             // eax &= f，即保留低4位
  401030: 89 44 24 0c           mov    %eax,0xc(%rsp)        // M[rsp+12] = eax ，即保留eax的低4位
  401034: 83 f8 0f              cmp    $0xf,%eax             // 如果eax = 15那么bomb
  401037: 74 2c                 je     401065 <phase_5+0x63>
  401039: b9 00 00 00 00        mov    $0x0,%ecx             // ecx清零
  40103e: ba 00 00 00 00        mov    $0x0,%edx             // edx清零
  // ------------------------------------------ 循环 ------------------------------------------//
  401043: 83 c2 01              add    $0x1,%edx             // edx += 1 ，记录循环次数
  401046: 48 98                 cltq                         // 拓展字节
  401048: 8b 04 85 a0 1b 40 00  mov    0x401ba0(,%rax,4),%eax // eax = M[0x401ba0+rax*4] （eax rax是一个寄存器）
  40104f: 01 c1                 add    %eax,%ecx             // ecx += eax，记录累加值
  401051: 83 f8 0f              cmp    $0xf,%eax             // 如果eax不等于15则继续
  401054: 75 ed                 jne    401043 <phase_5+0x41>
  // ------------------------------------------ 循环结束 ------------------------------------------//
  401056: 89 44 24 0c           mov    %eax,0xc(%rsp)        // M[rsp+12] = eax，即=15
  40105a: 83 fa 0c              cmp    $0xc,%edx             // 如果edx（循环次数）不等于12则bomb
  40105d: 75 06                 jne    401065 <phase_5+0x63>
  40105f: 3b 4c 24 08           cmp    0x8(%rsp),%ecx        // M[rsp+8]即输入的第二个int，不等于ecx则bomb
  401063: 74 05                 je     40106a <phase_5+0x68>
  401065: e8 d3 05 00 00        callq  40163d <explode_bomb>
  40106a: 48 83 c4 18           add    $0x18,%rsp            // 退栈
  40106e: c3                    retq
```

- 首先观察 scanf 的输入，两个 lea 语句将 scanf 的第三第四个参数确定，采用与前面相同的方法，使用 gdb 对 scanf 的参数进行查看如下：

```
(gdb) p (char *)0x401ebe
$1 = 0x401ebe "%d %d"
```

即，本题输入为两个整数。

- 第 14 行对输入进行了截断，将第一个输出截断到 0~15 的，因此如果第一个输出被截断后等于 15，将直接爆炸。

- 通过阅读后面的语句，可以发现存在一个循环，edx 每次++，eax 每次指向内存中的一个地址，又因为 eax 和 rax 是同一个，则相当于 $k = next[k]$这种数组跳转语句，并且偏移的单位是 4 字节，刚好是一个 int，这更加确定了这里出现的是数组跳转语句。我们通过 gdb 查看这个数组。

- 使用 gdb 查看连续数组，结果如下：

```
(gdb) p *0x401ba0@16
$5 = {10, 2, 14, 7, 8, 12, 15, 11, 0, 4, 1, 13, 3, 9, 6, 5}
```

可绘制表格如下：

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
[i]	10	2	14	7	8	12	15	11	0	4	1	13	3	9	6	5

- 因此，继续对代码进行分析可知总跳转次数为 12。并且，第二个输入值必须等于每次跳转的目标的累加和，而且最后一次跳转的结果是 15。因此可以从 15 进行逆推 12 次并累加。

- 因此有$11 + 13 + 9 + 4 + 8 + 0 + 10 + 1 + 2 + 14 + 6 + 15 = 93$。因此最终答案为7 93

③进行测试：

输入7 93

```
7 93
Congratulations! You've (mostly) defused the bomb!
Hit Control-C to escape phase 6 (for free!), but if you want to
try phase 6 for extra credit, you can continue.  Just beware!
```

答案正确！我们现在只剩下最后一题需要解决了！

（6）phase6:（寻址）

①获取汇编代码：

```
00000000004010d9 <phase_6>:
  4010d9:       48 83 ec 08             sub    $0x8,%rsp
  4010dd:       ba 0a 00 00 00          mov    $0xa,%edx
  4010e2:       be 00 00 00 00          mov    $0x0,%esi
  4010e7:       e8 94 fa ff ff          callq  400b80 <strtol@plt>
  4010ec:       89 05 8e 16 20 00       mov    %eax,0x20168e(%rip)        # 602780 <node0>
  4010f2:       bf 80 27 60 00          mov    $0x602780,%edi
  4010f7:       e8 73 ff ff ff          callq  40106f <fun6>
  4010fc:       48 8b 40 08             mov    0x8(%rax),%rax
  401100:       48 8b 40 08             mov    0x8(%rax),%rax
  401104:       48 8b 40 08             mov    0x8(%rax),%rax
  401108:       8b 15 72 16 20 00       mov    0x201672(%rip),%edx        # 602780 <node0>
  40110e:       39 10                   cmp    %edx,(%rax)
  401110:       74 05                   je     401117 <phase_6+0x3e>
  401112:       e8 26 05 00 00          callq  40163d <explode_bomb>
  401117:       48 83 c4 08             add    $0x8,%rsp
  40111b:       c3                      retq
```

```
000000000040106f <fun6>:
  40106f:       4c 8b 47 08             mov    0x8(%rdi),%r8
  401073:       48 c7 47 08 00 00 00    movq   $0x0,0x8(%rdi)
  40107a:       00
  40107b:       48 89 f8                mov    %rdi,%rax
  40107e:       48 89 f9                mov    %rdi,%rcx
  401081:       4d 85 c0                test   %r8,%r8
  401084:       75 40                   jne    4010c6 <fun6+0x57>
  401086:       48 89 f8                mov    %rdi,%rax
  401089:       c3                      retq
  40108a:       48 89 d1                mov    %rdx,%rcx
  40108d:       48 8b 51 08             mov    0x8(%rcx),%rdx
  401091:       48 85 d2                test   %rdx,%rdx
  401094:       74 09                   je     40109f <fun6+0x30>
  401096:       39 32                   cmp    %esi,(%rdx)
  401098:       7f f0                   jg     40108a <fun6+0x1b>
  40109a:       48 89 cf                mov    %rcx,%rdi
  40109d:       eb 03                   jmp    4010a2 <fun6+0x33>
  40109f:       48 89 cf                mov    %rcx,%rdi
  4010a2:       48 39 d7                cmp    %rdx,%rdi
  4010a5:       74 06                   je     4010ad <fun6+0x3e>
  4010a7:       4c 89 47 08             mov    %r8,0x8(%rdi)
  4010ab:       eb 03                   jmp    4010b0 <fun6+0x41>
  4010ad:       4c 89 c0                mov    %r8,%rax
  4010b0:       49 8b 48 08             mov    0x8(%r8),%rcx
  4010b4:       49 89 50 08             mov    %rdx,0x8(%r8)
  4010b8:       48 85 c9                test   %rcx,%rcx
  4010bb:       74 1a                   je     4010d7 <fun6+0x68>
  4010bd:       49 89 c8                mov    %rcx,%r8
  4010c0:       48 89 c1                mov    %rax,%rcx
  4010c3:       48 89 c7                mov    %rax,%rdi
  4010c6:       48 89 ca                mov    %rcx,%rdx
  4010c9:       48 85 c9                test   %rcx,%rcx
  4010cc:       74 d4                   je     4010a2 <fun6+0x33>
  4010ce:       41 8b 30                mov    (%r8),%esi
  4010d1:       39 31                   cmp    %esi,(%rcx)
  4010d3:       7f b8                   jg     40108d <fun6+0x1e>
  4010d5:       eb cb                   jmp    4010a2 <fun6+0x33>
  4010d7:       f3 c3                   repz retq
```

②分析代码逻辑：

```
00000000004010d9 <phase_6>:
  4010d9: 48 83 ec 08           sub    $0x8,%rsp              // 开栈
  4010dd: ba 0a 00 00 00        mov    $0xa,%edx              // 第三个参数=10，strtol转换基数为10
  4010e2: be 00 00 00 00        mov    $0x0,%esi              // 第二个参数=0
  4010e7: e8 94 fa ff ff        callq  400b80 <strtol@plt>    // 将字符串根据基数转换为长整型数
  4010ec: 89 05 8e 16 20 00     mov    %eax,0x20168e(%rip)    # 602780 <node0> // M[20168e+rip] = eax（返回值）
  4010f2: bf 80 27 60 00        mov    $0x602780,%edi         // edi为第一个参数,这个位置应该是输入的字符串
  4010f7: e8 73 ff ff ff        callq  40106f <fun6>
  4010fc: 48 8b 40 08           mov    0x8(%rax),%rax         // rax = M[rax+8] rax:返回值
  401100: 48 8b 40 08           mov    0x8(%rax),%rax         // rax = M[rax+8]
  401104: 48 8b 40 08           mov    0x8(%rax),%rax         // rax = M[rax+8]
  401108: 8b 15 72 16 20 00     mov    0x201672(%rip),%edx    # 602780 <node0> // edx = M[201672+rip]
  40110e: 39 10                 cmp    %edx,(%rax)            // 如果edx 不等于 rax则bomb
  401110: 74 05                 je     401117 <phase_6+0x3e>
  401112: e8 26 05 00 00        callq  40163d <explode_bomb>
  401117: 48 83 c4 08           add    $0x8,%rsp              // 退栈
  40111b: c3                    retq
```

```
000000000040106f <fun6>:
  40106f: 4c 8b 47 08           mov    0x8(%rdi),%r8      // 第五个参数 = M[rdi（第一个参数）+8]
  401073: 48 c7 47 08 00 00 00  movq   $0x0,0x8(%rdi)     // M[rdi+8] = 0
  40107a: 00
  40107b: 48 89 f8              mov    %rdi,%rax          // rax = rdi
  40107e: 48 89 f9              mov    %rdi,%rcx          // rcx = rdi
  401081: 4d 85 c0              test   %r8,%r8            // 第五个参数不等于0则跳转
  401084: 75 40                 jne    4010c6 <fun6+0x57>
  401086: 48 89 f8              mov    %rdi,%rax          // rax = rdi（第一个参数）
  401089: c3                    retq                      // 退栈
  // ------------------------------------------------ 循环 ------------------------------------------------------------//
  40108a: 48 89 d1              mov    %rdx,%rcx          // rcx = rdx
  40108d: 48 8b 51 08           mov    0x8(%rcx),%rdx     // rdx = M[rcx+8]，即rdx=M[rdx+8]
  401091: 48 85 d2              test   %rdx,%rdx          // 如果rdx=0则跳转
  401094: 74 09                 je     40109f <fun6+0x30>
  401096: 39 32                 cmp    %esi,(%rdx)        // M[rdx]<esi则跳转，否则break
  401098: 7f f0                 jg     40108a <fun6+0x1b>
  // ------------------------------------------------ 循环结束 ---------------------------------------------------------//
  40109a: 48 89 cf              mov    %rcx,%rdi          // rdi = rcx
  40109d: eb 03                 jmp    4010a2 <fun6+0x33>
  40109f: 48 89 cf              mov    %rcx,%rdi          // 如果rdx=0则到这rdi = rcx
  // ------------------------------------------------ 循环 ------------------------------------------------------------//
  4010a2: 48 39 d7              cmp    %rdx,%rdi          // 如果rdi = rdx
  4010a5: 74 06                 je     4010ad <fun6+0x3e>
  4010a7: 4c 89 47 08           mov    %r8,0x8(%rdi)
  4010ab: eb 03                 jmp    4010b0 <fun6+0x41>
  4010ad: 4c 89 c0              mov    %r8,%rax           // 如果rdi=rdx则到这，rax=第五个参数
  4010b0: 49 8b 48 08           mov    0x8(%r8),%rcx      // rcx = M[r8+8]
  4010b4: 49 89 50 08           mov    %rdx,0x8(%r8)      // M[r8+8] = rdx
  4010b8: 48 85 c9              test   %rcx,%rcx          // 如果rcx=0则跳至结束
  4010bb: 74 1a                 je     4010d7 <fun6+0x68>
  4010bd: 49 89 c8              mov    %rcx,%r8           // r8 = cax
  4010c0: 48 89 c1              mov    %rax,%rcx          // rcx = rax
  4010c3: 48 89 c7              mov    %rax,%rdi          // rdi = rax
  4010c6: 48 89 ca              mov    %rcx,%rdx          // 如果第五个参数不等于0那么到这 rdx=rcx
  4010c9: 48 85 c9              test   %rcx,%rcx
  4010cc: 74 d4                 je     4010a2 <fun6+0x33>    // 如果rcx=0那么跳转到4010a2
  // ------------------------------------------------ 循环结束 ---------------------------------------------------------//
  4010ce: 41 8b 30              mov    (%r8),%esi         // esi = M[r8]
  4010d1: 39 31                 cmp    %esi,(%rcx)
  4010d3: 7f b8                 jg     40108d <fun6+0x1e> // 如果esi<M[rcx]则跳转到40108d
  4010d5: eb cb                 jmp    4010a2 <fun6+0x33>
  4010d7: f3 c3                 repz retq
```

- 通过观察代码，可以知道，如果程序执行 401112 语句将会引爆炸弹，所以需要满足跳转条件，即，当$edx == rax$时跳转。

- 上面的代码将edx赋值为$0xa$，而esi为 0。然后调用$strtol$函数。该函数的功能为把 string 转化为 long，然后，存储到$0x602780$中。

- 于是猜测$phase$ 6的功能是，将输入的字符串转化为数字后，经过$func$ 6的处理，使其与某个值相等。$func$ 6函数异常复杂，太多映射令人头疼。故而考虑利用gdb调试来测试rax和edx之间的关系。

- 在 cmp %edx,(%rax)处（0x40110e）添加断点，查看%rax 的值：

```
(gdb) b *0x40110e
Breakpoint 1 at 0x40110e
```

r

- 任意输入一个数字，例如 123456：

```
123456

Breakpoint 1, 0x000000000040110e in phase_6 ()
```

- 输入 rax 与 edx 进行查看，可以发现，edx 等于我们所输入的值 123456，而 rax 等于 673。因此 ，需要让二者相等，可以直接输入 673 进行尝试。

```
(gdb) p $edx
$1 = 123456
(gdb) p *$rax
$2 = 673
```

③进行测试：

```
673
Congratulations! You've defused the bomb! Again!
```

答案正确！我们解决了所有的问题！

五、实验结论：

经过上面的六次拆弹，将 6 个炸弹全部成功拆除如下：

```
dongyunhao_2019284073@ubuntu:~/expirement3$ ./bomb_64
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Science isn't about why, it's about why not?
Phase 1 defused. How about the next one?
1 1 1 1 1 1
That's number 2.  Keep going!
1 926
Halfway there!
9
So you got that one.  Try this one.
7 93
Congratulations! You've (mostly) defused the bomb!
Hit Control-C to escape phase 6 (for free!), but if you want to
try phase 6 for extra credit, you can continue.  Just beware!
673
Congratulations! You've defused the bomb! Again!
```