深 圳 大 学 实 验 报 告

课程名称：计算机系统(2)

实验项目名称：逆向工程实验

学院：计算机与软件学院

专业：软件工程

指导教师：冯禹洪

报告人：郑杨 学号：2020151002 班级：腾班

实验时间：2022 年 4 月 23 日至 4 月 24 日

实验报告提交时间：2022 年 5 月 4 日

教务处制

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

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)**。

四、实验过程及内容：

在开始分析 6 个关卡的函数之前，我观察了一下 main 函数中调用 Phase 函数之前所做的事情。

```
400dac:      e8 ae 08 00 00        callq  40165f <read_line>
400db1:      48 89 c7              mov    %rax,%rdi
400db4:      e8 b7 00 00 00        callq  400e70 <phase_1>
```

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

上图中可以看到，在调用 phase_1 函数之前，调用了 read_line 函数，顾名思义我猜测 read_line 函数就是从输入流中读取一行字符串并把字符串首地址返回（通过%rax 寄存器）。在这之后，把%rax 寄存器中的值复制到%rdi 寄存器中，称为 phase_1 的第一个参数，也就是说，phase_1 的参数是我们输入的字符串首地址！后面几个 phase 前面的代码都一样，那么也就是每个关卡都需要我们输入一个满足条件的字符串。所以接下来就进入各个 phase 中逐个击破。

1、phase_1

我们先分析一下 phase_1 如下：

```
0000000000400e70 <phase_1>:
  400e70:       48 83 ec 08             sub    $0x8,%rsp                       %rdi = str
  400e74:       be f8 1a 40 00          mov    $0x401af8,%esi                  %rsp -= 8 open up stack space
  400e79:       e8 bf 03 00 00          callq  40123d <strings_not_equal>      %esi = 0x401af8 = %rsi
  400e7e:       85 c0                   test   %eax,%eax                       strings_not_equal(str, 0x401af8)
  400e80:       74 05                   je     400e87 <phase_1+0x17>           compare %eax : 0
  400e82:       e8 b6 07 00 00          callq  40163d <explode_bomb>           if (%eax == 0) jump to 400e87
  400e87:       48 83 c4 08             add    $0x8,%rsp                       else bomb!!!!
  400e8b:       c3                      retq                                   %rsp += 8 destroy stack space
                                                                               return
```

第一关的函数体比较简单，我们假设输入的字符串首地址为 str，现在作为第一个参数传给 phase_1，也就是%rdi 寄存器里的值为 str。之后开辟了 8 个栈空间，暂时没有用到我们不用管他。然后把%esi 寄存器赋值为 0x401af8，因为对%esi 赋值会把%rsi 高四个字节清空，于是此时%rsi 寄存器里的值为 0x401af8。之后调用 strings_not_equal 函数，顾名思义，我猜测这个函数是判断两个字符串是否不相等，不相等返回 1，相等返回 0。于是猜测%rdi 与%rsi 作为两个参数传给 strings_not_equal 函数。现在我们分析一下 strings_not_equal 函数的汇编代码：

```
000000000040123d <strings_not_equal>:                                  %rdi = str, %rsi = 0x401af8
  40123d:       48 89 5c 24 e8          mov    %rbx,-0x18(%rsp)          save %rbx
  401242:       48 89 6c 24 f0          mov    %rbp,-0x10(%rsp)          save %rbp
  401247:       4c 89 64 24 f8          mov    %r12,-0x8(%rsp)           save %r12
  40124c:       48 83 ec 18             sub    $0x18,%rsp                %rsp -= 0x18
  401250:       48 89 fb                mov    %rdi,%rbx                 %rdi -> %rbx => %rbx = str
  401253:       48 89 f5                mov    %rsi,%rbp                 %rsi -> %rbp => %rbp = 0x401af8
  401256:       e8 c6 ff ff ff          callq  401221 <string_length>    string_length(str)
  40125b:       41 89 c4                mov    %eax,%r12d                %eax -> %r12d => %r12d = %eax = strlen(str)
  40125e:       48 89 ef                mov    %rbp,%rdi                 %rbp -> %rdi => %rdi = 0x401af8
  401261:       e8 bb ff ff ff          callq  401221 <string_length>    string_length(0x401af8) %eax = strlen(0x401af8)
  401266:       ba 01 00 00 00          mov    $0x1,%edx                 %edx = 1
  40126b:       41 39 c4                cmp    %eax,%r12d                compare strlen(str) : strlen(0x401af8)
  40126e:       75 36                   jne    4012a6 <strings_not_equal+0x69>  if (strlen(str) != strlen(0x401af8)) return 1
  401270:       0f b6 03                movzbl (%rbx),%eax               %eax = str[0]
  401273:       b2 00                   mov    $0x0,%dl                  %dl = 0
  401275:       84 c0                   test   %al,%al                   compare %al : 0
  401277:       74 2d                   je     4012a6 <strings_not_equal+0x69>  if (%al == 0) return 0;
  401279:       b2 01                   mov    $0x1,%dl                  %dl = 1
  40127b:       3a 45 00                cmp    0x0(%rbp),%al             compare str[0] : str2[0]
  40127e:       75 26                   jne    4012a6 <strings_not_equal+0x69>  if (str[0] != str2[0]) return 1
  401280:       b8 00 00 00 00          mov    $0x0,%eax                 %eax = 0
  401285:       eb 0a                   jmp    401291 <strings_not_equal+0x54>  jump to 401291
  401287:       48 83 c0 01             add    $0x1,%rax                 i ++;
  40128b:       3a 54 05 00             cmp    0x0(%rbp,%rax,1),%dl      if (str[i] != str2[i]) return 1;
  40128f:       75 10                   jne    4012a1 <strings_not_equal+0x64>
  401291:       0f b6 54 03 01          movzbl 0x1(%rbx,%rax,1),%edx     %edx = M[%rbx + %rax + 1] = str[1]
  401296:       84 d2                   test   %dl,%dl                   compare str[%rax + 1] : 0
  401298:       75 ed                   jne    401287 <strings_not_equal+0x4a>  while(str[%rax + 1] != 0) jump to 0x401287
  40129a:       ba 00 00 00 00          mov    $0x0,%edx                 return 0;
  40129f:       eb 05                   jmp    4012a6 <strings_not_equal+0x69>
  4012a1:       ba 01 00 00 00          mov    $0x1,%edx
  4012a6:       89 d0                   mov    %edx,%eax
  4012a8:       48 8b 1c 24             mov    (%rsp),%rbx
  4012ac:       48 8b 6c 24 08          mov    0x8(%rsp),%rbp
  4012b1:       4c 8b 64 24 10          mov    0x10(%rsp),%r12
  4012b6:       48 83 c4 18             add    $0x18,%rsp
  4012ba:       c3                      retq
```


如图所示，strings_not_equal 函数首先需要把用到的调用者保存寄存器保存在栈中，然后我把接下来的内容分为三部分。

首先第一部分是根据两个字符串的首地址，调用 string_length 函数计算字符串的长度，如果两个字符串长度不同那么两个字符串必然不相等，直接返回 1；如果两个字符串长度相同那么到达第二部分。

第二部分首先判断两个字符串是否为空字符串，若为空字符串则返回 0；若不为空字符串

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

则判断两个字符串的首字符是否相等，若不相等则返回 1；若首字符相等则到达第三部分。第三部分个人认为是一个 jump to middle 形式的 while 循环，首先判断下一个字符是否为结束符，若为结束符号则跳出循环；若不为结束符号则判断继续判断两个字符串的下一个字符是否相等。若在循环中遇到不相等的字符，则直接跳出循环并返回 1；若循环结束则说明两个字符串的所有字符都相同，返回 0。

分析完 strings_not_equal 函数之后，我们知道，它接受两个字符串的首地址，然后判断两个字符串是否不相同，不相同返回 1，相同返回 0。于是我们回到 phase_1 的函数体继续分析。

```
0000000000400e70 <phase_1>:                                          %rdi = str
  400e70:       48 83 ec 08           sub    $0x8,%rsp               %rsp -= 8 open up stack space
  400e74:       be f8 1a 40 00        mov    $0x401af8,%esi          %esi = 0x401af8 = %rsi
  400e79:       e8 bf 03 00 00        callq  40123d <strings_not_equal>  strings_not_equal(str, 0x401af8)
  400e7e:       85 c0                 test   %eax,%eax               compare %eax : 0
  400e80:       74 05                 je     400e87 <phase_1+0x17>   if (%eax == 0) jump to 400e87
  400e82:       e8 b6 07 00 00        callq  40163d <explode_bomb>   else bomb!!!!
  400e87:       48 83 c4 08           add    $0x8,%rsp               %rsp += 8 destroy stack space
  400e8b:       c3                    retq                           return
```

于是 0x401af8 必然是一个字符串的首地址，我们可以用 gdb 调试工具的 p 命令对这个地址的内容用字符串的形式打印出来，如下：

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

在执行 strings_not_equal 之后，返回值被放在了%rax 里面，接下来判断了%eax 寄存器的值与 0 是否相等，如果%eax 的值为 0 则跳过炸弹否则爆炸，那么就是说，如果两个字符串相等则炸弹成功拆除！于是我们输入上图字符串即可成功拆除第一个炸弹！

```
Starting program: /home/zhengyang_2020151002/csapp/bomblab/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?
```

2、phase_2

接下来进入到函数 phase_2 中进行分析

```
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
```

```
%rdi = str the string user input
①
caller stored
%rsp -= 0x48    open up stack space
%rsi = %rsp
read_six_number(str, %rsp)
②
%rbp = %rsp
%r13 = %rsp + 12
%r12 = 0
%rbx = %rbp = %rsp
%rax = %eax = M[%rbp + 12] = M[%rsp + 12]
compare %eax : M[%rbp] => compare %eax : M[%rsp]
if M[%rsp] == M[%rsp + 12] jump to 400eca
bomb!!!! if M[%rsp] != M[%rsp + 12]
%r12 += M[%rbp] => %r12 += M[%rsp] => %r12 = M[%rsp]
%rbp += 4 => %rbp = %rsp + 4
compare %rbp : %r13
if (%rbp != %r13) => if(%rbp != %rsp + 12) jump while
③
compare %r12d : 0
if (%r12d != 0) jump
bomb!!!! if(%r12d == 0)
destroy stack space
```

我还是把整个函数分为三个部分进行分析。

首先还是需要保存所用到的调用者保存寄存器，然后进入第一部分。

第一部分开辟了 72 个字节的栈空间，由调用者保存寄存器占据一部分栈空间之后，剩下 40 个字节的栈空间。之后把%rsi 寄存器的值赋值为当前的栈顶指针%rsp，作为参数传给

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

函数 read_six_numbers 函数，同时作为参数的还有%rdi，是我们输入的字符串的首地址。接下来我们先分析一下 read_six_numbers 函数：

```
0000000000401743 <read_six_numbers>:                                    %rdi = str, %rsi = %rsp
  401743:       48 83 ec 18             sub    $0x18,%rsp
  401747:       48 89 f2                mov    %rsi,%rdx                %rdx = %rsi
  40174a:       48 8d 4e 04             lea    0x4(%rsi),%rcx           %rcx = %rsi + 4
  40174e:       48 8d 46 14             lea    0x14(%rsi),%rax          %rax = %rsi + 20
  401752:       48 89 44 24 08          mov    %rax,0x8(%rsp)           M[%rsp + 8] = %rax = %rsi + 20
  401757:       48 8d 46 10             lea    0x10(%rsi),%rax          %rax = %rsi + 16
  40175b:       48 89 04 24             mov    %rax,(%rsp)              M[%rsp] = %rax = %rsi + 16
  40175f:       4c 8d 4e 0c             lea    0xc(%rsi),%r9            %r9 = %rsi + 12
  401763:       4c 8d 46 08             lea    0x8(%rsi),%r8            %r8 = %rsi + 8
  401767:       be b2 1e 40 00          mov    $0x401eb2,%esi           %rsi = %esi = 0x401eb2
  40176c:       b8 00 00 00 00          mov    $0x0,%eax                %rax = %eax = 0
  401771:       e8 3a f3 ff ff          callq  400ab0 <__isoc99_sscanf@plt>   "%d %d %d %d %d %d"
  401776:       83 f8 05                cmp    $0x5,%eax                compare %rax : 5
  401779:       7f 05                   jg     401780 <read_six_numbers+0x3d> if (%rax > 5) jump to 401780
  40177b:       e8 bd fe ff ff          callq  40163d <explode_bomb>    if (%rax <= 5) bomb!!!!
  401780:       48 83 c4 18             add    $0x18,%rsp
  401784:       c3                      retq
```

进入 read_six_numbers 函数之后是熟悉的开辟栈空间，然后就一直在准备参数，猜测是在为__isoc99_sscanf 函数准备参数。可以看到第一个参数%rdi 没有发生变化，仍然为我们输入的字符串首地址；第二个参数%rsi 被赋值为 0x401eb2；第三个参数%rdx 的值被赋为%rsi，第四个参数%rcx 的值被赋值为%rsi + 4；第五个参数%r8 的值被赋值为%rsi + 8；第六个参数%r9=%rsi+12；第七个参数放于栈中被赋值为%rsi+16；第八个参数放于栈中被赋值为%rsi+20。在这里我猜测，后六个参数为读入六个数存放的地址，从 phase_2 的栈顶指针指向的地址往后 24 个字节，也就是 6 个 int 类型的数。

对于__isoc99_sscanf 函数，我猜测与 C 语言函数 sscanf 相同，第一个和第二个参数是两个字符串首地址，分别表示文本字符串和格式化字符串，其余参数是数值存放的地址。我把 0x401eb2 以字符串形式打印出来可以得到：

```
(gdb) p (char*)0x401eb2
$1 = 0x401eb2 "%d %d %d %d %d %d"
```

也就是输入六个 int 类型数的形式。__isoc99_sscanf 函数的返回值我猜测是实际输入数字的个数，最后判断%eax 是否大于 5，就是在判断实际输入的数字个数是否大于五，也就是是否至少为 6 个数，如果不足 6 个数则炸弹被引爆。那么也就是说我们需要输入六个整数。

紧接着我们回到 phase_2 的函数体，进行第二部分与第三部分的分析：

```
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
```

```
read_six_number(str, %rsp)
%rbp = %rsp
%r13 = %rsp + 12                                          ②
%r12 = 0
%rbx = %rbp = %rsp
%rax = %eax = M[%rbp + 12] = M[%rsp + 12]
compare %eax : M[%rbp] => compare %eax : M[%rsp]
if M[%rsp] == M[%rsp + 12] jump to 400eca
bomb!!!! if M[%rsp] != M[%rsp + 12]
%r12 += M[%rbp] => %r12 += M[%rsp] => %r12 = M[%rsp]
%rbp += 4 => %rbp = %rsp + 4
compare %rbp : %r13
if (%rbp != %r13) => if(%rbp != %rsp + 12) jump while
compare %r12d : 0                                         ③
if (%r12d != 0) jump
bomb!!!! if(%r12d == 0)



destroy stack space
```

通过对 read_six_number 函数的分析，我们知道输入的六个数被放在 phase_2 的栈帧中，起始地址分别为%rsp,%rsp+4,%rsp+8,%rsp+12,%rsp+16,%rsp+20，然后进入第二部分。

第二部分首先把起始地址位于%rsp 和%rsp+12 的数值拿出来进行比较，如果不相等则炸弹被引爆，这两个数对应我们输入的第一个数和第四个数，也就是说第一个数字和第四个数字要相等才不会引爆炸弹。继续分析下面的代码可以发现这是一个循环体，继续比较第二个数和第五个数，第三个数和第六个数，都需要相等才不会引爆炸弹。至此我们知道，

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

输入的六个数字中，第一个和第四个需要相等，第二个和第五个需要相等，第三个和第六个需要相等。在循环体中，我们还可以发现，代码对前三个数进行求和。
第三部分对前三个数的求和结果进行判断，如果和为 0 则会引爆炸弹。
所以得出最终结论，输入的数字中第一个和第四个需要相等，第二个和第五个需要相等，第三个和第六个需要相等且前三个数字求和不能为 0。
于是我们输入以下数字即可拆除第二个炸弹!!

```
Phase 1 defused. How about the next one?
1 2 3 1 2 3
That's number 2.  Keep going!
```

如果数字不满足相等条件或求和为 0 均会爆炸!!

```
Phase 1 defused. How about the next one?
1 2 3 4 5 6

BOOM!!!
The bomb has blown up.
```

```
Phase 1 defused. How about the next one?
3 -1 -2 3 -1 -2

BOOM!!!
The bomb has blown up.
```

3、phase_3

接下来分析第三个炸弹如何拆除：

```
0000000000400ef9 <phase_3>:
  400ef9:       48 83 ec 18           sub    $0x18,%rsp
  400efd:       48 8d 4c 24 08        lea    0x8(%rsp),%rcx
  400f02:       48 8d 54 24 0c        lea    0xc(%rsp),%rdx
  400f07:       be be 1e 40 00        mov    $0x401ebe,%esi
  400f0c:       b8 00 00 00 00        mov    $0x0,%eax
  400f11:       e8 9a fb ff ff        callq  400ab0 <__isoc99_sscanf@plt>
  400f16:       83 f8 01              cmp    $0x1,%eax
  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)
  400f25:       77 3c                 ja     400f63 <phase_3+0x6a>
  400f27:       8b 44 24 0c           mov    0xc(%rsp),%eax
  400f2b:       ff 24 c5 60 1b 40 00  jmpq   *0x401b60(,%rax,8)
  400f32:       b8 17 02 00 00        mov    $0x217,%eax
  400f37:       eb 3b                 jmp    400f74 <phase_3+0x7b>
  400f39:       b8 d6 00 00 00        mov    $0xd6,%eax
  400f3e:       eb 34                 jmp    400f74 <phase_3+0x7b>
  400f40:       b8 53 01 00 00        mov    $0x153,%eax
  400f45:       eb 2d                 jmp    400f74 <phase_3+0x7b>
  400f47:       b8 77 00 00 00        mov    $0x77,%eax
  400f4c:       eb 26                 jmp    400f74 <phase_3+0x7b>
  400f4e:       b8 60 01 00 00        mov    $0x160,%eax
  400f53:       eb 1f                 jmp    400f74 <phase_3+0x7b>
  400f55:       b8 97 03 00 00        mov    $0x397,%eax
  400f5a:       eb 18                 jmp    400f74 <phase_3+0x7b>
  400f5c:       b8 9c 01 00 00        mov    $0x19c,%eax
  400f61:       eb 11                 jmp    400f74 <phase_3+0x7b>
  400f63:       e8 d5 06 00 00        callq  40163d <explode_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
  400f74:       3b 44 24 08           cmp    0x8(%rsp),%eax
  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
  400f83:       c3                    retq
```

```
%rdi = str
%rsp -= 0x18 open up space
%rcx = %rsp + 8
%rdx = %rsp + 12
%esi = %0x401ebe
%eax = 0
"%d %d"
compare %rax : 1
if %rax > 1 jump to 0x400f20
if (%rax <= 1) bomb!!!!
```

①

```
compare M[%rsp + 12:%rsp + 16] : 7
if (M[%rsp + 12:%rsp + 16] > 7) (unsigned compare) bomb!!!!
%eax = M[%rsp + 12 : %rsp + 15]

case 0 %eax = 0x217
case 0 break
case 2 %eax = 0xd6
case 2 break
case 3 %eax = 0x153
case 3 break
case 4 %eax = 0x77
case 4 break
case 5 %eax = 0x160
case 5 break
case 6 %eax = 0x397
case 6 break
case 7 %eax = 0x19c
case 7 break



case 1 %eax = 0x39e
```

②

```
compare %eax : M[%rsp + 8 : %rsp + 11]
if (%eax == M[%rsp + 8 : %rsp + 11]) jump
else bomb!!!!
detroy space
```

③

我依然是把 phase_3 的整个函数体分为三个部分。
首先第一部分仍然是利用__isoc99_sscanf 函数进行输入，这个函数在 phase_2 已经分析过了，就是格式化从字符串中输入，这里的第二个参数%rsi 中的值为 0x401ebe，我利用 gdb 把以 0x401ebe 为首地址的字符串输出得到：

```
(gdb) p (char*)0x401ebe
$3 = 0x401ebe "%d %d"
```

也就是说，要求输入两个 int 整数。__isoc99_sscanf 函数仍然有对实际输入个数的判断，只有输入至少包含两个整数才不会引爆炸弹。这里输入的两个数分别放在了起始地址为%rsp+12 和%rsp+8 的空间中，然后进入第二部分。
第二部分首先对输入的取出输入的第一个数进行判断，如果该数大于 7 或者小于 0（这里利用了无符号判断，只要无符号数大于 7 即可），就会引爆炸弹。所以第一个数的范围必须是是 0~7。
第二部分接下来就是 switch 的一个结构，首先可以看到跳转表的首地址为 0x401b60，然后由于是以第一个输入的值作为跳转表的偏移量，可以得到跳转表中只有八个元素（因为

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

第一个输入的值只有 0~7 八种情况），我用 gdb 的 x/8bx 命令输出以下跳转表：

```
(gdb) x/8bx 0x401b60
0x401b60:       0x32    0x0f    0x40    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x401b68
0x401b68:       0x6f    0x0f    0x40    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x401b70
0x401b70:       0x39    0x0f    0x40    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x401b78
0x401b78:       0x40    0x0f    0x40    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x401b80
0x401b80:       0x47    0x0f    0x40    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x401b88
0x401b88:       0x4e    0x0f    0x40    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x401b90
0x401b90:       0x55    0x0f    0x40    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x401b98
0x401b98:       0x5c    0x0f    0x40    0x00    0x00    0x00    0x00    0x00
```

参考跳转表就可以把每一个 case 的情况找到并做好标注。我们发现，每一个 case 都是把%eax 寄存器的值赋值为一个立即数，然后进入第三部分。

第三部分把%eax 寄存器的值与输入的第二个数进行比较，只有两个数相等才能成功拆弹。所以结论是，我们需要输入两个整数，其中第一个数的范围必须为 0~7，第二个数根据第一个数的值进行对应调整，如下表：

case	hex	dec
0	0x217	535
1	0x39e	926
2	0xd6	214
3	0x153	339
4	0x77	119
5	0x160	352
6	0x397	919
7	0x19c	412

第三个炸弹成功拆除!!

```
That's number 2.  Keep going!
0 535
Halfway there!
```

4、phase_4

继续分析第四个炸弹的拆除条件：

```
0000000000400fc1 <phase_4>:
  400fc1:       48 83 ec 18             sub    $0x18,%rsp
  400fc5:       48 8d 54 24 0c          lea    0xc(%rsp),%rdx                %rdx = %rsp + 12
  400fca:       be c1 1e 40 00          mov    $0x401ec1,%esi                %esi = 0x401ec1
  400fcf:       b8 00 00 00 00          mov    $0x0,%eax                     %eax = 0
  400fd4:       e8 d7 fa ff ff          callq  400ab0 <__isoc99_sscanf@plt>
  400fd9:       83 f8 01                cmp    $0x1,%eax                     compare %eax : 1
  400fdc:       75 07                   jne    400fe5 <phase_4+0x24>         if (%eax != 1) bomb!!!!
  400fde:       83 7c 24 0c 00          cmpl   $0x0,0xc(%rsp)                compare M[%rsp + 12 : %rsp + 15] : 0
  400fe3:       7f 05                   jg     400fea <phase_4+0x29>         if (M[%rsp + 12 : %rsp + 15] > 0) jump 0x400fea
  400fe5:       e8 53 06 00 00          callq  40163d <explode_bomb>         else bomb!!!!
  400fea:       8b 7c 24 0c             mov    0xc(%rsp),%edi                %edi = M[%rsp + 12 : %rsp + 15]
  400fee:       e8 91 ff ff ff          callq  400f84 <func4>                func4(%rdi)
  400ff3:       83 f8 37                cmp    $0x37,%eax                    compare %eax : 0x37
  400ff6:       74 05                   je     400ffd <phase_4+0x3c>         if (%eax == 0x37) jump 0x400ffd
  400ff8:       e8 40 06 00 00          callq  40163d <explode_bomb>         else bomb!!!!
  400ffd:       48 83 c4 18             add    $0x18,%rsp
  401001:       c3                      retq
```

可以看到 phase_4 一开始就调用__isoc99_sscanf 函数进行输入，此时是以 0x401ec1 为格式化字符串的首地址，使用 gdb 打印出来观察：

```
(gdb) p (char*)0x401ec1
$6 = 0x401ec1 "%d"
```

那么也就是要输入一个数，可以看到这个数会放在第三个参数%rdx 指向的地址中，此时%rdx 为%rsp+12。在输入结束之后，仍然判断了实际输入的个数，如果输入的数的个数

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

不为 1，则会引爆炸弹。然后判断了输入的数是否大于 0，若小于等于 0 则引爆炸弹，若大于 0 则调用 func4 函数并把刚刚输入的值作为第一个参数%rdi 传给 func4 函数。至此，我们得到，只能输入一个大于 0 的数。

接下来分析一下 func4 函数的功能：

```
0000000000400f84 <func4>:                                       %rdi = M[%rsp + 12 : %rsp + 15] x //fibonacci
  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               %ebx = %edi
  400f94:       b8 01 00 00 00          mov    $0x1,%eax               %eax = 1
  400f99:       83 ff 01                cmp    $0x1,%edi               compare %edi : 1
  400f9c:       7e 14                   jle    400fb2 <func4+0x2e>     if (%edi <= 1) jump to 400fb2 return 1
  400f9e:       8d 7b ff                lea    -0x1(%rbx),%edi         %edi = %rbx - 1
  400fa1:       e8 de ff ff ff          callq  400f84 <func4>          func(x - 1)
  400fa6:       89 c5                   mov    %eax,%ebp               %ebp = %eax
  400fa8:       8d 7b fe                lea    -0x2(%rbx),%edi         %edi = %rbx - 2
  400fab:       e8 d4 ff ff ff          callq  400f84 <func4>          func(x - 2)
  400fb0:       01 e8                   add    %ebp,%eax               %eax += %ebp
  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
```

这个函数比较简单如果当前参数 x 小于等于 1，那么直接返回 1；否则返回 func(x-1)+func(x-2)。不难看出这是一个求解第 x 项斐波那契数的函数。

回到 phase_4 的函数体，在调用 func4 之后，它判断了返回值是否与 0x37（55）相等，只有相等才能成功拆除炸弹。由于 55 是第九项斐波那契数，故推测出输入 9 即可拆除第四个炸弹！！

```
Halfway there!
9
So you got that one.  Try this one.
```

5、phase_5

继续分析第五关：

```
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
```

```
%rcx = %rsp + 8
%rdx = %rsp + 12                                                ①
%esi = 0x401ebe "%d %d"
%eax = 0
input two integers
compare %eax : 1
if (%eax > 1) jump to 0x401029
else bomb!!!!
%eax = M[%rsp + 12 ~ %rsp + 15]
%eax &= 0xf
M[%rsp + 12 ~ %rsp + 15] = %eax
compare %eax : 0xf
if (%eax == 0xf) bomb!!!!
%ecx = 0
%edx = 0
```

```
%edx = %edx + 1 = 1                                             ②
%eax signed extend to %rax
%eax = array[%eax]
%ecx += %eax
compare %eax : 0xf
if (%eax != 0xf) jump to 0x401043
```

```
M[%rsp + 12 ~ %rsp + 15] = %eax                                 ③
compare %edx : 12
if (%edx != 12) bomb!!!!
compare %ecx : M[%rsp + 8 ~ %rsp + 11]
if (%ecx == M[%rsp + 8 ~ %rsp + 11]) jump to 0x40106a
else bomb!!!!
```

仍然把整个函数体分为三个部分。

第一部分首先还是输入，这个输入的格式化字符串与第三题的一致，如下：

```
(gdb) p (char*)0x401ebe
$3 = 0x401ebe "%d %d"
```

所以要求输入两个整型数字，分别存放在以%rsp+12 与%rsp+8 为首地址的空间中。然后取出第一个输入值放到寄存器%eax 中，并把%eax 中的值也就是第一个输入值和 0xf 做与操作，这个操作相当于取出%eax 中的值的最低 4 位 bit，也就是第一个输入值的最低 4 位

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

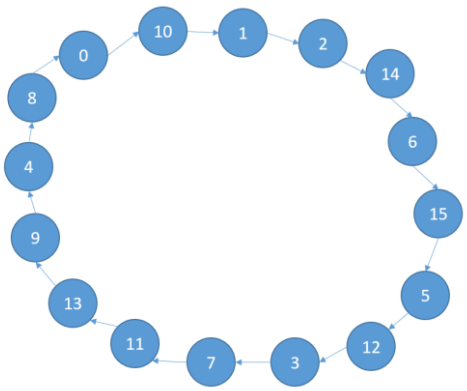
bit，那么此时%eax 中值的范围为 0~15。接下来判断%eax 中的值是否为 0xf（15），如果是的话就引爆炸弹，否则把寄存器%ecx 和%edx 清空并跳到第二部分。至此，我们可以看出，拆除炸弹的第一个条件是第一个输入值与 0xf 按位与之后的值不能为 0xf。

第二部分是一个 do….while 框架，%edx 寄存器充当一个计数器，记录循环执行的次数。每一次循环把%eax 的值符号扩展到%rax（其实这里好像不需要扩展），然后取出以 0x401ba0+4*%rax 为首地址往后的四个字节放到%eax 中。然后把%eax 寄存器的值累加到%ecx 寄存器中。之后判断此时%eax 中的值是否为 0xf，若为 0xf 则跳出循环；否则继续循环。循环体里涉及到了 0x401ba0 这个地址，我把这个地址往后的信息以四个字节为一组输出，发现这里存放了一个有 16 个元素的 int 类型数组。

```
(gdb) x/4bx 0x401ba0
0x401ba0 <array.3014>:  0x0a    0x00    0x00    0x00
(gdb) x/4bx 0x401ba4
0x401ba4 <array.3014+4>:        0x02    0x00    0x00    0x00
(gdb) x/4bx 0x401ba8
0x401ba8 <array.3014+8>:        0x0e    0x00    0x00    0x00
(gdb) x/4bx 0x401bac
0x401bac <array.3014+12>:       0x07    0x00    0x00    0x00
(gdb) x/4bx 0x401bb0
0x401bb0 <array.3014+16>:       0x08    0x00    0x00    0x00
(gdb) x/4bx 0x401bb4
0x401bb4 <array.3014+20>:       0x0c    0x00    0x00    0x00
(gdb) x/4bx 0x401bb8
0x401bb8 <array.3014+24>:       0x0f    0x00    0x00    0x00
(gdb) x/4bx 0x401bbc
0x401bbc <array.3014+28>:       0x0b    0x00    0x00    0x00
(gdb) x/4bx 0x401bc0
0x401bc0 <array.3014+32>:       0x00    0x00    0x00    0x00
(gdb) x/4bx 0x401bc4
0x401bc4 <array.3014+36>:       0x04    0x00    0x00    0x00
(gdb) x/4bx 0x401bc8
0x401bc8 <array.3014+40>:       0x01    0x00    0x00    0x00
(gdb) x/4bx 0x401bcc
0x401bcc <array.3014+44>:       0x0d    0x00    0x00    0x00
(gdb) x/4bx 0x401bd0
0x401bd0 <array.3014+48>:       0x03    0x00    0x00    0x00
(gdb) x/4bx 0x401bd4
0x401bd4 <array.3014+52>:       0x09    0x00    0x00    0x00
(gdb) x/4bx 0x401bd8
0x401bd8 <array.3014+56>:       0x06    0x00    0x00    0x00
(gdb) x/4bx 0x401bdc
0x401bdc <array.3014+60>:       0x05    0x00    0x00    0x00
```

把该数组称为 a，则 a[0] = 10, a[1] = 2, a[2] = 14, a[3] = 7, a[4] = 8, a[5] = 12, a[6] = 15, a[7] = 11, a[8] = 0, a[9] = 4, a[10] = 1, a[11] = 13, a[12] = 3, a[13] = 9, a[14] = 6, a[15] = 5。

循环体做的事情其实就是把数组里元素的值取出来作为下标直到该值为 0xf 也就是 15 的时候停止，循环期间把沿途数组元素的值进行累加并且记录数组循环次数。那么其实可以根据数组元素的值画出下面的图：



注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

我们发现所有元素构成了一个环，我们输入的第一个数就会成为起点，而 15 为终点。接下来解释 phase_5 的第三部分。

第三部分将%edx 寄存器里的值与 12 进行比较，如果不等于 12 则引爆炸弹，之前我们分析过，%edx 寄存器里的值为循环体执行的次数，那么循环体执行的次数必须为 12。于是由上图可以确定出我们输入的第一个数必须为 7。在这之后，还需将%ecx 寄存器里的值与我们输入的第二个值比较，如果相等炸弹才不会被引爆，此时%ecx 寄存器里的值为：11+13+9+4+8+0+10+1+2+14+6+15=93，也就是说，我们输入的第二个数字必须为 93 才能成功拆除第五个炸弹!!

```
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!
```

6、phase_6

第六关个人认为是最难的一关了，首先分析一下 phase_6 的函数体：

```
00000000004010d9 <phase_6>:                                     %rdi = str
  4010d9:       48 83 ec 08             sub    $0x8,%rsp
  4010dd:       ba 0a 00 00 00          mov    $0xa,%edx         %edx = 10
  4010e2:       be 00 00 00 00          mov    $0x0,%esi         %esi = 0
  4010e7:       e8 94 fa ff ff          callq  400b80 <strtol@plt>   string to long??
  4010ec:       89 05 8e 16 20 00       mov    %eax,0x20168e(%rip)   # 602780 <node0>
  4010f2:       bf 80 27 60 00          mov    $0x602780,%edi    %edi = 0x602780
  4010f7:       e8 73 ff ff ff          callq  40106f <fun6>     func6(0x602780)
  4010fc:       48 8b 40 08             mov    0x8(%rax),%rax    %rax = M[%rax + 8]
  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>
  40110e:       39 10                   cmp    %edx,(%rax)       compare M[%rax] : %edx
  401110:       74 05                   je     401117 <phase_6+0x3e>   if ( == ) jump to 0x401117
  401112:       e8 26 05 00 00          callq  40163d <explode_bomb>   else bomb!!!!
  401117:       48 83 c4 08             add    $0x8,%rsp
  40111b:       c3                      retq
```

开辟了栈空间之后，调用了 strtol 函数，猜测是把输入的字符串变为 long 类型的数字放在%rax 中返回，之后把%rax 的值赋给首地址为 0x602780 的空间，再把 0x602780 赋值给%edi 寄存器作为参数传递给 func6。根据提示，0x602780 中存放了一个名为 node0 的内容，我猜测是结构体，然后我把 0x602780 中的内容输出来，发现它包含了 10 个 node：

```
(gdb) x/8bx 0x602780
0x602780 <node0>:       0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x602788
0x602788 <node0+8>:     0x90    0x27    0x60    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x602790
0x602790 <node1>:       0x76    0x01    0x00    0x00    0x01    0x00    0x00    0x00
(gdb) x/8bx 0x602798
0x602798 <node1+8>:     0xa0    0x27    0x60    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x6027a0
0x6027a0 <node2>:       0x3a    0x03    0x00    0x00    0x02    0x00    0x00    0x00
(gdb) x/8bx 0x6027a8
0x6027a8 <node2+8>:     0xb0    0x27    0x60    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x6027b0
0x6027b0 <node3>:       0x72    0x01    0x00    0x00    0x03    0x00    0x00    0x00
(gdb) x/8bx 0x6027b8
0x6027b8 <node3+8>:     0xc0    0x27    0x60    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x6027c0
0x6027c0 <node4>:       0x0e    0x03    0x00    0x00    0x04    0x00    0x00    0x00
(gdb) x/8bx 0x6027c8
0x6027c8 <node4+8>:     0xd0    0x27    0x60    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x6027d0
0x6027d0 <node5>:       0xe8    0x01    0x00    0x00    0x05    0x00    0x00    0x00
(gdb) x/8bx 0x6027d8
0x6027d8 <node5+8>:     0xe0    0x27    0x60    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x6027e0
0x6027e0 <node6>:       0xa1    0x02    0x00    0x00    0x06    0x00    0x00    0x00
(gdb) x/8bx 0x6027e8
0x6027e8 <node6+8>:     0xf0    0x27    0x60    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x6027f0
0x6027f0 <node7>:       0x1e    0x01    0x00    0x00    0x07    0x00    0x00    0x00
(gdb) x/8bx 0x6027f8
0x6027f8 <node7+8>:     0x00    0x28    0x60    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x602800
0x602800 <node8>:       0x58    0x02    0x00    0x00    0x08    0x00    0x00    0x00
(gdb) x/8bx 0x602808
0x602808 <node8+8>:     0x10    0x28    0x60    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x602810
0x602810 <node9>:       0x11    0x02    0x00    0x00    0x09    0x00    0x00    0x00
(gdb) x/8bx 0x602818
0x602818 <node9+8>:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) x/8bx 0x602820
0x602820 <lab_id>:      0x63    0x6f    0x75    0x72    0x73    0x65    0x72    0x61
(gdb)
```

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

发现在每一个 node 的前四个字节中存放了一个值，在后八个字节中存放了一个地址，这个地址是下一个 node 的首地址，中间四个字节应该是编号，从 0~9。那么我猜测这是一个链表，我们输入的值放在了链表的第一个元素中。

接下来进入 func6 函数体中分析：

```
000000000040106f <fun6>:                                    %rdi = 0x602780
  40106f:       4c 8b 47 08             mov    0x8(%rdi),%r8        %r8 = M[%rdi + 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              compare %r8 : 0
  401084:       75 40                   jne    4010c6 <fun6+0x57>   if (%r8) jump to 4010c6
  401086:       48 89 f8                mov    %rdi,%rax            %rax = %rdi
  401089:       c3                      retq                        return
  40108a:       48 89 d1                mov    %rdx,%rcx            %rcx = %rdx
  40108d:       48 8b 51 08             mov    0x8(%rcx),%rdx       %rdx = M[%rcx + 8]
  401091:       48 85 d2                test   %rdx,%rdx            compare %rdx : 0
  401094:       74 09                   je     40109f <fun6+0x30>   if (%rdx == 0) jump to 0x40109f
  401096:       39 32                   cmp    %esi,(%rdx)          compare M[%rdx] : %esi
  40109a:       48 89 cf                mov    %rcx,%rdi            %rdi = %rcx
  40109d:       eb 03                   jmp    4010a2 <fun6+0x33>   jmp to 0x4010a2
  40109f:       48 89 cf                mov    %rcx,%rdi            %rdi = %rcx
  4010a2:       48 39 d7                cmp    %rdx,%rdi            compare %rdi : %rdx
  4010a5:       74 06                   je     4010ad <fun6+0x3e>   if (%rdi == %rdx) jump to 0x4010ad
  4010a7:       4c 89 47 08             mov    %r8,0x8(%rdi)        M[%rdi + 8] = %r8
  4010ab:       eb 03                   jmp    4010b0 <fun6+0x41>   jump to 0x4010b0
  4010ad:       4c 89 c0                mov    %r8,%rax             %rax = %r8
  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            compare %rcx : 0
  4010bb:       74 1a                   je     4010d7 <fun6+0x68>   if (%rcx == 0) jump to 0x4010d7
  4010bd:       49 89 c8                mov    %rcx,%r8             %r8 = %rcx
  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            %rdx = %rcx
  4010c9:       48 85 c9                test   %rcx,%rcx            compare %rcx : 0
  4010cc:       74 d4                   je     4010a2 <fun6+0x33>   if (%rcx == 0) jump to 0x4010a2
  4010ce:       41 8b 30                mov    (%r8),%esi           %esi = M[%r8]
  4010d1:       39 31                   cmp    %esi,(%rcx)          compare %esi : M[%rcx]
  4010d3:       7f b8                   jg     40108d <fun6+0x1e>   if (%esi > 0) jump to 0x40108d
  4010d5:       eb cb                   jmp    4010a2 <fun6+0x33>   jump to 0x4010a2
  4010d7:       f3 c3                   repz retq                   return
```

分析了 func6 函数体之后，我发现它包含了很多条件跳转指令，直接分析汇编代码非常的难，我通过对数据进行模拟来分析这个函数，假设随便输入了一个值之后，分析了 func6 的运行过程。

通过模拟分析，我发现 func6 函数的功能是对这个链表进行插入排序（降序），最终返回链表首元素的地址。具体的插入排序过程就是，从第一个节点开始维护一个有序的降序的序列，每次新加入一个数就从有序序列中从头往后遍历找到应该插入的位置，把新节点插入到有序序列中使序列长度增加 1。其中，各个寄存器的功能分别是，%r8 记录一个地址指向当前新加入的节点，%rax 保存当前有序序列的头节点地址，%rdi、%rcx 和%rdx 都作为临时寄存器辅助循环过程。

执行完函数 func6 之后回到 phase_6 函数体进行分析：

```
00000000004010d9 <phase_6>:                                 %rdi = str
  4010d9:       48 83 ec 08             sub    $0x8,%rsp
  4010dd:       ba 0a 00 00 00          mov    $0xa,%edx            %edx = 10
  4010e2:       be 00 00 00 00          mov    $0x0,%esi            %esi = 0
  4010e7:       e8 94 fa ff ff          callq  400b80 <strtol@plt>  string to long??
  4010ec:       89 05 8e 16 20 00       mov    %eax,0x20168e(%rip)  # 602780 <node0>
  4010f2:       bf 80 27 60 00          mov    $0x602780,%edi       %edi = 0x602780
  4010f7:       e8 73 ff ff ff          callq  40106f <fun6>        func6(0x602780)
  4010fc:       48 8b 40 08             mov    0x8(%rax),%rax       %rax = M[%rax + 8]
  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>
  40110e:       39 10                   cmp    %edx,(%rax)          compare M[%rax] : %edx
  401110:       74 05                   je     401117 <phase_6+0x3e> if ( == ) jump to 0x401117
  401112:       e8 26 05 00 00          callq  40163d <explode_bomb> else bomb!!!!
  401117:       48 83 c4 08             add    $0x8,%rsp
  40111b:       c3                      retq
```

在 func6 返回之后，把%rax 中的值连续赋值为 M[%rax+8]这个操作实际上就是找到链表的第四个元素。之后判断该元素的值是否为我们输入的那个数，如果不是的话则炸弹被引爆，如果是的话则成功通关。于是结论是，我们输入的数需要排在第四位！除了第一个节

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

点的值由我们自己输入以外，其他值都已经存在内存中了，分别是：
374 826 370 782 488 673 286 600 529
那么他们从大到小排序之后得到：
826 782 673 600 529 488 374 370 286
要使得我们输入的值排在第四位，那么我们输入的值的范围位[600,673]。
第六个炸弹成功拆除！

```
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!
601
Congratulations! You've defused the bomb! Again!
```

五、实验结论：

仔细分析了大部分汇编代码，梳理出了每一关的通关条件。

通过了所有的关卡，截图如下：

```
Starting program: /home/zhengyang_2020151002/csapp/bomblab/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 2 3 1 2 3
That's number 2.  Keep going!
0 535
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!
601
Congratulations! You've defused the bomb! Again!
```

六、心得体会：

通过分析每一句汇编代码，我了解到每一个函数的功能，知道了每一个使得炸弹爆炸的条件，顺利通过。通过这个实验我对于机器级代码的理解更上一层楼，从理论学习到实验动手实践，个人觉得已经基本掌握了汇编代码。
在做这个实验的过程中，也遇到过一些小问题，比如说一开始不知道下面这个函数的功能：

```
callq  400ab0 <__isoc99_sscanf@plt>
```

但是想到 C 语言的函数 sscanf 就猜测是不是这个函数，由于 sscanf 需要两个字符串指针参数，正好在调用这个函数之前也准备了两个字符串的首地址（放在了%rdi 和%rsi 寄存器中），由于%rdi 此时又是我们输入字符串的首地址，我就猜测%rsi 中会不会是格式化的那个字符串，于是把%rsi 首地址指向的内容用字符串的形式打印出来，发现确实是这样的。

```
(gdb) p (char*)0x401eb2
$1 = 0x401eb2 "%d %d %d %d %d %d"
```

于是也就验证了我的猜想。

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

一开始还出现了一点小问题，就是在做 Phase2 的时候，把十六进制 0x10 直接就当成十进制的 10 去分析了，后来觉得有哪里不对才发现这个表示的是十进制的 16。 处理完了一开始的小问题之后，后面几道题做起来就得心应手了。 对于最后一道题复杂的跳转指令，个人的解决方法是模拟程序运行过程，在过程中发现各个寄存器功能最后推导出函数的功能。
指导教师批阅意见： 成绩评定： 指导教师签字：冯禹洪 2022 年　　月　　日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。