CSAPP--BombLab

原创

maihc1 于 2021-08-23 18:38:17 发布 416 收藏 1

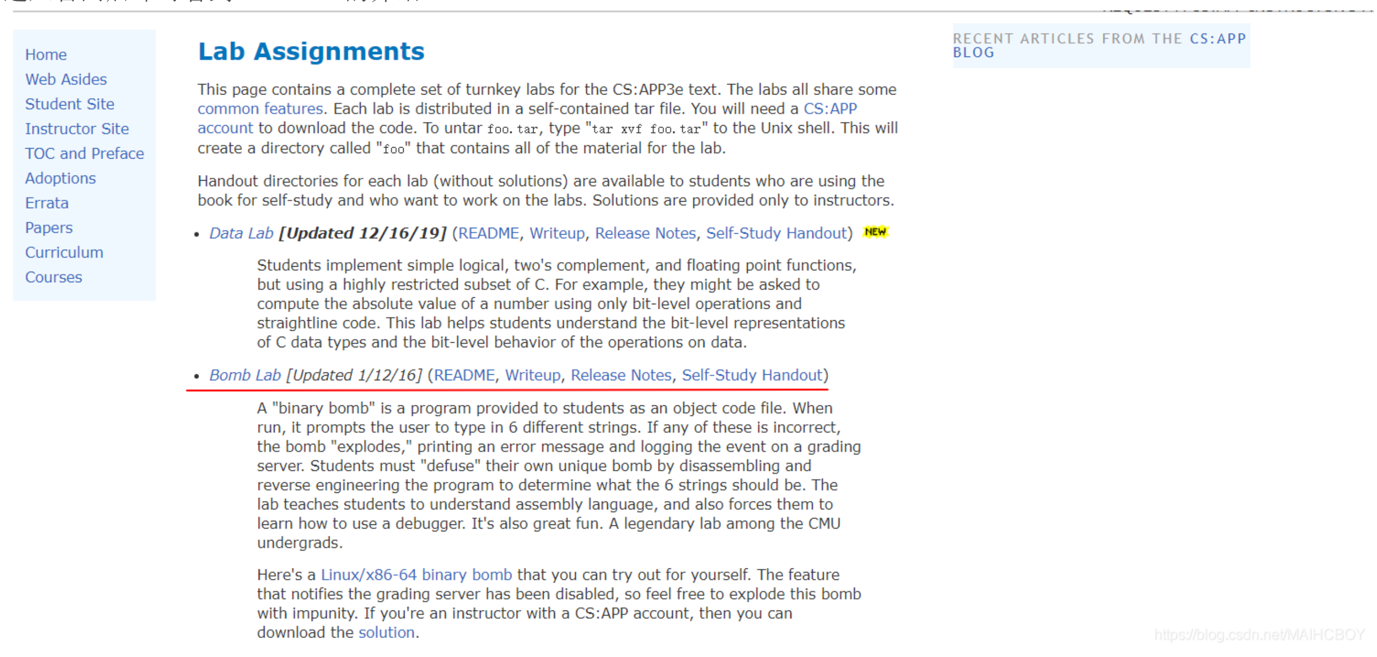
BombLab

一、 Lab介绍

官网地址

在这里首先给出BombLab的官网地址：http://csapp.cs.cmu.edu/3e/labs.html

进入官网后即可看到BombLab的介绍：



Home
Web Asides
Student Site
Instructor Site
TOC and Preface
Adoptions
Errata
Papers
Curriculum
Courses

Lab Assignments

This page contains a complete set of turnkey labs for the CS:APP3e text. The labs all share some common features. Each lab is distributed in a self-contained tar file. You will need a CS:APP account to download the code. To untar foo.tar, type "tar xvf foo.tar" to the Unix shell. This will create a directory called "foo" that contains all of the material for the lab.

Handout directories for each lab (without solutions) are available to students who are using the book for self-study and who want to work on the labs. Solutions are provided only to instructors.

- *Data Lab* ***[Updated 12/16/19]*** (README, Writeup, Release Notes, Self-Study Handout) NEW

  Students implement simple logical, two's complement, and floating point functions, but using a highly restricted subset of C. For example, they might be asked to compute the absolute value of a number using only bit-level operations and straightline code. This lab helps students understand the bit-level representations of C data types and the bit-level behavior of the operations on data.

- *Bomb Lab [Updated 1/12/16]* (README, Writeup, Release Notes, Self-Study Handout)

  A "binary bomb" is a program provided to students as an object code file. When run, it prompts the user to type in 6 different strings. If any of these is incorrect, the bomb "explodes," printing an error message and logging the event on a grading server. Students must "defuse" their own unique bomb by disassembling and reverse engineering the program to determine what the 6 strings should be. The lab teaches students to understand assembly language, and also forces them to learn how to use a debugger. It's also great fun. A legendary lab among the CMU undergrads.

  Here's a Linux/x86-64 binary bomb that you can try out for yourself. The feature that notifies the grading server has been disabled, so feel free to explode this bomb with impunity. If you're an instructor with a CS:APP account, then you can download the solution.

RECENT ARTICLES FROM THE CS:APP BLOG

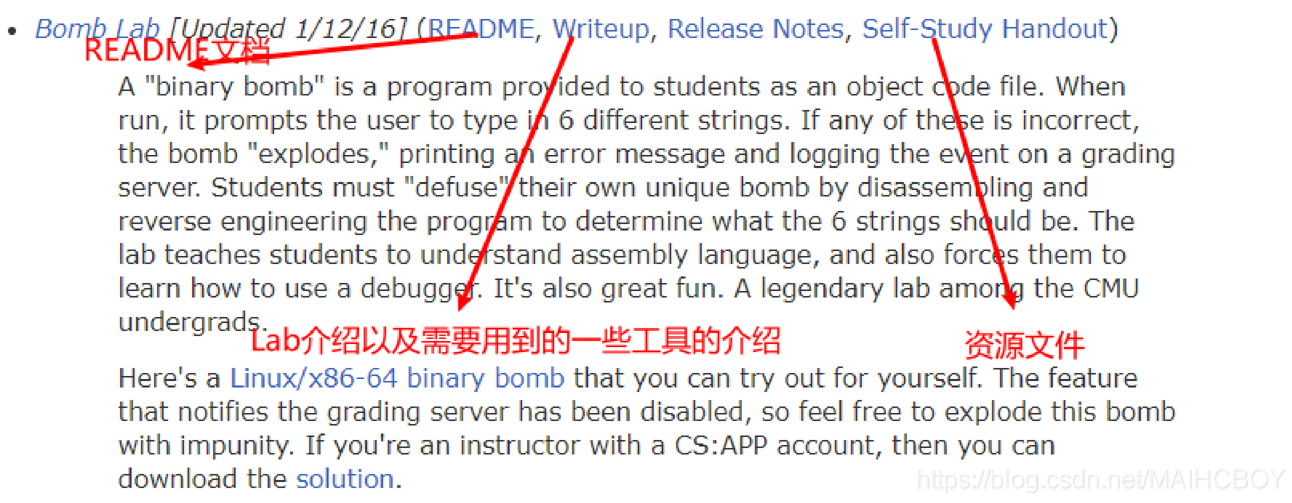
简单翻译一下：BombLab其实就是只提供给你一个可执行文件，让你利用对可执行文件进行逆向，分析逆向出来的汇编代码，以**提高机器级编程的能力**。BombLab总共有六关，难度也是越来越高，但一个个分析出来的话，成就感是非常高的！！！

前置知识

1、 了解Linux操作系统的基本使用，能熟练使用Linux的基本指令。
2、 熟练掌握**x86汇编语言**（如果不熟悉x86汇编语言，可以去CSAPP第三章学习）
3、 熟练掌握gdb

二、 环境准备

下载相关资源

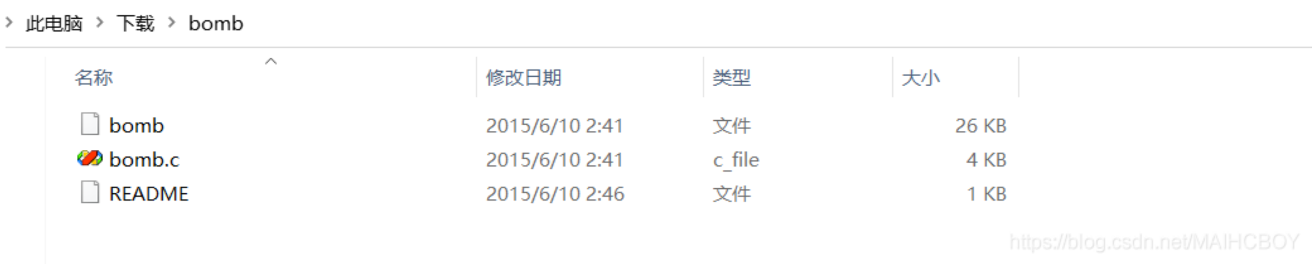

- *Bomb Lab [Updated 1/12/16]* (README, Writeup, Release Notes, Self-Study Handout)

  A "binary bomb" is a program provided to students as an object code file. When run, it prompts the user to type in 6 different strings. If any of these is incorrect, the bomb "explodes," printing an error message and logging the event on a grading server. Students must "defuse" their own unique bomb by disassembling and reverse engineering the program to determine what the 6 strings should be. The lab teaches students to understand assembly language, and also forces them to learn how to use a debugger. It's also great fun. A legendary lab among the CMU undergrads.

  Here's a Linux/x86-64 binary bomb that you can try out for yourself. The feature that notifies the grading server has been disabled, so feel free to explode this bomb with impunity. If you're an instructor with a CS:APP account, then you can download the solution.

我们用到的资源也就两个：Writeup和Self-Study Handout

1. Writeup里面放置着Lab的介绍
2. Self-StudyHandout里面即是资源文件（一个c文件、一个可执行文件、一个README文档）。
   Self-Study Handout下载下来解压完成之后如下：



名称	修改日期	类型	大小
bomb	2015/6/10 2:41	文件	26 KB
bomb.c	2015/6/10 2:41	c_file	4 KB
README	2015/6/10 2:46	文件	1 KB

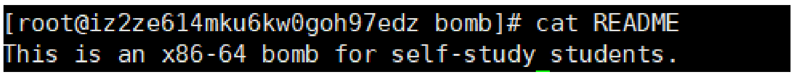
Linux环境搭建
因为该Lab必须基于Linux操作系统，所以我建议完全不了解Linux的小伙伴先去补一下Linux的基础知识，再返回来做这个Lab。在这里推荐一下韩顺平老师的Linux课程，在B站有视频。
Linux环境有了之后，之后就是检查Linux系统是否有我们开发所需的工具。
最关键的两个工具分别是**objdump**和**gdb**。这里我们就只演示gdb，objdump也是一样的道理。
首先需要判断本机是否有gdb。使用命令 `which gdb` 。
如果下面显示出gdb的路径来：如 `usr/bin/gdb` ，则说明本机已有gdb。如果不是，则使用命令 `yum install gdb` 命令下载gdb。
好的，以上所有关于环境的问题，我们就都解决了，下面我们就可以正式开启BombLab了！！！

三、 具体内容

Level 1

首先，将资源文件放入Linux系统后，我们需要做的一件事便是查看各个资源文件：

README文件如下：



```
[root@iz2ze614mku6kw0goh97edz bomb]# cat README
This is an x86-64 bomb for self-study students.
```

C文件如下：

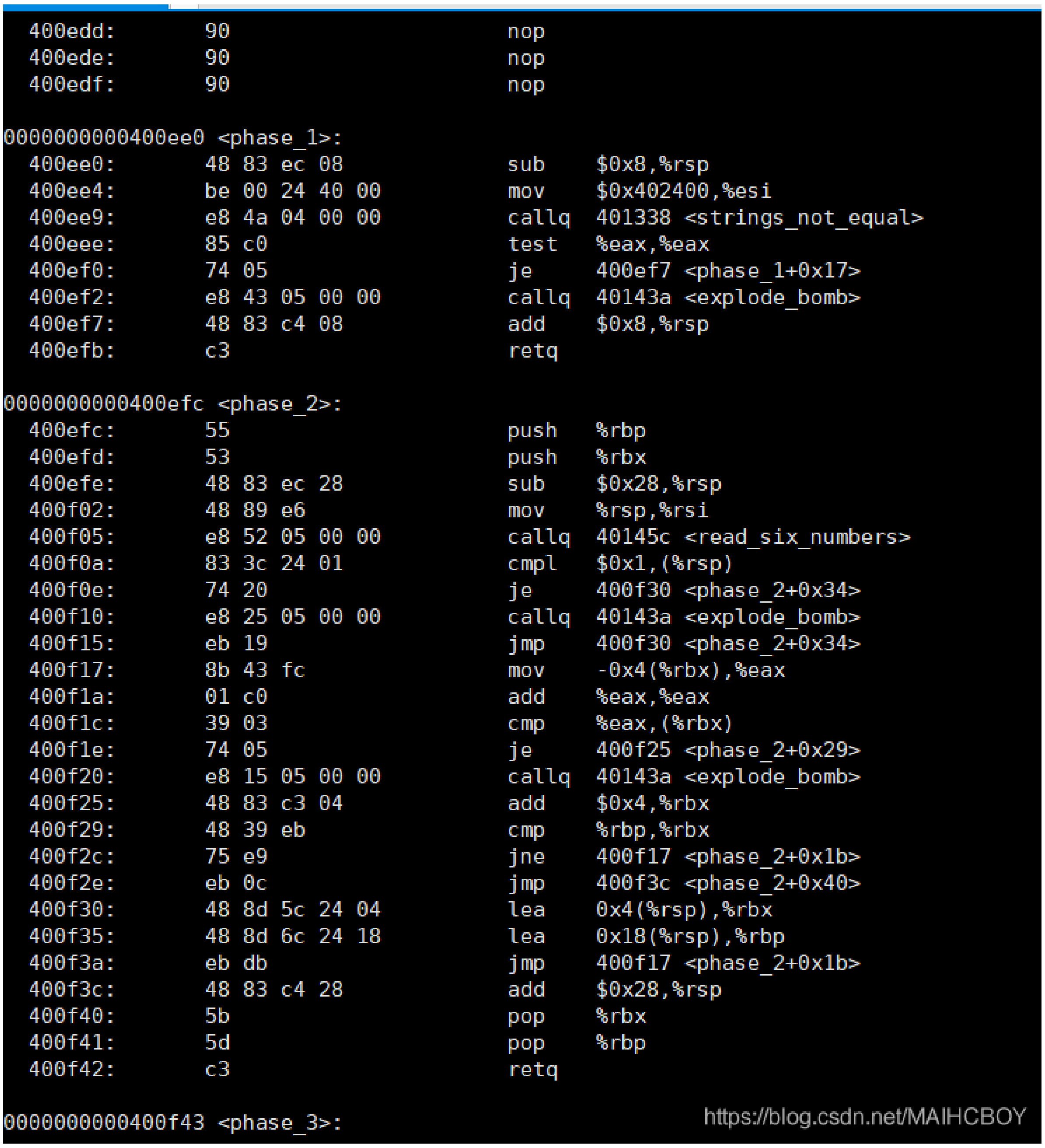
```c
/***************************************************************************
 * Dr. Evil's Insidious Bomb, Version 1.1
 * Copyright 2011, Dr. Evil Incorporated. All rights reserved.
 *
 * LICENSE:
 *
 * Dr. Evil Incorporated (the PERPETRATOR) hereby grants you (the
 * VICTIM) explicit permission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

#include <stdio.h>
#include <stdlib.h>
#include "support.h"
#include "phases.h"

/*
 * Note to self: Remember to erase this file so my victims will have no
 * idea what is going on, and so they will all blow up in a
 * spectaculary fiendish explosion. -- Dr. Evil
 */

FILE *infile;

int main(int argc, char *argv[])
{
    char *input;

    /* Note to self: remember to port this bomb to Windows and put a
     * fantastic GUI on it. */

    /* When run with no arguments, the bomb reads its input lines
     * from standard input. */
    if (argc == 1) {
 infile = stdin;
    }
```

```c
   }

   /* When run with one argument <file>, the bomb reads from <file>
    * until EOF, and then switches to standard input. Thus, as you
    * defuse each phase, you can add its defusing string to <file> and
    * avoid having to retype it. */
   else if (argc == 2) {
if (!(infile = fopen(argv[1], "r"))) {
    printf("%s: Error: Couldn't open %s\n", argv[0], argv[1]);
    exit(8);
}
   }

   /* You can't call the bomb with more than 1 command line argument. */
   else {
printf("Usage: %s [<input_file>]\n", argv[0]);
exit(8);
   }

   /* Do all sorts of secret stuff that makes the bomb harder to defuse. */
   initialize_bomb();

   printf("Welcome to my fiendish little bomb. You have 6 phases with\n");
   printf("which to blow yourself up. Have a nice day!\n");

   /* Hmm...  Six phases must be more secure than one phase! */
   input = read_line();             /* Get input                   */
   phase_1(input);                  /* Run the phase               */
   phase_defused();                 /* Drat!  They figured it out!
         * Let me know how they did it. */
   printf("Phase 1 defused. How about the next one?\n");

   /* The second phase is harder.  No one will ever figure out
    * how to defuse this... */
   input = read_line();
   phase_2(input);
   phase_defused();
   printf("That's number 2.  Keep going!\n");

   /* I guess this is too easy so far.  Some more complex code will
    * confuse people. */
   input = read_line();
   phase_3(input);
   phase_defused();
   printf("Halfway there!\n");

   /* Oh yeah?  Well, how good is your math?  Try on this saucy problem! */
   input = read_line();
   phase_4(input);
   phase_defused();
   printf("So you got that one.  Try this one.\n");

   /* Round and 'round in memory we go, where we stop, the bomb blows! */
   input = read_line();
   phase_5(input);
   phase_defused();
   printf("Good work!  On to the next...\n");

   /* This phase will never be used, since no one will get past the
    * earlier ones.  But just in case, make this one extra hard. */
    input = read_line();
    phase_6(input);
    phase_defused();

    /* Wow, they got it!  But isn't something... missing?  Perhaps
     * something they overlooked?  Mua ha ha ha ha! */

    return 0;
}
```

通过分析C文件我们可以发现，分别调用6个函数，分别代表六关，每一关都需要我们输入一个字符串，然后将该字符串的首地址传入函数，如果传入的字符串正确，函数正常返回，进入下一关。如果不正确，则闯关失败！

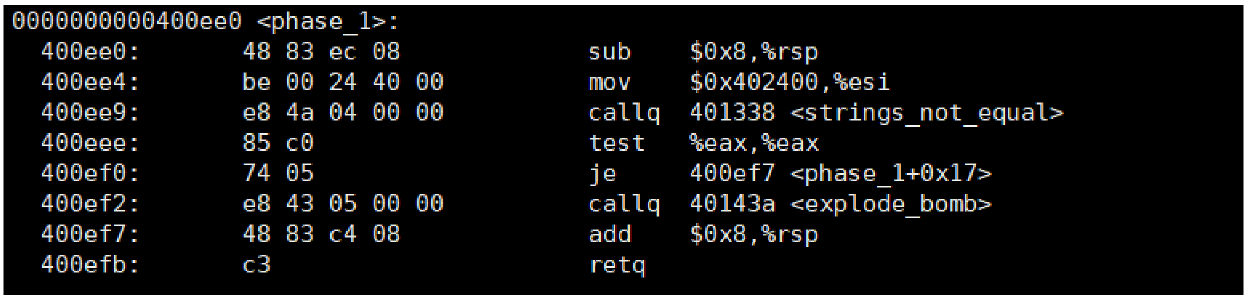
可执行文件不可直接查看，我们将其反汇编并写入bomb.txt中。
使用命令：`objdump -S -d bomb > bomb.txt` 即可完成上面的操作。
查看bomb.txt就可以发现可执行文件所需的所有函数的反汇编代码都可以看到，当然我们只需要对我们有用的就够了。由于代码太多，我这里就不全放过来了。

部分代码如下：



```
  400edd:       90                      nop
  400ede:       90                      nop
  400edf:       90                      nop

0000000000400ee0 <phase_1>:
  400ee0:       48 83 ec 08             sub    $0x8,%rsp
  400ee4:       be 00 24 40 00          mov    $0x402400,%esi
  400ee9:       e8 4a 04 00 00          callq  401338 <strings_not_equal>
  400eee:       85 c0                   test   %eax,%eax
  400ef0:       74 05                   je     400ef7 <phase_1+0x17>
  400ef2:       e8 43 05 00 00          callq  40143a <explode_bomb>
  400ef7:       48 83 c4 08             add    $0x8,%rsp
  400efb:       c3                      retq

0000000000400efc <phase_2>:
  400efc:       55                      push   %rbp
  400efd:       53                      push   %rbx
  400efe:       48 83 ec 28             sub    $0x28,%rsp
  400f02:       48 89 e6                mov    %rsp,%rsi
  400f05:       e8 52 05 00 00          callq  40145c <read_six_numbers>
  400f0a:       83 3c 24 01             cmpl   $0x1,(%rsp)
  400f0e:       74 20                   je     400f30 <phase_2+0x34>
  400f10:       e8 25 05 00 00          callq  40143a <explode_bomb>
  400f15:       eb 19                   jmp    400f30 <phase_2+0x34>
  400f17:       8b 43 fc                mov    -0x4(%rbx),%eax
  400f1a:       01 c0                   add    %eax,%eax
  400f1c:       39 03                   cmp    %eax,(%rbx)
  400f1e:       74 05                   je     400f25 <phase_2+0x29>
  400f20:       e8 15 05 00 00          callq  40143a <explode_bomb>
  400f25:       48 83 c3 04             add    $0x4,%rbx
  400f29:       48 39 eb                cmp    %rbp,%rbx
  400f2c:       75 e9                   jne    400f17 <phase_2+0x1b>
  400f2e:       eb 0c                   jmp    400f3c <phase_2+0x40>
  400f30:       48 8d 5c 24 04          lea    0x4(%rsp),%rbx
  400f35:       48 8d 6c 24 18          lea    0x18(%rsp),%rbp
  400f3a:       eb db                   jmp    400f17 <phase_2+0x1b>
  400f3c:       48 83 c4 28             add    $0x28,%rsp
  400f40:       5b                      pop    %rbx
  400f41:       5d                      pop    %rbp
  400f42:       c3                      retq

0000000000400f43 <phase_3>:
```

接下来，我们就可以分析第一关的反汇编代码了。

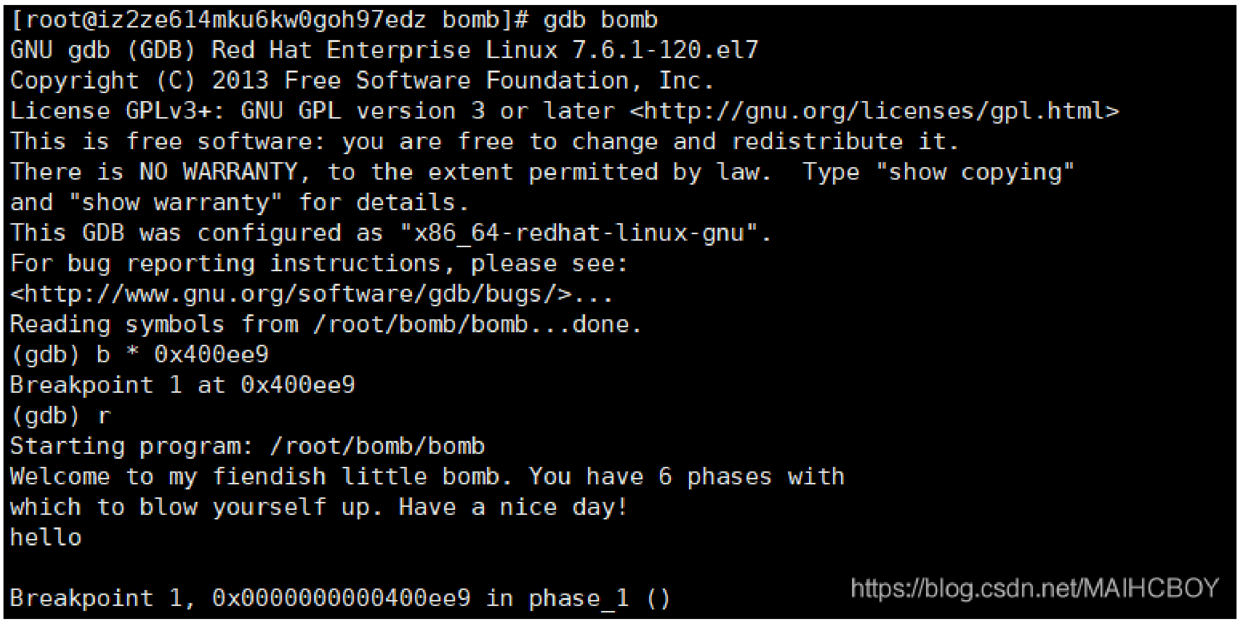


```
0000000000400ee0 <phase_1>:
  400ee0:       48 83 ec 08             sub    $0x8,%rsp
  400ee4:       be 00 24 40 00          mov    $0x402400,%esi
  400ee9:       e8 4a 04 00 00          callq  401338 <strings_not_equal>
  400eee:       85 c0                   test   %eax,%eax
  400ef0:       74 05                   je     400ef7 <phase_1+0x17>
  400ef2:       e8 43 05 00 00          callq  40143a <explode_bomb>
  400ef7:       48 83 c4 08             add    $0x8,%rsp
  400efb:       c3                      retq
```

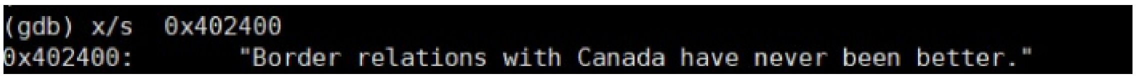
400ee0：即对栈帧进行初始化。
400ee4：将$0x402400传入%esi中
400ee9：调用strings_not_equal函数，将%rdi和%rsi传入，%rdi即为我们输入的字符串的首地址，%rsi即为$0x402400
400eee：将strings_not_equal函数的返回值%eax与0进行比较
400ef0：若%eax为0跳转到400ef7，成功返回函数
400ef2：若%eax不为0，执行这句，调用explode_bomb，即引爆炸弹，闯关失败。

通过以上分析，我们可以得出：只需满足$0x402400里存放的字符串与我们输入的字符串相等即可。

我们通过gdb运行程序bomb，在0x400ee9处打上断点。暂时随便输入一个字符串，查看0x402400处存放的字符串。



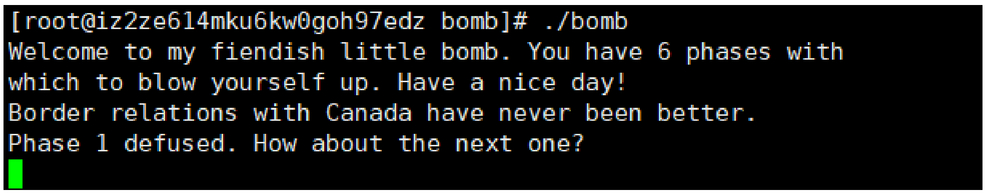
```
[root@iz2ze614mku6kw0goh97edz bomb]# gdb bomb
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /root/bomb/bomb...done.
(gdb) b * 0x400ee9
Breakpoint 1 at 0x400ee9
(gdb) r
Starting program: /root/bomb/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
hello

Breakpoint 1, 0x0000000000400ee9 in phase_1 ()
```



```
(gdb) x/s  0x402400
0x402400:       "Border relations with Canada have never been better."
```

可以看到，0x402400处存放的字符串为：“Border relations with Canada have never been better.”
所以第一关的答案就是`"Border relations with Canada have never been better."`

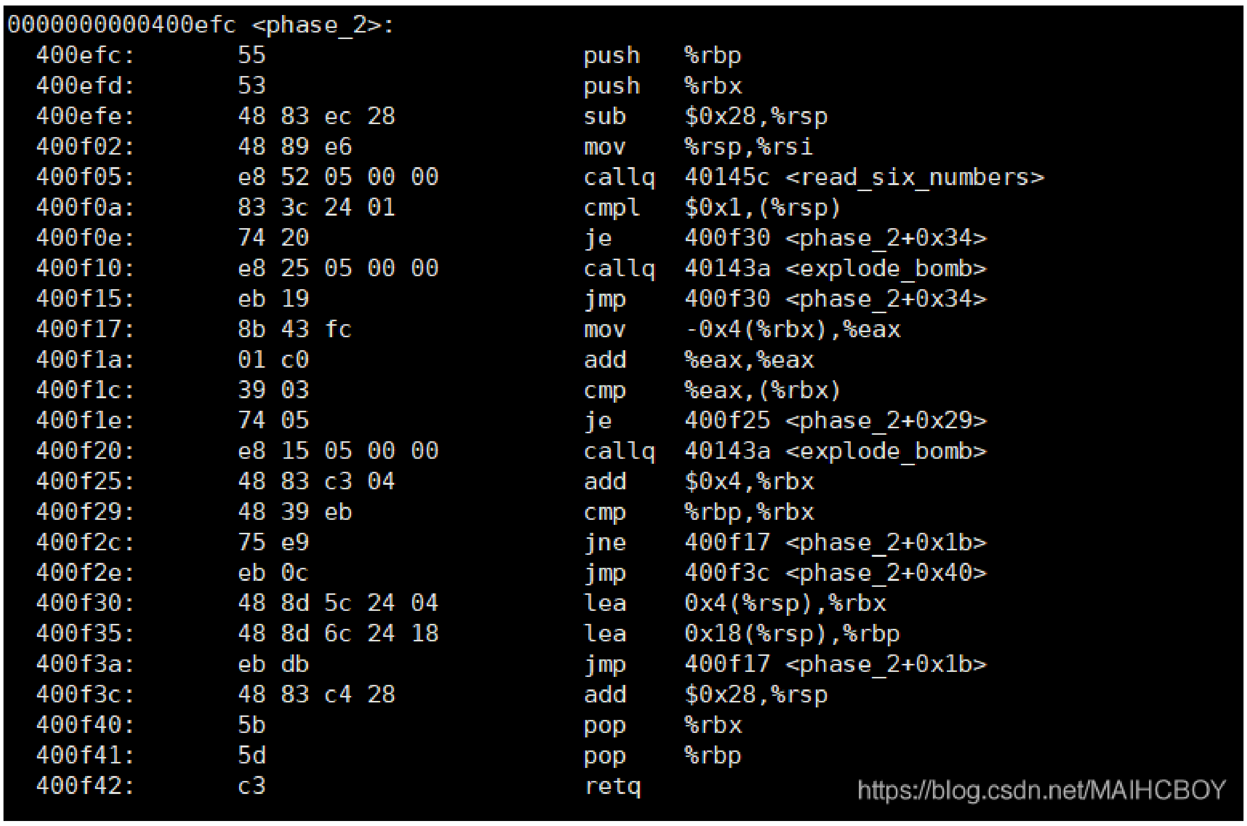
我们输入答案，可以看到进入第二关了！！！



```
[root@iz2ze614mku6kw0goh97edz bomb]# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
```

2. Level 2

首先，我们分析第二关的反汇编代码：

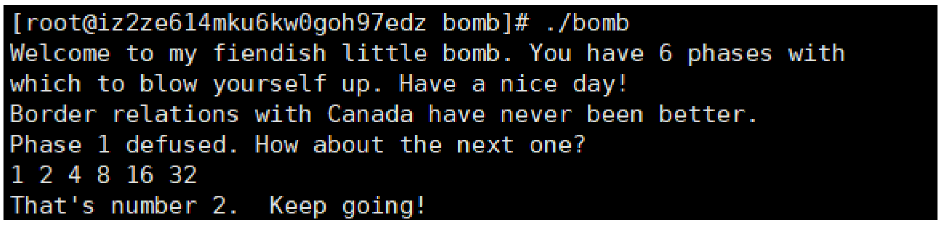


```
0000000000400efc <phase_2>:
  400efc:       55                      push   %rbp
  400efd:       53                      push   %rbx
  400efe:       48 83 ec 28             sub    $0x28,%rsp
  400f02:       48 89 e6                mov    %rsp,%rsi
  400f05:       e8 52 05 00 00          callq  40145c <read_six_numbers>
  400f0a:       83 3c 24 01             cmpl   $0x1,(%rsp)
  400f0e:       74 20                   je     400f30 <phase_2+0x34>
  400f10:       e8 25 05 00 00          callq  40143a <explode_bomb>
  400f15:       eb 19                   jmp    400f30 <phase_2+0x34>
  400f17:       8b 43 fc                mov    -0x4(%rbx),%eax
  400f1a:       01 c0                   add    %eax,%eax
  400f1c:       39 03                   cmp    %eax,(%rbx)
  400f1e:       74 05                   je     400f25 <phase_2+0x29>
  400f20:       e8 15 05 00 00          callq  40143a <explode_bomb>
  400f25:       48 83 c3 04             add    $0x4,%rbx
  400f29:       48 39 eb                cmp    %rbp,%rbx
  400f2c:       75 e9                   jne    400f17 <phase_2+0x1b>
  400f2e:       eb 0c                   jmp    400f3c <phase_2+0x40>
  400f30:       48 8d 5c 24 04          lea    0x4(%rsp),%rbx
  400f35:       48 8d 6c 24 18          lea    0x18(%rsp),%rbp
  400f3a:       eb db                   jmp    400f17 <phase_2+0x1b>
  400f3c:       48 83 c4 28             add    $0x28,%rsp
  400f40:       5b                      pop    %rbx
  400f41:       5d                      pop    %rbp
  400f42:       c3                      retq
```

400efc~400efe：对栈帧进行初始化
400f02：将栈指针放入%rsi中
400f05：调用read_six_numbers函数，将%rdi和%rsi传入，%rdi即为我们输入的字符串首地址，%rsi即为栈指针
400f0a：比较栈指针指向的数据是否等于1，若等于1，跳转到400f30，若不等于1，接着执行400f10，即引爆炸弹，闯关失败。
400f30：%rsp + 0x4 赋值给 %rbx
400f35：%rsp + 0x18 赋值给 %rbp
400f3a：无条件跳转到400f17
400f17：将（%rbx - 0x4）处的值赋值给%eax
400f1a：将%eax的值乘2
400f1c：比较%eax和（%rbx）处的值比较，若相等，则跳转到400f25，若不相等，则继续执行400f20，即引爆炸弹，闯关失败。
400f25：将%rbx的值+4
400f29：比较%rbp和%rbx的值，如果不相等，则继续跳转到400f17，如果相等则跳转到400f3c，即成功返回函数。

通过以上分析，我们得出结论，我们需要输入一个包含6个数字的字符串，而read_six_numbers函数会将我们字符串的六个数字保存在栈中，这六个数字需要满足的条件是：1、第一个数字为1。2、 之后每个是之前的2倍。所以得出第二关的答案：`1 2 4 8 16 32`

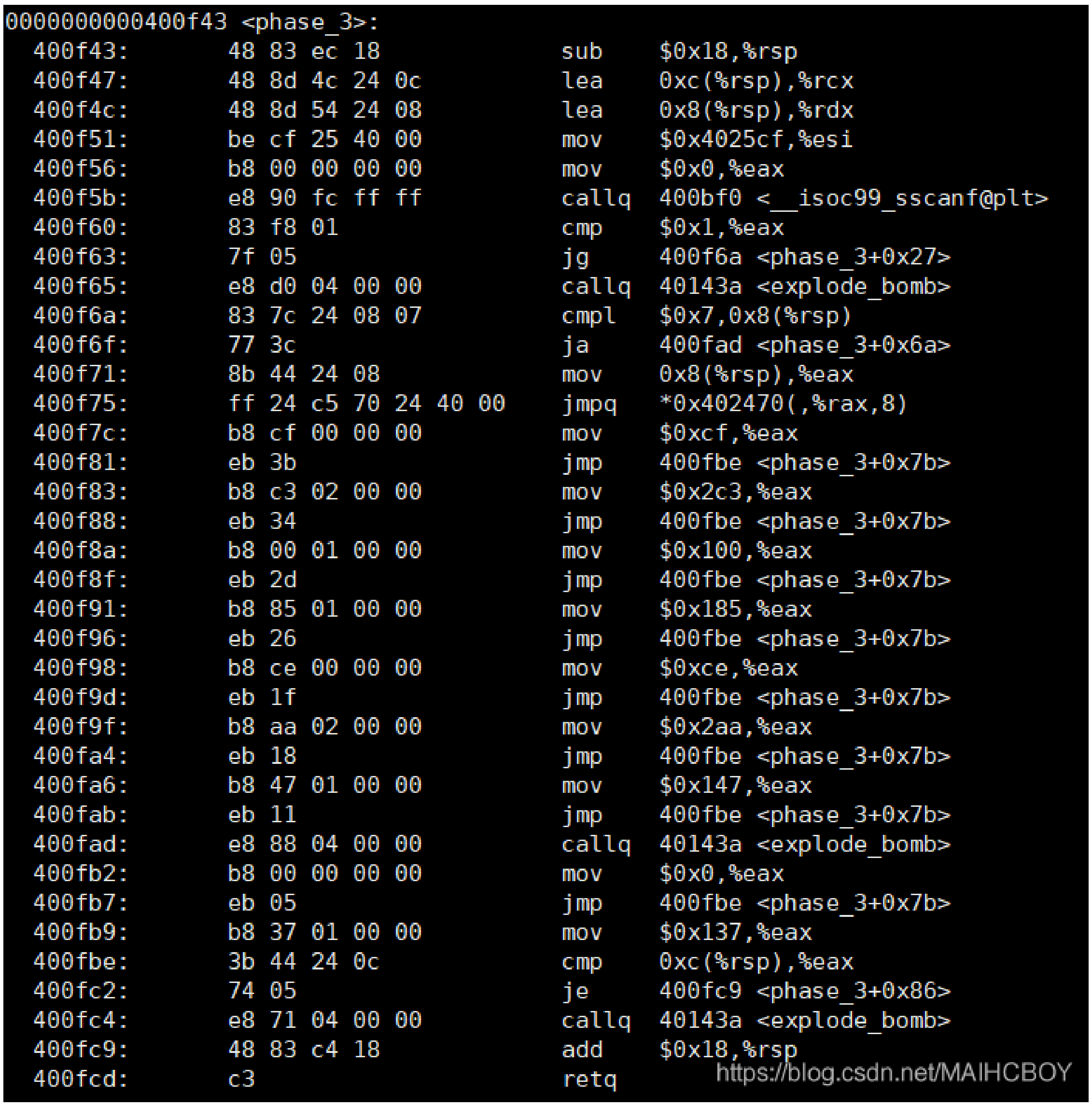
我们输入答案，可以看到进入第三关了！！！



```
[root@iz2ze614mku6kw0goh97edz bomb]# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
```

Level 3

首先，我们分析第三关的反汇编代码：



```
0000000000400f43 <phase_3>:
  400f43:       48 83 ec 18             sub    $0x18,%rsp
  400f47:       48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
  400f4c:       48 8d 54 24 08          lea    0x8(%rsp),%rdx
  400f51:       be cf 25 40 00          mov    $0x4025cf,%esi
  400f56:       b8 00 00 00 00          mov    $0x0,%eax
  400f5b:       e8 90 fc ff ff          callq  400bf0 <__isoc99_sscanf@plt>
  400f60:       83 f8 01                cmp    $0x1,%eax
  400f63:       7f 05                   jg     400f6a <phase_3+0x27>
  400f65:       e8 d0 04 00 00          callq  40143a <explode_bomb>
  400f6a:       83 7c 24 08 07          cmpl   $0x7,0x8(%rsp)
  400f6f:       77 3c                   ja     400fad <phase_3+0x6a>
  400f71:       8b 44 24 08             mov    0x8(%rsp),%eax
  400f75:       ff 24 c5 70 24 40 00    jmpq   *0x402470(,%rax,8)
  400f7c:       b8 cf 00 00 00          mov    $0xcf,%eax
  400f81:       eb 3b                   jmp    400fbe <phase_3+0x7b>
  400f83:       b8 c3 02 00 00          mov    $0x2c3,%eax
  400f88:       eb 34                   jmp    400fbe <phase_3+0x7b>
  400f8a:       b8 00 01 00 00          mov    $0x100,%eax
  400f8f:       eb 2d                   jmp    400fbe <phase_3+0x7b>
  400f91:       b8 85 01 00 00          mov    $0x185,%eax
  400f96:       eb 26                   jmp    400fbe <phase_3+0x7b>
  400f98:       b8 ce 00 00 00          mov    $0xce,%eax
  400f9d:       eb 1f                   jmp    400fbe <phase_3+0x7b>
  400f9f:       b8 aa 02 00 00          mov    $0x2aa,%eax
  400fa4:       eb 18                   jmp    400fbe <phase_3+0x7b>
  400fa6:       b8 47 01 00 00          mov    $0x147,%eax
  400fab:       eb 11                   jmp    400fbe <phase_3+0x7b>
  400fad:       e8 88 04 00 00          callq  40143a <explode_bomb>
  400fb2:       b8 00 00 00 00          mov    $0x0,%eax
  400fb7:       eb 05                   jmp    400fbe <phase_3+0x7b>
  400fb9:       b8 37 01 00 00          mov    $0x137,%eax
  400fbe:       3b 44 24 0c             cmp    0xc(%rsp),%eax
  400fc2:       74 05                   je     400fc9 <phase_3+0x86>
  400fc4:       e8 71 04 00 00          callq  40143a <explode_bomb>
  400fc9:       48 83 c4 18             add    $0x18,%rsp
  400fcd:       c3                      retq
```

400f43：栈帧的初始化
400f47：将 %rsp + 0xc 赋值给%rcx
400f4c：将 %rsp + 0x8 赋值给%rdx
400f51：将 0x4025cf 赋值给%esi
400f56：将 0x0 赋值给%eax
400f5b：调用函数__isoc99_sscanf@plt
400f60：比较0x1和%eax的大小，若大于1，跳转到400f6a，反之，引爆炸弹，闯关失败。
400f6a：比较0x7和（0x8 + %rsp）的大小，若大于7，则引爆炸弹，闯关失败，若不大于7，继续执行400f71。
400f71：将（%rsp + 0x8）赋值给%eax
400f75：跳转到（0x402070 + 0x8 * %rax）处保存的地址处，即（0x402070 + 0x8 * %rax）是一个地址，我们要去的地方的地址，存在这个地址的数据处。
400f7c~400fab：这部分即上一步要跳转的地址，将特定数据赋值给%eax，然后跳转到400fbe
400fbe：比较（0xc + %rsp）中的值和%eax，若等于，则跳转到400fc9，即函数正常结束。若不等于，则继续执行400fc4，即引爆炸弹，闯关失败。

通过以上分析，我们可以得出：我们输入的字符串需要两个整数，（0x8 + %rsp）中存放的便是第一个数字，（0xc + %rsp）中存放的便是第二个数字。第一个数字的取值不可大于7，故取值范围为：0~7。第一个数字的每个对应的第二个数字都在400f7c – 400fab处告诉我们了。所以第三关的答案不止一个。
下面列出所有答案：

`0 207`

`1 311`

`2 707`

`3 256`

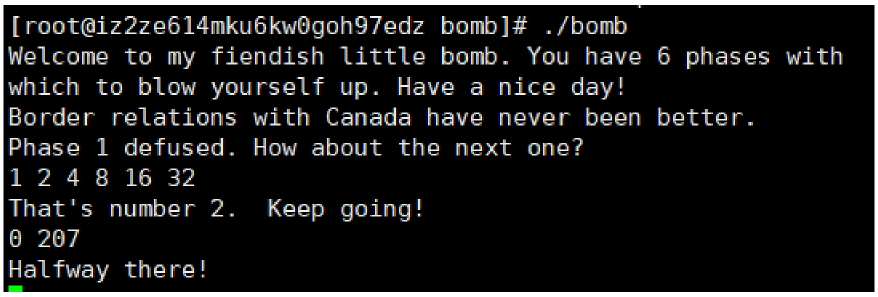
`4 389`

`5 206`

`6 682`

`7 327`

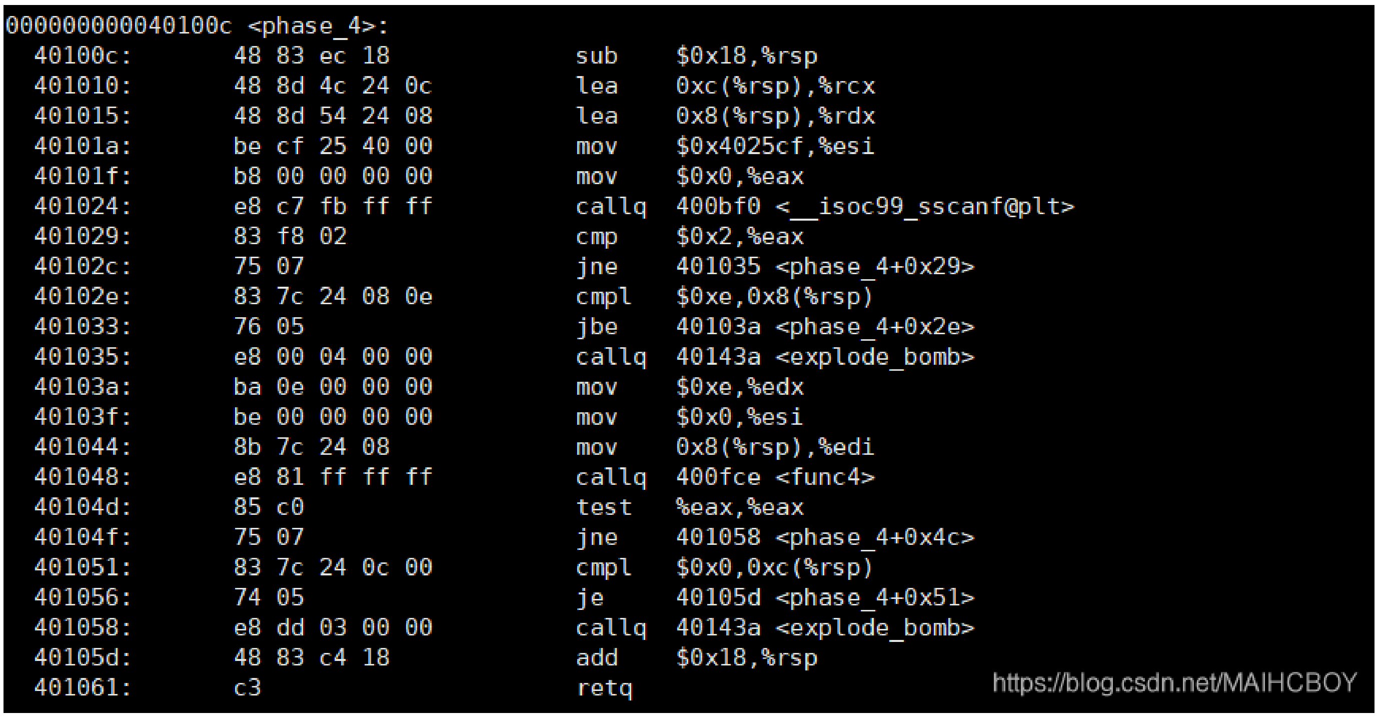
我们输入答案，可以看到进入第四关了！！！



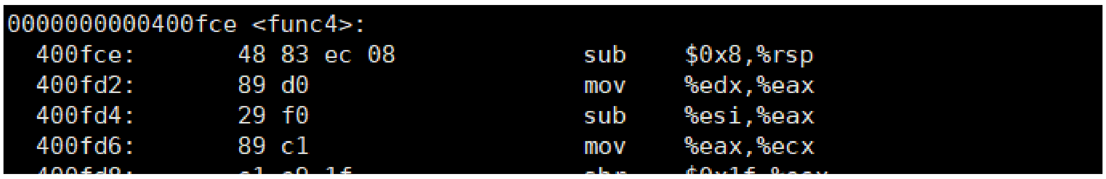
```
[root@iz2ze614mku6kw0goh97edz bomb]# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
0 207
Halfway there!
```

Level 4

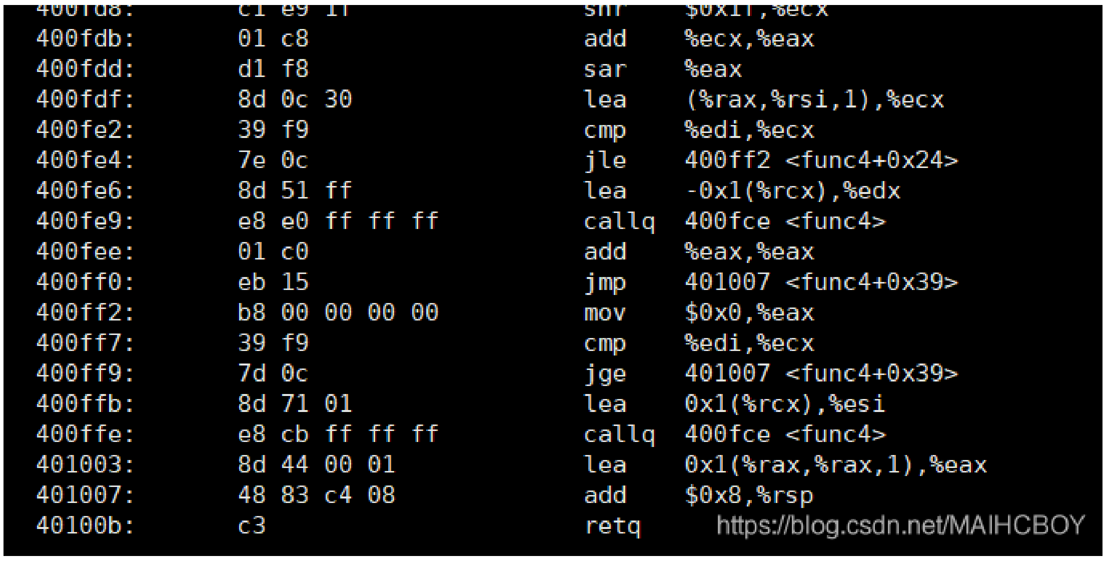
首先，我们分析第四关的反汇编代码：



```
000000000040100c <phase_4>:
  40100c:       48 83 ec 18             sub    $0x18,%rsp
  401010:       48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
  401015:       48 8d 54 24 08          lea    0x8(%rsp),%rdx
  40101a:       be cf 25 40 00          mov    $0x4025cf,%esi
  40101f:       b8 00 00 00 00          mov    $0x0,%eax
  401024:       e8 c7 fb ff ff          callq  400bf0 <__isoc99_sscanf@plt>
  401029:       83 f8 02                cmp    $0x2,%eax
  40102c:       75 07                   jne    401035 <phase_4+0x29>
  40102e:       83 7c 24 08 0e          cmpl   $0xe,0x8(%rsp)
  401033:       76 05                   jbe    40103a <phase_4+0x2e>
  401035:       e8 00 04 00 00          callq  40143a <explode_bomb>
  40103a:       ba 0e 00 00 00          mov    $0xe,%edx
  40103f:       be 00 00 00 00          mov    $0x0,%esi
  401044:       8b 7c 24 08             mov    0x8(%rsp),%edi
  401048:       e8 81 ff ff ff          callq  400fce <func4>
  40104d:       85 c0                   test   %eax,%eax
  40104f:       75 07                   jne    401058 <phase_4+0x4c>
  401051:       83 7c 24 0c 00          cmpl   $0x0,0xc(%rsp)
  401056:       74 05                   je     40105d <phase_4+0x51>
  401058:       e8 dd 03 00 00          callq  40143a <explode_bomb>
  40105d:       48 83 c4 18             add    $0x18,%rsp
  401061:       c3                      retq
```



```
0000000000400fce <func4>:
  400fce:       48 83 ec 08             sub    $0x8,%rsp
  400fd2:       89 d0                   mov    %edx,%eax
  400fd4:       29 f0                   sub    %esi,%eax
  400fd6:       89 c1                   mov    %eax,%ecx
```



```
  400fd8:       c1 e9 1f                shr    $0x1f,%ecx
  400fdb:       01 c8                   add    %ecx,%eax
  400fdd:       d1 f8                   sar    %eax
  400fdf:       8d 0c 30                lea    (%rax,%rsi,1),%ecx
  400fe2:       39 f9                   cmp    %edi,%ecx
  400fe4:       7e 0c                   jle    400ff2 <func4+0x24>
  400fe6:       8d 51 ff                lea    -0x1(%rcx),%edx
  400fe9:       e8 e0 ff ff ff          callq  400fce <func4>
  400fee:       01 c0                   add    %eax,%eax
  400ff0:       eb 15                   jmp    401007 <func4+0x39>
  400ff2:       b8 00 00 00 00          mov    $0x0,%eax
  400ff7:       39 f9                   cmp    %edi,%ecx
  400ff9:       7d 0c                   jge    401007 <func4+0x39>
  400ffb:       8d 71 01                lea    0x1(%rcx),%esi
  400ffe:       e8 cb ff ff ff          callq  400fce <func4>
  401003:       8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
  401007:       48 83 c4 08             add    $0x8,%rsp
  40100b:       c3                      retq
```

40100c：栈帧的从初始化

401010：将 %rsp + 0xc 赋值给%rcx

401015：将 %rsp + 0x8 赋值给%rdx

40101a：将 0x4025cf 赋值给%esi

40101f：将 0 赋值给%eax

401024：调用函数__isoc99_sscanf@plt

401029：比较函数返回值%eax和0x2的大小，若不等于2，则跳转到401035，即引爆炸弹，闯关失败。若等于2，则继续执行40102e

40102e：比较（0x8 + %rsp）处的值和 0xe 的大小，若小于等于，则跳转到40103a。反之，则继续执行，即引爆炸弹，闯关失败。

40103a：将0xe赋值给%edx

40103f：将0x0赋值给%esi

401044：将（0x8 + %rsp）赋值给%edi

401048：跳转到400fce，调用func4函数，参数分别为%rdi，%rsi，%rdx

400fce：func4函数栈帧的初始化

400fd2：将%edx赋值给%eax

400fd4：%eax减去%esi

400fd6：将%eax赋值给%ecx

400fd8：将%ecx逻辑右移0x1f位

400fdb：将 %eax + %ecx 赋值给 %eax

400fdd：将 %eax 算术右移一位

400fdf： 将 %rax + %rsi * 1 赋值给%ecx

400fe2：比较%ecx和%edi，若%ecx小于等于%edi，则跳转到400ff2。反之，将%rcx - 0x1赋值给%edx，继续递归调用func4函数。

400ff2： 将0x0赋值给%eax

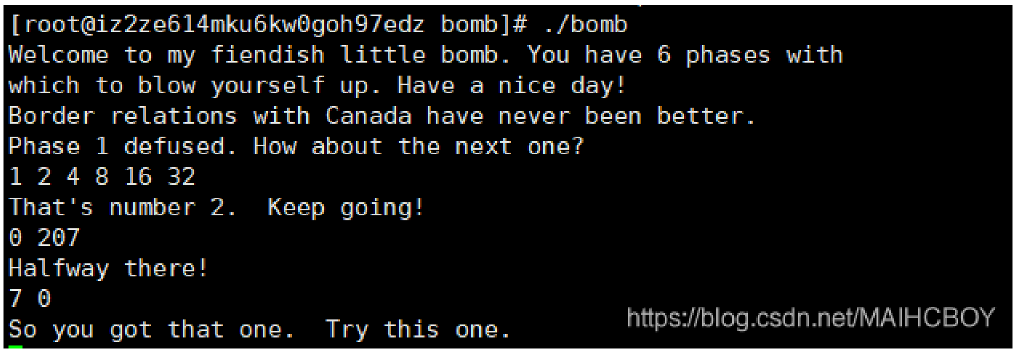
400ff7： 比较%ecx和%edi的值，若%ecx大于等于%edi，则跳转到401007，函数正常返回，转而执行40104d。反之，将%rcx + 0x1赋值给%esi，继续递归调用func4函数。

40104d：比较0x0与返回值%eax的大小，若不等于，则调用401058，即引爆炸弹，闯关失败。反之继续执行401051

401051：比较（0xc + %rsp）的值与0x0的大小，若相等，函数正常结束，若不相等，引爆炸弹，闯关失败。

通过以上分析，我们可以得出：这一关我们输入的字符串须是两个数字，并且第二个数字必须是0，第一个数字必须小于等于0xe。通过分析func4中的代码，我们可以发现当第一个数字为0x7时，func4函数可以正常返回0。故第四关的答案为：`7 0`

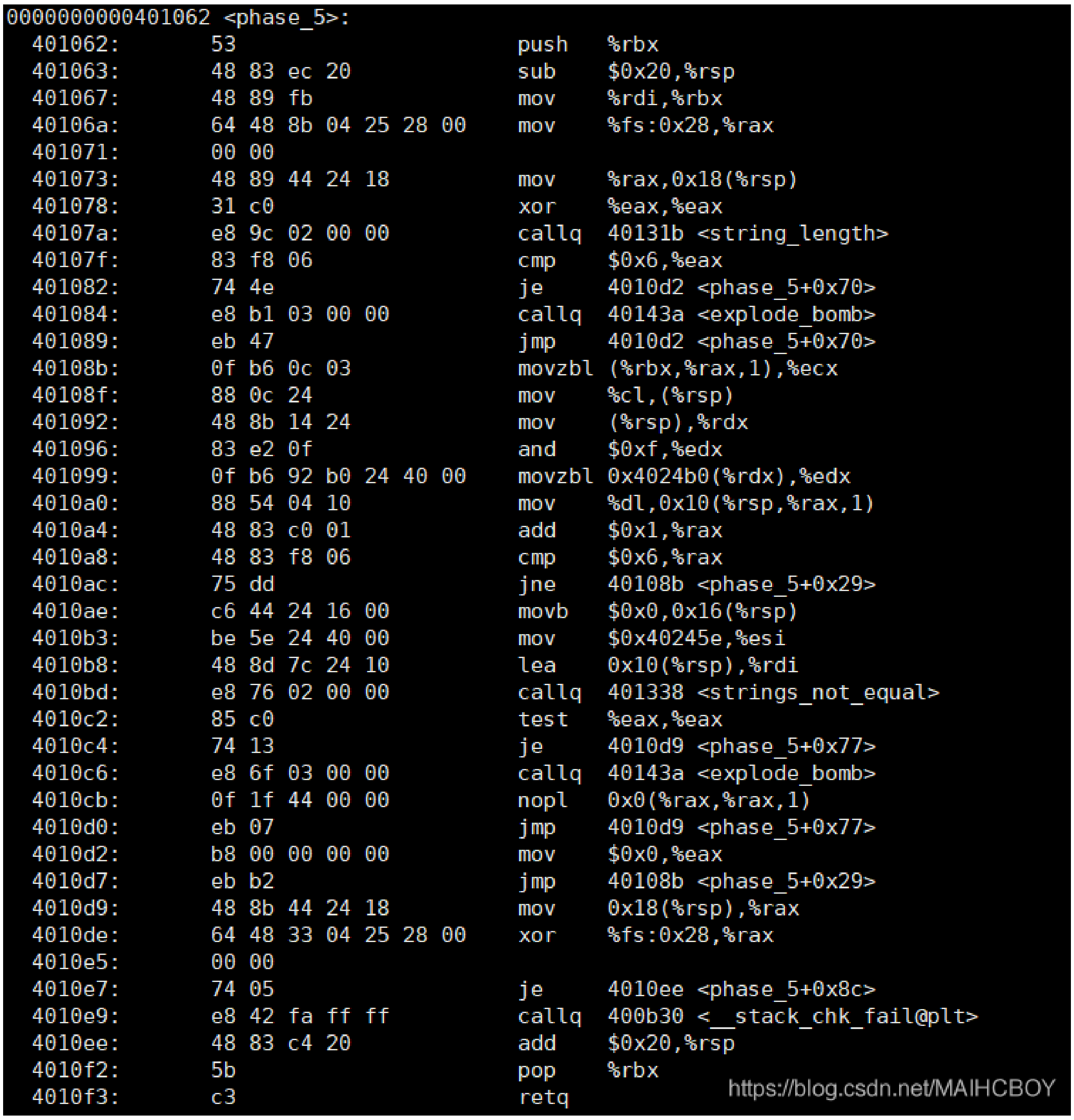
我们输入答案，可以看到进入第五关了！！！



```
[root@iz2ze614mku6kw0goh97edz bomb]# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
0 207
Halfway there!
7 0
So you got that one.  Try this one.
```

5. Level 5

首先，我们分析第五关的反汇编代码：



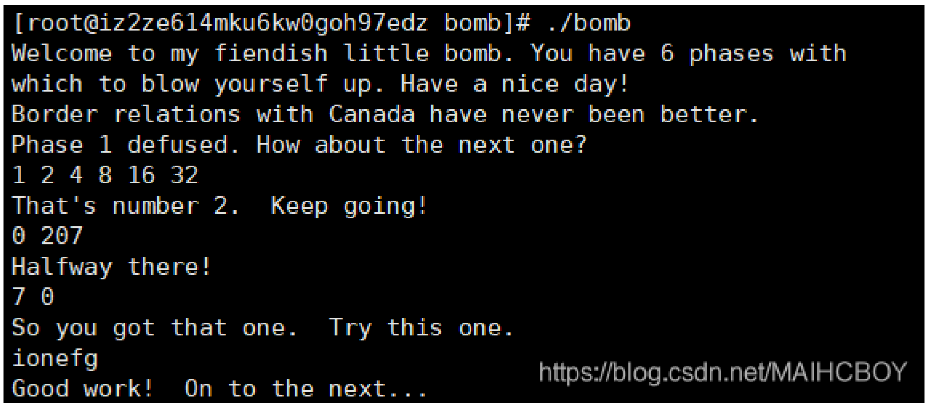
```
0000000000401062 <phase_5>:
  401062:       53                      push   %rbx
  401063:       48 83 ec 20             sub    $0x20,%rsp
  401067:       48 89 fb                mov    %rdi,%rbx
  40106a:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  401071:       00 00
  401073:       48 89 44 24 18          mov    %rax,0x18(%rsp)
  401078:       31 c0                   xor    %eax,%eax
  40107a:       e8 9c 02 00 00          callq  40131b <string_length>
  40107f:       83 f8 06                cmp    $0x6,%eax
  401082:       74 4e                   je     4010d2 <phase_5+0x70>
  401084:       e8 b1 03 00 00          callq  40143a <explode_bomb>
  401089:       eb 47                   jmp    4010d2 <phase_5+0x70>
  40108b:       0f b6 0c 03             movzbl (%rbx,%rax,1),%ecx
  40108f:       88 0c 24                mov    %cl,(%rsp)
  401092:       48 8b 14 24             mov    (%rsp),%rdx
  401096:       83 e2 0f                and    $0xf,%edx
  401099:       0f b6 92 b0 24 40 00    movzbl 0x4024b0(%rdx),%edx
  4010a0:       88 54 04 10             mov    %dl,0x10(%rsp,%rax,1)
  4010a4:       48 83 c0 01             add    $0x1,%rax
  4010a8:       48 83 f8 06             cmp    $0x6,%rax
  4010ac:       75 dd                   jne    40108b <phase_5+0x29>
  4010ae:       c6 44 24 16 00          movb   $0x0,0x16(%rsp)
  4010b3:       be 5e 24 40 00          mov    $0x40245e,%esi
  4010b8:       48 8d 7c 24 10          lea    0x10(%rsp),%rdi
  4010bd:       e8 76 02 00 00          callq  401338 <strings_not_equal>
  4010c2:       85 c0                   test   %eax,%eax
  4010c4:       74 13                   je     4010d9 <phase_5+0x77>
  4010c6:       e8 6f 03 00 00          callq  40143a <explode_bomb>
  4010cb:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
  4010d0:       eb 07                   jmp    4010d9 <phase_5+0x77>
  4010d2:       b8 00 00 00 00          mov    $0x0,%eax
  4010d7:       eb b2                   jmp    40108b <phase_5+0x29>
  4010d9:       48 8b 44 24 18          mov    0x18(%rsp),%rax
  4010de:       64 48 33 04 25 28 00    xor    %fs:0x28,%rax
  4010e5:       00 00
  4010e7:       74 05                   je     4010ee <phase_5+0x8c>
  4010e9:       e8 42 fa ff ff          callq  400b30 <__stack_chk_fail@plt>
  4010ee:       48 83 c4 20             add    $0x20,%rsp
  4010f2:       5b                      pop    %rbx
  4010f3:       c3                      retq
```

401062~40106a：栈帧的初始化
401073：将%rax存入到（%rsp + 0x18）
401078：将%rax置零
40107a：调用string_length函数，将%rdi（我们输入的字符串的首地址）传入

40107f：比较%eax与0x6的大小，若相等，则跳转到4010d2，若不相等，则引爆炸弹，闯关失败。
4010d2：将0x0赋值给%eax
4010d7：无条件跳转到40108b
40108b：使用零拓展数据传送指令，将（%rbx + %rax * 1）的值赋值给%ecx
40108f：将%cl中的值赋值给（%rsp）
401092：将（%rsp）中的值赋值给%rdx
401096：将寄存器 %edx 与 0xf 做与运算的值赋值给 %edx
401099：使用零拓展数据传送指令，将（%rdx + 0x4024b0）的值赋值给%edx
4010a0：将 %dl 中的值赋值给（0x10 + %rsp + %rax * 1）
4010a4：将 %rax 的值+1
4010a8：比较 %rax 的值与0x6，若相等，则继续执行4010ae，若不等，则返回40108b。
4010ae：将0x0赋值给（0x16 + %rsp）
4010b3：将0x40245e赋值给%esi
4010b8：将 0x10 + %rsp 赋值给%rdi
4010bd：调用函数string_not_equal，第一个参数为我们输入的字符串经修改后的地址，第二个参数为正确答案所在的字符串的地址
4010c2：测试%eax是否等于0，若等于0，则跳转到4010d9，即函数正常返回，若不等于0，则引爆炸弹，闯关失败。

通过以上分析，我们可以得出，我们输入的字符串的长度必须为6，并且经过变换之后可以得到目标字符串：“flyers”，经计算，最后得出第五关的答案为：`ionefg`

我们输入答案，可以看到进入第六关了！！！



```
[root@iz2ze614mku6kw0goh97edz bomb]# ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
0 207
Halfway there!
7 0
So you got that one.  Try this one.
ionefg
Good work!  On to the next...
```

https://blog.csdn.net/MAIHCBOY

Level 6

第六关的反汇编代码：

```
00000000004010f4 <phase_6>:
  4010f4: 41 56                   push   %r14
  4010f6: 41 55                   push   %r13
  4010f8: 41 54                   push   %r12
  4010fa: 55                      push   %rbp
  4010fb: 53                      push   %rbx
  4010fc: 48 83 ec 50             sub    $0x50,%rsp
  401100: 49 89 e5                mov    %rsp,%r13
  401103: 48 89 e6                mov    %rsp,%rsi
  401106: e8 51 03 00 00          callq  40145c <read_six_numbers>
  40110b: 49 89 e6                mov    %rsp,%r14
  40110e: 41 bc 00 00 00 00       mov    $0x0,%r12d
  401114: 4c 89 ed                mov    %r13,%rbp
  401117: 41 8b 45 00             mov    0x0(%r13),%eax
  40111b: 83 e8 01                sub    $0x1,%eax
  40111e: 83 f8 05                cmp    $0x5,%eax
  401121: 76 05                   jbe    401128 <phase_6+0x34>
  401123: e8 12 03 00 00          callq  40143a <explode_bomb>
  401128: 41 83 c4 01             add    $0x1,%r12d
  40112c: 41 83 fc 06             cmp    $0x6,%r12d
  401130: 74 21                   je     401153 <phase_6+0x5f>
  401132: 44 89 e3                mov    %r12d,%ebx
  401135: 48 63 c3                movslq %ebx,%rax
  401138: 8b 04 84                mov    (%rsp,%rax,4),%eax
  40113b: 39 45 00                cmp    %eax,0x0(%rbp)
  40113e: 75 05                   jne    401145 <phase_6+0x51>
  401140: e8 f5 02 00 00          callq  40143a <explode_bomb>
  401145: 83 c3 01                add    $0x1,%ebx
  401148: 83 fb 05                cmp    $0x5,%ebx
  40114b: 7e e8                   jle    401135 <phase_6+0x41>
  40114d: 49 83 c5 04             add    $0x4,%r13
  401151: eb c1                   jmp    401114 <phase_6+0x20>
  401153: 48 8d 74 24 18          lea    0x18(%rsp),%rsi
  401158: 4c 89 f0                mov    %r14,%rax
  40115b: b9 07 00 00 00          mov    $0x7,%ecx
  401160: 89 ca                   mov    %ecx,%edx
  401162: 2b 10                   sub    (%rax),%edx
  401164: 89 10                   mov    %edx,(%rax)
  401166: 48 83 c0 04             add    $0x4,%rax
  40116a: 48 39 f0                cmp    %rsi,%rax
  40116d: 75 f1                   jne    401160 <phase_6+0x6c>
  40116f: be 00 00 00 00          mov    $0x0,%esi
  401174: eb 21                   jmp    401197 <phase_6+0xa3>
  401176: 48 8b 52 08             mov    0x8(%rdx),%rdx
  40117a: 83 c0 01                add    $0x1,%eax
  40117d: 39 c8                   cmp    %ecx,%eax
  40117f: 75 f5                   jne    401176 <phase_6+0x82>
  401181: eb 05                   jmp    401188 <phase_6+0x94>
  401183: ba d0 32 60 00          mov    $0x6032d0,%edx
  401188: 48 89 54 74 20          mov    %rdx,0x20(%rsp,%rsi,2)
  40118d: 48 83 c6 04             add    $0x4,%rsi
  401191: 48 83 fe 18             cmp    $0x18,%rsi
  401195: 74 14                   je     4011ab <phase_6+0xb7>
  401197: 8b 0c 34                mov    (%rsp,%rsi,1),%ecx
  40119a: 83 f9 01                cmp    $0x1,%ecx
  40119d: 7e e4                   jle    401183 <phase_6+0x8f>
  40119f: b8 01 00 00 00          mov    $0x1,%eax
  4011a4: ba d0 32 60 00          mov    $0x6032d0,%edx
  4011a9: eb cb                   jmp    401176 <phase_6+0x82>
  4011ab: 48 8b 5c 24 20          mov    0x20(%rsp),%rbx
  4011b0: 48 8d 44 24 28          lea    0x28(%rsp),%rax
  4011b5: 48 8d 74 24 50          lea    0x50(%rsp),%rsi
  4011ba: 48 89 d9                mov    %rbx,%rcx
  4011bd: 48 8b 10                mov    (%rax),%rdx
  4011c0: 48 89 51 08             mov    %rdx,0x8(%rcx)
  4011c4: 48 83 c0 08             add    $0x8,%rax
  4011c8: 48 39 f0                cmp    %rsi,%rax
  4011cb: 74 05                   je     4011d2 <phase_6+0xde>
  4011cd: 48 89 d1                mov    %rdx,%rcx
  4011d0: eb eb                   jmp    4011bd <phase_6+0xc9>
  4011d2: 48 c7 42 08 00 00 00    movq   $0x0,0x8(%rdx)
  4011d9: 00
  4011da: bd 05 00 00 00          mov    $0x5,%ebp
  4011df: 48 8b 43 08             mov    0x8(%rbx),%rax
  4011df: 48 8b 43 08             mov    0x8(%rbx),%rax
  4011e3: 8b 00                   mov    (%rax),%eax
  4011e5: 39 03                   cmp    %eax,(%rbx)
  4011e7: 7d 05                   jge    4011ee <phase_6+0xfa>
  4011e9: e8 4c 02 00 00          callq  40143a <explode_bomb>
  4011ee: 48 8b 5b 08             mov    0x8(%rbx),%rbx
  4011f2: 83 ed 01                sub    $0x1,%ebp
  4011f5: 75 e8                   jne    4011df <phase_6+0xeb>
  4011f7: 48 83 c4 50             add    $0x50,%rsp
  4011fb: 5b                      pop    %rbx
  4011fc: 5d                      pop    %rbp
  4011fd: 41 5c                   pop    %r12
  4011ff: 41 5d                   pop    %r13
  401201: 41 5e                   pop    %r14
  401203: c3                      retq
```

第六关略显复杂，若要一步一步仔细讲解，会花费大量篇幅，而且也不一定能讲明白（因为太过复杂ε(┬┬﹏┬┬)3）。这里直接给出**第六关的答案**：`4 3 2 1 6 5`。小伙伴们如果觉得第六关过于复杂，可以看情况跳过，把前五个弄明白就可以了，当然想挑战自己的小伙伴也可以尝试一下，并不难，只是特麻烦，多花点时间也不是问题。

四、 结尾语

以上就是BombLab的全部内容了。通过这个Lab我们可以更加深入的理解了机器级编程，对计算机底层有了更深的认识。希望本人写的这篇文章对您有所帮助，谢谢！！！