哈爾濱工業大學

# 实验报告

## 实 验（三）

题　　目 Binary Bomb

二进制炸弹

专　　业 计算机系

学　　号 1190201421

班　　级 1936603

学　　生 张瑞

指 导 教 师 刘宏伟

实 验 地 点 G709

实 验 日 期 2021 年 4 月 22 日

## 计算机科学与技术学院

# 目 录

**第 1 章 实验基本信息 ..... - 3 -**

1.1 实验目的 ..... - 3 -
1.2 实验环境与工具 ..... - 3 -
*1.2.1 硬件环境 ..... - 3 -*
*1.2.2 软件环境 ..... - 3 -*
*1.2.3 开发工具 ..... - 3 -*
1.3 实验预习 ..... - 3 -

**第 2 章 实验环境建立 ..... - 5 -**

2.1 UBUNTU 下 CODEBLOCKS 反汇编 ..... - 5 -
2.2 UBUNTU 下 EDB 运行环境建立 ..... - 5 -

**第 3 章 各阶段炸弹破解与分析 ..... - 7 -**

3.1 阶段 1 的破解与分析 ..... - 7 -
3.2 阶段 2 的破解与分析 ..... - 7 -
3.3 阶段 3 的破解与分析 ..... - 9 -
3.4 阶段 4 的破解与分析 ..... - 11 -
3.5 阶段 5 的破解与分析 ..... - 14 -
3.6 阶段 6 的破解与分析 ..... - 16 -
3.7 阶段 7 的破解与分析(隐藏阶段) ..... - 20 -

**第 4 章 总结 ..... - 24 -**

4.1 请总结本次实验的收获 ..... - 24 -
4.2 请给出对本次实验内容的建议 ..... - 24 -

**参考文献 ..... - 25 -**

# 第 1 章 实验基本信息

## 1.1 实验目的

熟练掌握计算机系统的 ISA 指令系统与寻址方式。

熟练掌握 Linux 下调试器的反汇编调试跟踪分析机器语言的方法。

增强对程序机器级表示、汇编语言、调试器和逆向工程等的理解。

## 1.2 实验环境与工具

### 1.2.1 硬件环境

X64 CPU；2GHz；2G RAM；256GHD Disk 以上

### 1.2.2 软件环境

Windows7 64 位以上；VirtualBox/Vmware 11 以上；Ubuntu 16.04 LTS 64 位/优麒麟 64 位

### 1.2.3 开发工具

GDB/OBJDUMP；EDB；KDD 等

## 1.3 实验预习

上实验课前，必须认真预习实验指导书（PPT 或 PDF）。

了解实验的目的、实验环境与软硬件工具、实验操作步骤，复习与实验有关的理论知识。

请写出 C 语言下包含字符串比较、循环、分支（含 switch）、函数调用、递归、指针、结构、链表等的例子程序 sample.c。

生成执行程序 sample.out。

用 gcc –S 或 CodeBlocks 或 GDB 或 OBJDUMP 等，反汇编，比较。

列出每一部分的 C 语言对应的汇编语言。

修改编译选项-O (缺省 2)、O0、O1、O2、O3，-m32/m64。再次查看生成的汇编语言与原来的区别。

注意 O1 之后无栈帧，EBP 做别的用途。-fno-omit-frame-pointer 加上栈指针。

GDB 命令详解 –tui 模式 ^XA 切换 layout 改变等等。

有目的地学习: 看 VS 的功能 GDB 命令用什么？

# 第 2 章 实验环境建立

## 2.1 Ubuntu 下 CodeBlocks 反汇编（10 分）

CodeBlocks 运行 hellolinux.c。反汇编查看 printf 函数的实现。

要求：C、ASM、内存(显示 hello 等内容)、堆栈（call printf 前）、寄存器同时在一个窗口。

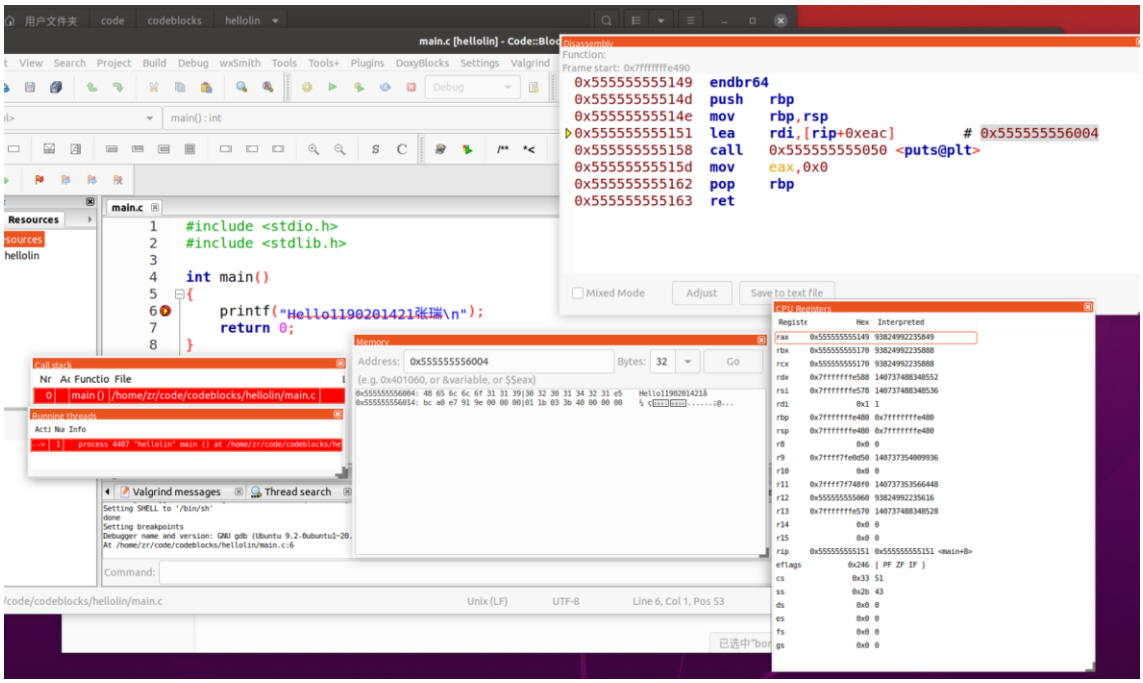


图 2-1 Ubuntu 下 CodeBlocks 反汇编截图

## 2.2 Ubuntu 下 EDB 运行环境建立（10 分）

用 EDB 调试 hellolinux.c 的执行文件，截图，要求同 2.1。

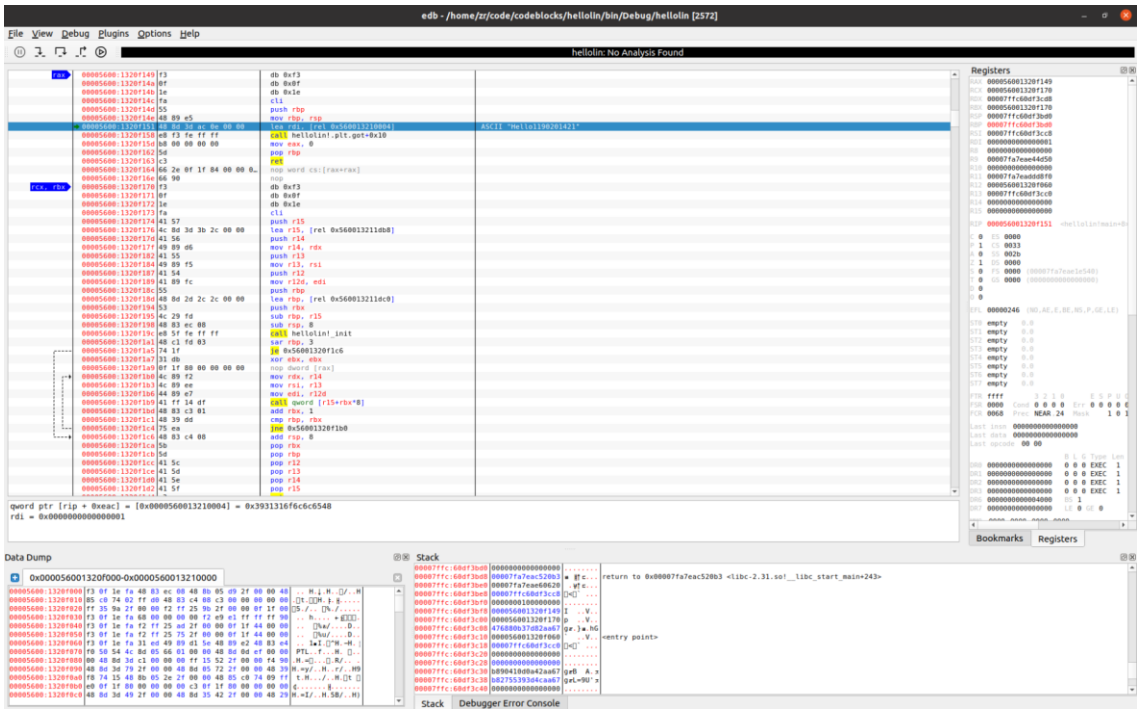


图 2-2　Ubuntu 下 EDB 截图

# 第 3 章 各阶段炸弹破解与分析

每阶段 15 分（密码 10 分，分析 5 分），总分不超过 80 分

## 3.1 阶段 1 的破解与分析

密码如下：Brownie, you are doing a heck of a job.

破解过程：

首先查看 phase_1 的反汇编代码：

```
0000000000401f9 <phase_1>:
  4013f9:       55                      push   %rbp
  4013fa:       48 89 e5                mov    %rsp,%rbp
  4013fd:       be 50 31 40 00          mov    $0x403150,%esi
  401402:       e8 23 04 00 00          callq  40182a <strings_not_equal>
  401407:       85 c0                   test   %eax,%eax
  401409:       75 02                   jne    40140d <phase_1+0x14>
  40140b:       5d                      pop    %rbp
  40140c:       c3                      retq
  40140d:       e8 14 05 00 00          callq  401926 <explode_bomb>
  401412:       eb f7                   jmp    40140b <phase_1+0x12>
```

可以看到，它将 0x403150 压入%esi 之后调用了 stings_not_equal 来比较两个字符串是否相同，只需查看 0x403150 里的内容即可。

用 GDB 查看该地址内容：

```
(gdb) x/s 0x403150
0x403150:       "Brownie, you are doing a heck of a job."
```

将该字符串编辑到 zr_ans.txt，再将该文本作为参数传入即可通过阶段 1：

```
zr@ubuntu:~/桌面/bomb_1190201421/bomb554$ ./bomb zr_ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
```

## 3.2 阶段 2 的破解与分析

密码如下：n n+1 n+3 n+6 n+10 n+15　(n>=0)

破解过程：

首先查看 phase_2 的反汇编代码：

```
0000000000401414 <phase_2>:
  401414:       55                      push   %rbp
  401415:       48 89 e5                mov    %rsp,%rbp
  401418:       53                      push   %rbx
  401419:       48 83 ec 28             sub    $0x28,%rsp
  40141d:       48 8d 75 d0             lea    -0x30(%rbp),%rsi
  401421:       e8 22 05 00 00          callq  401948 <read_six_numbers>
```

一开始开辟了调用函数需要的栈空间，然后调用了 read_six_numbers，猜测应该是要读入 6 个数。

查看 read_six_numbers 的反汇编代码：

```
0000000000401948 <read_six_numbers>:
  401948:       55                      push   %rbp
  401949:       48 89 e5                mov    %rsp,%rbp
  40194c:       48 89 f2                mov    %rsi,%rdx
  40194f:       48 8d 4e 04             lea    0x4(%rsi),%rcx
  401953:       48 8d 46 14             lea    0x14(%rsi),%rax
  401957:       50                      push   %rax
  401958:       48 8d 46 10             lea    0x10(%rsi),%rax
  40195c:       50                      push   %rax
  40195d:       4c 8d 4e 0c             lea    0xc(%rsi),%r9
  401961:       4c 8d 46 08             lea    0x8(%rsi),%r8
  401965:       be 23 33 40 00          mov    $0x403323,%esi
  40196a:       b8 00 00 00 00          mov    $0x0,%eax
  40196f:       e8 9c f7 ff ff          callq  401110 <__isoc99_sscanf@plt>
  401974:       48 83 c4 10             add    $0x10,%rsp
  401978:       83 f8 05                cmp    $0x5,%eax
  40197b:       7e 02                   jle    40197f <read_six_numbers+0x37>
  40197d:       c9                      leaveq
  40197e:       c3                      retq
  40197f:       e8 a2 ff ff ff          callq  401926 <explode_bomb>
```

可以看到，这里用 6 个数调用了 scanf，读入的数据连续存放在堆栈区，并且在末尾进行了一次输入数据量的检测，当输入的数据量小于等于 5 个时，炸弹直接引爆。

继续阅读 phase_2 的反汇编代码：

```
401426:      83 7d d0 00           cmpl   $0x0,-0x30(%rbp)
40142a:      78 07                 js     401433 <phase_2+0x1f>
40142c:      bb 01 00 00 00        mov    $0x1,%ebx
401431:      eb 0f                 jmp    401442 <phase_2+0x2e>
401433:      e8 ee 04 00 00        callq  401926 <explode_bomb>
401438:      eb f2                 jmp    40142c <phase_2+0x18>
40143a:      e8 e7 04 00 00        callq  401926 <explode_bomb>
40143f:      83 c3 01              add    $0x1,%ebx
401442:      83 fb 05              cmp    $0x5,%ebx
401445:      7f 17                 jg     40145e <phase_2+0x4a>
401447:      48 63 c3              movslq %ebx,%rax
40144a:      8d 53 ff              lea    -0x1(%rbx),%edx
40144d:      48 63 d2              movslq %edx,%rdx
401450:      89 d9                 mov    %ebx,%ecx
401452:      03 4c 95 d0           add    -0x30(%rbp,%rdx,4),%ecx
401456:      39 4c 85 d0           cmp    %ecx,-0x30(%rbp,%rax,4)
40145a:      74 e3                 je     40143f <phase_2+0x2b>
40145c:      eb dc                 jmp    40143a <phase_2+0x26>
40145e:      48 83 c4 28           add    $0x28,%rsp
401462:      5b                    pop    %rbx
401463:      5d                    pop    %rbp
401464:      c3                    retq
```

可以看到先是将-0x30(%rbp)与 0 做了比较，如果该值为负数，则炸弹爆炸，猜测此处存放第一个数，要求不为负数，不妨设为 n(n>=0)。后面的代码则构造了一个循环体：%ebx 中存放的数表示循环次数，当次数大于 5 时，退出循环体。每次循环时，将%ebx 符号扩展传送给%rax，再将%rbx 中的值减 1 后给%edx，%edx 再符号扩展给%rdx，再将%ebx 中的值传送给%ecx，然后将%rbp+4*%rdx-0x30 的值加到%ecx 中，（其中%rbp-0x30 为第一个数的地址，%rdx 表示相对于第一个数的偏移的数据个数，4 表示每个数据的大小为 4 个字节），完成加法后，再将得到的值与%rbp+4*%rax-0x30 比较，若相等，则%ebx 的值加 1，继续下一次循环……按此循环规律，可推出输入的 6 个数为 n n+1 n+3 n+6 n+10 n+15　(n>=0)

不妨取 n=1，得到 6 个数 1、2、4、7、11、16，验证得拆弹成功：

```
zr@ubuntu:~/桌面/bomb_1190201421/bomb554$ ./bomb zr_ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
```

## 3.3 阶段 3 的破解与分析

密码如下：0 93 或 1 -200 或 2 -8 或 3 -293 或 4 0 或 5 -293

破解过程：

首先查看 phase_3 的反汇编代码：

```
0000000000401465 <phase_3>:
  401465:       55                      push   %rbp
  401466:       48 89 e5                mov    %rsp,%rbp
  401469:       48 83 ec 10             sub    $0x10,%rsp
  40146d:       48 8d 4d f8             lea    -0x8(%rbp),%rcx
  401471:       48 8d 55 fc             lea    -0x4(%rbp),%rdx
  401475:       be 2f 33 40 00          mov    $0x40332f,%esi
  40147a:       b8 00 00 00 00          mov    $0x0,%eax
  40147f:       e8 8c fc ff ff          callq  401110 <__isoc99_sscanf@plt>
  401484:       83 f8 01                cmp    $0x1,%eax
  401487:       7e 11                   jle    40149a <phase_3+0x35>
```

最后一行跳转到：

```
  40149a:       e8 87 04 00 00          callq  401926 <explode_bomb>
```

即需要读入两个参数，读少了炸弹会直接爆炸。

继续看反汇编代码：

```
  401489:       8b 45 fc                mov    -0x4(%rbp),%eax
  40148c:       83 f8 07                cmp    $0x7,%eax
  40148f:       77 7b                   ja     40150c <phase_3+0xa7>
```

最后一行跳转到：

```
  40150c:       e8 15 04 00 00          callq  401926 <explode_bomb>
```

即第一个参数应该为 0~7 中的一个，否则直接爆炸。

再继续看反汇编代码：

```
  401491:       89 c0                   mov    %eax,%eax
  401493:       ff 24 c5 a0 31 40 00    jmpq   *0x4031a0(,%rax,8)
```

因为此处的跳转涉及到第一个参数的取值，我们不妨先用 0 进行测试。用 GDB 查看 0x4031a0 处的内容：

```
(gdb) x/1x 0x4031a0
0x4031a0:       0x004014db
```

即程序会跳转到地址为 0x004014db 处继续运行，继续查看该处反汇编代码：

```
4014db:      b8 25 01 00 00          mov    $0x125,%eax
4014e0:      eb c4                   jmp    4014a6 <phase_3+0x41>
```

继续跳转：

```
4014a6:      2d c0 00 00 00          sub    $0xc0,%eax
4014ab:      05 1d 01 00 00          add    $0x11d,%eax
4014b0:      2d 25 01 00 00          sub    $0x125,%eax
4014b5:      05 25 01 00 00          add    $0x125,%eax
4014ba:      2d 25 01 00 00          sub    $0x125,%eax
4014bf:      05 25 01 00 00          add    $0x125,%eax
4014c4:      2d 25 01 00 00          sub    $0x125,%eax
4014c9:      83 7d fc 05             cmpl   $0x5,-0x4(%rbp)
4014cd:      7f 05                   jg     4014d4 <phase_3+0x6f>
4014cf:      39 45 f8                cmp    %eax,-0x8(%rbp)
4014d2:      74 05                   je     4014d9 <phase_3+0x74>
4014d4:      e8 4d 04 00 00          callq  401926 <explode_bomb>
4014d9:      c9                      leaveq
4014da:      c3                      retq
```

通过这一段代码，我们能算出，第二个参数为 93。且注意到对第一个参数（此时取为 0）有第二次检验，若大于 5，炸弹也会爆炸（就是说第一个参数的范围实际只能在 0~5 中）。

将参数 0 和 93 输入，验证得拆弹成功：

```
zr@ubuntu:~/桌面/bomb_1190201421/bomb554$ ./bomb zr_ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
```

此外，将第一个参数的取值依次增加，会导致*0x4031a0(,%eax,8)的跳转不同，第二个参数也将变化，同理可求得分别为：1 -200；2 -8；3 -293；4 0；5 -293。

## 3.4 阶段 4 的破解与分析

密码如下：11 1 或 9 1 或 8 1

破解过程：

查看 phase_4 反汇编代码：

```
0000000000401552 <phase_4>:
  401552:       55                      push   %rbp
  401553:       48 89 e5                mov    %rsp,%rbp
  401556:       48 83 ec 10             sub    $0x10,%rsp
  40155a:       48 8d 4d f8             lea    -0x8(%rbp),%rcx
  40155e:       48 8d 55 fc             lea    -0x4(%rbp),%rdx
  401562:       be 2f 33 40 00          mov    $0x40332f,%esi
  401567:       b8 00 00 00 00          mov    $0x0,%eax
  40156c:       e8 9f fb ff ff          callq  401110 <__isoc99_sscanf@plt>
  401571:       83 f8 02                cmp    $0x2,%eax
  401574:       75 0c                   jne    401582 <phase_4+0x30>
```

最后一行跳转到：

```
401582:       e8 9f 03 00 00          callq  401926 <explode_bomb>
```

即需要读入两个参数，否则炸弹将会爆炸。

继续看反汇编代码：

```
401576:       8b 45 fc                mov    -0x4(%rbp),%eax
401579:       85 c0                   test   %eax,%eax
40157b:       78 05                   js     401582 <phase_4+0x30>
40157d:       83 f8 0e                cmp    $0xe,%eax
401580:       7e 05                   jle    401587 <phase_4+0x35>
401582:       e8 9f 03 00 00          callq  401926 <explode_bomb>
401587:       ba 0e 00 00 00          mov    $0xe,%edx
40158c:       be 00 00 00 00          mov    $0x0,%esi
401591:       8b 7d fc                mov    -0x4(%rbp),%edi
401594:       e8 7f ff ff ff          callq  401518 <func4>
```

这里检验第一个参数，若为负数炸弹爆炸，若大于 14 也会爆炸，即初步确定了第一个参数的范围是在 0~14 之间。然后把 14 传送到%edx 中，0 传送到%esi 中，第一个参数传送到%edi 中，再次跳转：

```
0000000000401518 <func4>:
  401518:       55                      push   %rbp
  401519:       48 89 e5                mov    %rsp,%rbp
  40151c:       89 d1                   mov    %edx,%ecx
  40151e:       29 f1                   sub    %esi,%ecx
  401520:       89 c8                   mov    %ecx,%eax
  401522:       c1 e8 1f                shr    $0x1f,%eax
  401525:       01 c8                   add    %ecx,%eax
  401527:       d1 f8                   sar    %eax
  401529:       01 f0                   add    %esi,%eax
  40152b:       39 f8                   cmp    %edi,%eax
  40152d:       7f 09                   jg     401538 <func4+0x20>
  40152f:       7c 13                   jl     401544 <func4+0x2c>
  401531:       b8 00 00 00 00          mov    $0x0,%eax
  401536:       5d                      pop    %rbp
  401537:       c3                      retq
  401538:       8d 50 ff                lea    -0x1(%rax),%edx
  40153b:       e8 d8 ff ff ff          callq  401518 <func4>
  401540:       01 c0                   add    %eax,%eax
  401542:       eb f2                   jmp    401536 <func4+0x1e>
  401544:       8d 70 01                lea    0x1(%rax),%esi
  401547:       e8 cc ff ff ff          callq  401518 <func4>
  40154c:       8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
  401550:       eb e4                   jmp    401536 <func4+0x1e>
```

经过一系列操作得到%ecx 为 14，%eax 为 7，将第一个参数与%eax 比较，由于第一个参数取值在 0~14 之间，则当其值为 7 时，%eax 将被赋为 0，返回；当其值小于 7 时，%rax 的值减 1 传送给%edx，将会发生对 fun_4 自身的调用（出现递归）；当其值大于 7 时，将%rax 的值加 1 传送给%esi，发生对 fun_4 自身的调用（出现递归）……

分析不难得出，其实这段代码做的事情就是通过二分法不断缩小范围，最终确定参数一和 0~14 中哪个值相等。%edx 中存储的是范围的上限，%esi 中储存的是范围的下限。

先继续看 phase_4 的反汇编代码：

```
401599:       83 f8 01                cmp    $0x1,%eax
40159c:       75 06                   jne    4015a4 <phase_4+0x52>
40159e:       83 7d f8 01             cmpl   $0x1,-0x8(%rbp)
4015a2:       74 05                   je     4015a9 <phase_4+0x57>
4015a4:       e8 7d 03 00 00          callq  401926 <explode_bomb>
4015a9:       c9                      leaveq
4015aa:       c3                      retq
```

可以看到若%eax 的值不等于 1，将会爆炸，由此推出，参数一的值只能为 0；参数二也应该为 1，否则爆炸。

下面再逆推参数一的值。从上面可以看出，最后一次递归调用 fun_4 之前，必须修改的是范围的下限，才能让最终的%eax 有等于 1 的可能性，否则，若最后一次递归调用 fun_4 之前，必须修改的是范围的上限，由 0x401540 处易看出，%eax 只能是 0；或者 fun_4 只运行了一次，%eax 直接以 0 返回。以上两种情况均不符合。且 fun_4 被递归调用时，只能修改一次下限且必须在第一次进行修改（之后只能修改上限），否则%eax 的值会在多次递归过程中超出 1。由上述分析容易推出，第一个参数为 11 或 9 或 8。

以 11 和 1 为两个参数，输入验证得成功：

```
zr@ubuntu:~/桌面/bomb_1190201421/bomb554$ ./bomb zr_ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
```

## 3.5 阶段 5 的破解与分析

密码如下：16*n+5(n>=0) 115

破解过程：

先查看 phase_5 的反汇编代码：

```
00000000004015ab <phase_5>:
  4015ab:       55                      push   %rbp
  4015ac:       48 89 e5                mov    %rsp,%rbp
  4015af:       48 83 ec 10             sub    $0x10,%rsp
  4015b3:       48 8d 4d f8             lea    -0x8(%rbp),%rcx
  4015b7:       48 8d 55 fc             lea    -0x4(%rbp),%rdx
  4015bb:       be 2f 33 40 00          mov    $0x40332f,%esi
  4015c0:       b8 00 00 00 00          mov    $0x0,%eax
  4015c5:       e8 46 fb ff ff          callq  401110 <__isoc99_sscanf@plt>
  4015ca:       83 f8 01                cmp    $0x1,%eax
  4015cd:       7e 2e                   jle    4015fd <phase_5+0x52>
```

最后一行跳转到：

```
4015fd:       e8 24 03 00 00          callq  401926 <explode_bomb>
```

即读入两个参数，读不够个数时直接爆炸。

再继续看反汇编代码：

```
4015cf:       8b 45 fc                mov    -0x4(%rbp),%eax
4015d2:       83 e0 0f                and    $0xf,%eax
4015d5:       89 45 fc                mov    %eax,-0x4(%rbp)
4015d8:       b9 00 00 00 00          mov    $0x0,%ecx
4015dd:       ba 00 00 00 00          mov    $0x0,%edx
4015e2:       8b 45 fc                mov    -0x4(%rbp),%eax
4015e5:       83 f8 0f                cmp    $0xf,%eax
4015e8:       74 1a                   je     401604 <phase_5+0x59>
```

最后一行跳转到：

```
401604:       83 fa 0f                cmp    $0xf,%edx
401607:       75 05                   jne    40160e <phase_5+0x63>
401609:       39 4d f8                cmp    %ecx,-0x8(%rbp)
40160c:       74 05                   je     401613 <phase_5+0x68>
40160e:       e8 13 03 00 00          callq  401926 <explode_bomb>
401613:       c9                      leaveq
401614:       c3                      retq
```

可以看到，程序将第一个参数传送给%eax，并只取其最低四位，再将这一值重新传送到参数一处。然后将%ecx 和%edx 均赋为 0。然后会将%eax 中的值与 0xf 进行比较，若相等，会跳转，再对%edx 与 0xf 进行比较，因为此时%edx 值为 0，则两者不等爆炸，故参数一最低四位不能为 0xf。

继续看反汇编代码：

```
4015ea:       83 c2 01                add    $0x1,%edx
4015ed:       48 98                   cltq
4015ef:       8b 04 85 e0 31 40 00    mov    0x4031e0(,%rax,4),%eax
4015f6:       89 45 fc                mov    %eax,-0x4(%rbp)
4015f9:       01 c1                   add    %eax,%ecx
4015fb:       eb e5                   jmp    4015e2 <phase_5+0x37>
```

%edx 被赋值为 1，%eax 的值与其初始取值有关，不妨先查看 0x4031e0 处的内容：

```
(gdb) x/100x 0x4031e0
0x4031e0 <array.3403>:  0x0a    0x00    0x00    0x00    0x02    0x00    0x00    0x00
0x4031e8 <array.3403+8>:        0x0e    0x00    0x00    0x00    0x07    0x00    0x00    0x00
0x4031f0 <array.3403+16>:       0x08    0x00    0x00    0x00    0x0c    0x00    0x00    0x00
0x4031f8 <array.3403+24>:       0x0f    0x00    0x00    0x00    0x0b    0x00    0x00    0x00
0x403200 <array.3403+32>:       0x00    0x00    0x00    0x00    0x04    0x00    0x00    0x00
0x403208 <array.3403+40>:       0x01    0x00    0x00    0x00    0x0d    0x00    0x00    0x00
0x403210 <array.3403+48>:       0x03    0x00    0x00    0x00    0x09    0x00    0x00    0x00
0x403218 <array.3403+56>:       0x06    0x00    0x00    0x00    0x05    0x00    0x00    0x00
0x403220:       0x53    0x6f    0x20    0x79    0x6f    0x75    0x20    0x74
0x403228:       0x68    0x69    0x6e    0x6b    0x20    0x79    0x6f    0x75
0x403230:       0x20    0x63    0x61    0x6e    0x20    0x73    0x74    0x6f
0x403238:       0x70    0x20    0x74    0x68    0x65    0x20    0x62    0x6f
0x403240:       0x6d    0x62    0x20    0x77
```

可以看到，该地址存入了一个数组，分析代码发现程序每次会以%rax 取值为

偏移量查找数组中的一个值，传送给%eax 和参数一，%rdx 作为计数器，计算查找了多少次，%ecx 用来计算所有查找过的值之和。而这一过程将一直持续到查找到%eax 等于 0xf 的时候，检验次数是否刚好是 15 次，第二个参数是否等于%ecx 中的值，否则将会爆炸。

逆向分析得，若要在第 15 次查找到%eax 为 0xf，初始时的%eax 必须为 0x5，最后%ecx 中的值之和为 115。即第一个参数最低四位为 0x5(也就是 16*n+5,n>=0)，第二个参数为 115。

（附上数组中各个数的查找顺序：0x05,0x0c,0x03,0x07,0x0b,0x0d,0x09,0x04,0x08,0x00,0x0a,0x01,0x02,0x0e,0x06,0x0f）

将 5 和 115 输入验证，成功：

```
zr@ubuntu:~/桌面/bomb_1190201421/bomb554$ ./bomb zr_ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
```

## 3.6 阶段 6 的破解与分析

密码如下：2 6 5 4 1 3

破解过程：

首先查看 phase_6 反汇编代码：

```
0000000000401615 <phase_6>:
  401615:       55                      push   %rbp
  401616:       48 89 e5                mov    %rsp,%rbp
  401619:       41 55                   push   %r13
  40161b:       41 54                   push   %r12
  40161d:       53                      push   %rbx
  40161e:       48 83 ec 58             sub    $0x58,%rsp
  401622:       48 8d 75 c0             lea    -0x40(%rbp),%rsi
  401626:       e8 1d 03 00 00          callq  401948 <read_six_numbers>
```

发现和 phase_2 一开始类似，开辟栈空间，读入 6 个数，若读不够直接爆炸。

继续查看反汇编代码：

```
40162b:      41 bc 00 00 00 00      mov    $0x0,%r12d
401631:      eb 29                  jmp    40165c <phase_6+0x47>
401633:      e8 ee 02 00 00         callq  401926 <explode_bomb>
401638:      eb 37                  jmp    401671 <phase_6+0x5c>
40163a:      83 c3 01               add    $0x1,%ebx
40163d:      83 fb 05               cmp    $0x5,%ebx
401640:      7f 17                  jg     401659 <phase_6+0x44>
401642:      49 63 c4               movslq %r12d,%rax
401645:      48 63 d3               movslq %ebx,%rdx
401648:      8b 7c 95 c0            mov    -0x40(%rbp,%rdx,4),%edi
40164c:      39 7c 85 c0            cmp    %edi,-0x40(%rbp,%rax,4)
401650:      75 e8                  jne    40163a <phase_6+0x25>
401652:      e8 cf 02 00 00         callq  401926 <explode_bomb>
401657:      eb e1                  jmp    40163a <phase_6+0x25>
401659:      45 89 ec               mov    %r13d,%r12d
40165c:      41 83 fc 05            cmp    $0x5,%r12d
401660:      7f 19                  jg     40167b <phase_6+0x66>
401662:      49 63 c4               movslq %r12d,%rax
401665:      8b 44 85 c0            mov    -0x40(%rbp,%rax,4),%eax
401669:      83 e8 01               sub    $0x1,%eax
40166c:      83 f8 05               cmp    $0x5,%eax
40166f:      77 c2                  ja     401633 <phase_6+0x1e>
401671:      45 8d 6c 24 01         lea    0x1(%r12),%r13d
401676:      44 89 eb               mov    %r13d,%ebx
401679:      eb c2                  jmp    40163d <phase_6+0x28>
```

这一段完成了一个循环：%r12d 相当于外层循环的计数器，%r13d 中存放了外层计数器的下一个数；%rdx 和%ebx 相当于内层循环的计数器；%edi 用于存储每次取出的参数（即输入的 6 个数之一）；%eax 用于存储各个参数减 1 的值，对各参数的大小做出了限制。这一段循环要求各个参数的取值范围都在 1~6 之间，且任意两参数不能相等，分析易得我们需要按一定顺序输入 1、2、3、4、5、6。

再继续看反汇编代码，确定 1~6 各作为第几个参数：

```
40167b:      be 00 00 00 00         mov    $0x0,%esi
401680:      eb 08                  jmp    40168a <phase_6+0x75>
401682:      48 89 54 cd 90         mov    %rdx,-0x70(%rbp,%rcx,8)
401687:      83 c6 01               add    $0x1,%esi
40168a:      83 fe 05               cmp    $0x5,%esi
40168d:      7f 1c                  jg     4016ab <phase_6+0x96>
40168f:      b8 01 00 00 00         mov    $0x1,%eax
401694:      ba d0 52 40 00         mov    $0x4052d0,%edx
401699:      48 63 ce               movslq %esi,%rcx
40169c:      39 44 8d c0            cmp    %eax,-0x40(%rbp,%rcx,4)
4016a0:      7e e0                  jle    401682 <phase_6+0x6d>
4016a2:      48 8b 52 08            mov    0x8(%rdx),%rdx
4016a6:      83 c0 01               add    $0x1,%eax
4016a9:      eb ee                  jmp    401699 <phase_6+0x84>
```

这一段通过循环将输入的参数进行了预处理：%esi 相当于循环的计数器；%eax 中存放 1~6 当中的一个数，用于和输入参数进行比较；%edx 中根据输入参数的大小依次进行调整——以 0x4052d0 为基值，每次以 8 为偏移量单位进行计算，算出的值被作为内存中的地址，再将其对应内容重新传入%edx，将这一过程重复 i 次（i 为当前参数大小），最后将得到的 6 个新的值按参数读入的顺序存放于一段相邻内存。

再继续看反汇编代码：

```
4016ab:      48 8b 5d 90             mov    -0x70(%rbp),%rbx
4016af:      48 89 d9                mov    %rbx,%rcx
4016b2:      b8 01 00 00 00          mov    $0x1,%eax
4016b7:      eb 12                   jmp    4016cb <phase_6+0xb6>
4016b9:      48 63 d0                movslq %eax,%rdx
4016bc:      48 8b 54 d5 90          mov    -0x70(%rbp,%rdx,8),%rdx
4016c1:      48 89 51 08             mov    %rdx,0x8(%rcx)
4016c5:      83 c0 01                add    $0x1,%eax
4016c8:      48 89 d1                mov    %rdx,%rcx
4016cb:      83 f8 05                cmp    $0x5,%eax
4016ce:      7e e9                   jle    4016b9 <phase_6+0xa4>
```

这一段通过循环，将上述预处理所得到的新的 6 个值中的后 5 个传送给新的一段内存里。（后来验证发现此处其实是按输入参数的顺序建立各节点指针的新的链接）

再继续看反汇编代码：

```
4016d0:      48 c7 41 08 00 00 00    movq   $0x0,0x8(%rcx)
4016d7:      00
4016d8:      41 bc 00 00 00 00       mov    $0x0,%r12d
4016de:      eb 08                   jmp    4016e8 <phase_6+0xd3>
4016e0:      48 8b 5b 08             mov    0x8(%rbx),%rbx
4016e4:      41 83 c4 01             add    $0x1,%r12d
4016e8:      41 83 fc 04             cmp    $0x4,%r12d
4016ec:      7f 11                   jg     4016ff <phase_6+0xea>
4016ee:      48 8b 43 08             mov    0x8(%rbx),%rax
4016f2:      8b 00                   mov    (%rax),%eax
4016f4:      39 03                   cmp    %eax,(%rbx)
4016f6:      7e e8                   jle    4016e0 <phase_6+0xcb>
4016f8:      e8 29 02 00 00          callq  401926 <explode_bomb>
4016fd:      eb e1                   jmp    4016e0 <phase_6+0xcb>
4016ff:      48 83 c4 58             add    $0x58,%rsp
401703:      5b                      pop    %rbx
401704:      41 5c                   pop    %r12
401706:      41 5d                   pop    %r13
401708:      5d                      pop    %rbp
401709:      c3                      retq
```

在上述内存的末尾，再加上同样空间大小的立即数 0。后续的操作即将新的 6 个值的第一个与上一步所存入连续内存的 5 个值，看做地址，取其对应地址处的内容进行大小比较，要求按新参数的顺序排列时，前者小于后者。到这里我们已经明白了，我们输入的 6 个参数应该是对 0x4052d0 处的一段内容进行大小排序，要求从小到大进行排列。

现在查看 0x4052d0 处的内容：

```
(gdb) x/100x 0x4052d0
0x4052d0 <node1>:       0x0000033f      0x00000001      0x004052e0      0x00000000
0x4052e0 <node2>:       0x00000201      0x00000002      0x004052f0      0x00000000
0x4052f0 <node3>:       0x00000374      0x00000003      0x00405300      0x00000000
0x405300 <node4>:       0x0000029c      0x00000004      0x00405310      0x00000000
0x405310 <node5>:       0x0000026a      0x00000005      0x00405320      0x00000000
0x405320 <node6>:       0x00000221      0x00000006      0x00000000      0x00000000
0x405330 <bomb_id>:     0x0000022a      0x00000000      0x00000000      0x00000000
0x405340 <host_table>:  0x00403389      0x00000000      0x004033a3      0x00000000
0x405350 <host_table+16>:       0x004033bd      0x00000000      0x00000000      0x00000000
0x405360 <host_table+32>:       0x00000000      0x00000000      0x00000000      0x00000000
0x405370 <host_table+48>:       0x00000000      0x00000000      0x00000000      0x00000000
0x405380 <host_table+64>:       0x00000000      0x00000000      0x00000000      --Type--T--Ty----
```

可以看到，此处存在一个链表，刚好 6 个节点，按从小到大的顺序排列为 2 6 5 4 1 3。

将 2 6 5 4 1 3 作为答案输入，验证得成功：

```
zr@ubuntu:~/桌面/bomb_1190201421/bomb554$ ./bomb zr_ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Congratulations! You've defused the bomb!
```

## 3.7 阶段 7 的破解与分析(隐藏阶段)

密码如下：107（需要在 phase_4 的答案后面加上字符串 DrEvil）

破解过程：

首先我们发现在前面的各个阶段的反汇编代码中都没有出现调用 secret_phase 的地方，而查看 bomb.c 发现，main 函数里除了调用各个 phase，还有 initialize_bomb 和 phase_defused，再查看这两段代码，发现 phase_defused 中出现了对 secret_phase 的调用：

```
0000000000401aaf <phase_defused>:
  401aaf:       83 3d b6 3c 00 00 06    cmpl   $0x6,0x3cb6(%rip)        # 40576c <num_input_strings>
  401ab6:       74 01                   je     401ab9 <phase_defused+0xa>
  401ab8:       c3                      retq
  401ab9:       55                      push   %rbp
  401aba:       48 89 e5                mov    %rsp,%rbp
  401abd:       48 83 ec 60             sub    $0x60,%rsp
  401ac1:       4c 8d 45 b0             lea    -0x50(%rbp),%r8
  401ac5:       48 8d 4d a8             lea    -0x58(%rbp),%rcx
  401ac9:       48 8d 55 ac             lea    -0x54(%rbp),%rdx
  401acd:       be 79 33 40 00          mov    $0x403379,%esi
  401ad2:       bf 70 58 40 00          mov    $0x405870,%edi
  401ad7:       b8 00 00 00 00          mov    $0x0,%eax
  401adc:       e8 2f f6 ff ff          callq  401110 <__isoc99_sscanf@plt>
  401ae1:       83 f8 03                cmp    $0x3,%eax
  401ae4:       74 0c                   je     401af2 <phase_defused+0x43>
  401ae6:       bf b8 32 40 00          mov    $0x4032b8,%edi
  401aeb:       e8 70 f5 ff ff          callq  401060 <puts@plt>
  401af0:       c9                      leaveq
  401af1:       c3                      retq
  401af2:       be 82 33 40 00          mov    $0x403382,%esi
  401af7:       48 8d 7d b0             lea    -0x50(%rbp),%rdi
  401afb:       e8 2a fd ff ff          callq  40182a <strings_not_equal>
  401b00:       85 c0                   test   %eax,%eax
  401b02:       75 e2                   jne    401ae6 <phase_defused+0x37>
  401b04:       bf 58 32 40 00          mov    $0x403258,%edi
  401b09:       e8 52 f5 ff ff          callq  401060 <puts@plt>
  401b0e:       bf 80 32 40 00          mov    $0x403280,%edi
  401b13:       e8 48 f5 ff ff          callq  401060 <puts@plt>
  401b18:       b8 00 00 00 00          mov    $0x0,%eax
  401b1d:       e8 22 fc ff ff          callq  401744 <secret_phase>
  401b22:       eb c2                   jmp    401ae6 <phase_defused+0x37>
```

这一段代码表示要先检验是否读入了 6 个字符串（即 6 个阶段对应的 6 个答案），否则不会调用 secret_phase。当已经读入 6 个字符串的时候，验证前面某一地方的字符串（实际就是 phase_4 的答案对应字符串）是否是读入了 3 个参

数，否则也不会调用 secret_phase。读入格式如下：

```
(gdb) x/s 0x403379
0x403379:       "%d %d %s"
```

再结合实验给的提示，我们可以确定，要在 phase_4 的答案后面加上一个字符串才能激活隐藏阶段，若输入的字符串与给定的不同，还是不会调用 secret_phase，先查看给定字符串内容：

```
(gdb) x/s 0x403382
0x403382:       "DrEvil"
```

故我们需在 phase_4 的答案后面加上字符串 DrEvil，验证发现激活了隐藏阶段：

```
zr@ubuntu:~/桌面/bomb_1190201421/bomb554$ ./bomb zr_ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
```

现在查看 secret_phase 的反汇编代码：

```
0000000000401744 <secret_phase>:
  401744:       55                      push   %rbp
  401745:       48 89 e5                mov    %rsp,%rbp
  401748:       53                      push   %rbx
  401749:       48 83 ec 08             sub    $0x8,%rsp
  40174d:       e8 32 02 00 00          callq  401984 <read_line>
  401752:       48 89 c7                mov    %rax,%rdi
  401755:       e8 e6 f9 ff ff          callq  401140 <atoi@plt>
  40175a:       89 c3                   mov    %eax,%ebx
  40175c:       8d 40 ff                lea    -0x1(%rax),%eax
  40175f:       3d e8 03 00 00          cmp    $0x3e8,%eax
  401764:       77 27                   ja     40178d <secret_phase+0x49>
  401766:       89 de                   mov    %ebx,%esi
  401768:       bf f0 50 40 00          mov    $0x4050f0,%edi
  40176d:       e8 98 ff ff ff          callq  40170a <fun7>
  401772:       83 f8 03                cmp    $0x3,%eax
  401775:       75 1d                   jne    401794 <secret_phase+0x50>
  401777:       bf 78 31 40 00          mov    $0x403178,%edi
  40177c:       e8 df f8 ff ff          callq  401060 <puts@plt>
  401781:       e8 29 03 00 00          callq  401aaf <phase_defused>
  401786:       48 83 c4 08             add    $0x8,%rsp
  40178a:       5b                      pop    %rbx
  40178b:       5d                      pop    %rbp
  40178c:       c3                      retq
  40178d:       e8 94 01 00 00          callq  401926 <explode_bomb>
  401792:       eb d2                   jmp    401766 <secret_phase+0x22>
  401794:       e8 8d 01 00 00          callq  401926 <explode_bomb>
  401799:       eb dc                   jmp    401777 <secret_phase+0x33>
```

发现这里要求读入的字符串在转化为整型数之后范围在 0x1~0x3e9 之间，之后又出现了对 fun_7 的调用，要求返回值刚好等于 3，然后就能破解该阶段，否则就爆炸。再查看 fun_7 的反汇编代码：

```
000000000040170a <fun7>:
  40170a:       48 85 ff                test   %rdi,%rdi
  40170d:       74 2f                   je     40173e <fun7+0x34>
  40170f:       55                      push   %rbp
  401710:       48 89 e5                mov    %rsp,%rbp
  401713:       8b 07                   mov    (%rdi),%eax
  401715:       39 f0                   cmp    %esi,%eax
  401717:       7f 09                   jg     401722 <fun7+0x18>
  401719:       75 14                   jne    40172f <fun7+0x25>
  40171b:       b8 00 00 00 00          mov    $0x0,%eax
  401720:       5d                      pop    %rbp
  401721:       c3                      retq
  401722:       48 8b 7f 08             mov    0x8(%rdi),%rdi
  401726:       e8 df ff ff ff          callq  40170a <fun7>
  40172b:       01 c0                   add    %eax,%eax
  40172d:       eb f1                   jmp    401720 <fun7+0x16>
  40172f:       48 8b 7f 10             mov    0x10(%rdi),%rdi
  401733:       e8 d2 ff ff ff          callq  40170a <fun7>
  401738:       8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
  40173c:       eb e2                   jmp    401720 <fun7+0x16>
  40173e:       b8 ff ff ff ff          mov    $0xffffffff,%eax
  401743:       c3                      retq
```

分析这段代码，发现实际是有一棵二叉树，当我们输入的字符串转化为数之后，把这个数依次与二叉树的节点比较，若小于节点，则继续与左节点比较，返回（2*%eax）；若大于节点，则继续与右节点比较，返回（2*%eax+0x1）；若相等，则停止比较，返回 0。分析得，我们可以取两次与右节点的比较，则能使返回值为 3。连续查看两次右节点地址，并查看最后一次所取节点的大小：

```
(gdb) x/8x 0x405100
0x405100 <n1+16>:       0x30    0x51    0x40    0x00    0x00    0x00    0x00    0x00
(gdb) x/8x 0x405140
0x405140 <n22+16>:      0xb0    0x51    0x40    0x00    0x00    0x00    0x00    0x00
(gdb) x/8x 0x4051b0
0x4051b0 <n34>: 0x6b    0x00    0x00    0x00    0x00    0x00    0x00    0x00
```

则应输入 0x6b，即 107，验证得成功，完成所有阶段的拆弹：

```
zr@ubuntu:~/桌面/bomb_1190201421/bomb554$ ./bomb zr_ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```

# 第 4 章 总结

## 4.1 请总结本次实验的收获

本次实验中，我通过对汇编代码的多次查看，对教材的反复翻阅，巩固了汇编的相关知识，深入理解了一些机器级代码指令与操作，对寄存器与内存的作用更加熟悉了，同时也第一次切身感受了逆向工程，解密形式的实验也让我感受到了乐趣，拆弹成功后的喜悦感和成就感极强。

## 4.2 请给出对本次实验内容的建议

本次实验趣味性强，内容丰富，考察了有关汇编的多方面的知识，对能力提升有很大的帮助，可以多设置这样的实验。

注：本章为酌情加分项。

# 参考文献

**为完成本次实验你翻阅的书籍与网站等**

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A 集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/collection/anatmorp.