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　计算机类

学　　号　1170300821

班　　级　1703008

学　　生　罗瑞欣

指 导 教 师　郑贵滨

实 验 地 点　G712

实 验 日 期　2018.10.15

计算机科学与技术学院

目 录

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

Visual Studio 2010 64 位以上；GDB/OBJDUMP；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。

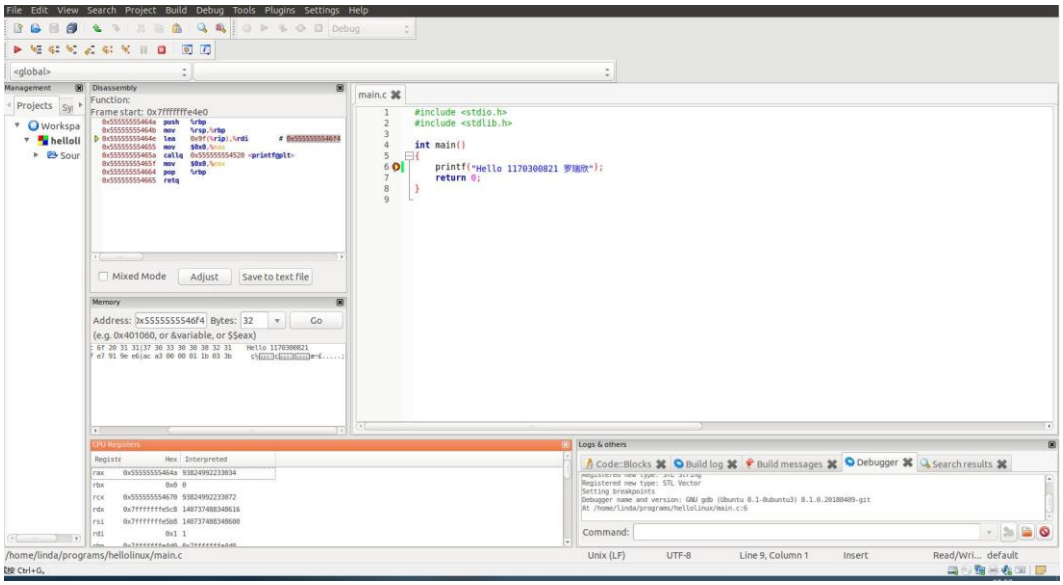


图 2-1　Ubuntu 下 CodeBlocks 反汇编截图

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

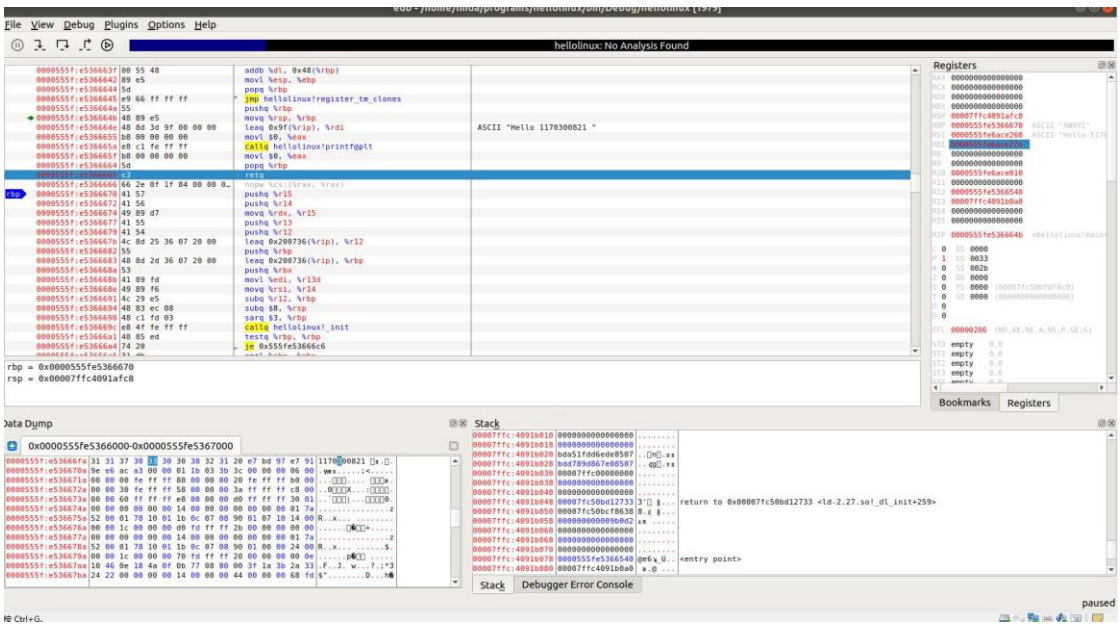


图 2-2　Ubuntu 下 EDB 截图

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：

破解过程：

（1）观察到调用的函数 string_not_equal，判断地址 0x4023c0 处的和输入是否相等。

```
0000000000400e8d <phase_1>:
  400e8d:       48 83 ec 08             sub    $0x8,%rsp
  400e91:       be c0 23 40 00          mov    $0x4023c0,%esi
  400e96:       e8 9c 04 00 00          callq  401337 <strings_not_equal>
  400e9b:       85 c0                   test   %eax,%eax
```

（2）用 gdb 查看地址 0x4023c0

```
(gdb) x/2s 0x4023c0
0x4023c0:       "Border relations with Canada have never been better."
```

得到答案"Border relations with Canada have never been better."

3.2 阶段 2 的破解与分析

密码如下：0 1 3 6 10 15

破解过程：

（1）phase_2 在初始化栈后，将 rsp 赋初值，观察到函数 phase_2 调用了函数 read_six_numbers

```
400eb8:       48 89 44 24 18          mov    %rax,0x18(%rsp)
400ebd:       31 c0                   xor    %eax,%eax
400ebf:       48 89 e6                mov    %rsp,%rsi
400ec2:       e8 91 05 00 00          callq  401458 <read_six_numbers>
```

（2）在函数 read_six_numbers 中利用%rsp 偏移定义了六个变量，并出现语句

mov $0x402583,%esi 并调用函数__isoc99_sscanf@plt，调用内存 0x402583，发现是%d %d %d %d %d %d，得知密码为六个整型数字。

```
(gdb) x/2s 0x402583
0x402583:       "%d %d %d %d %d %d"
```

```
401475:       be 83 25 40 00          mov    $0x402583,%esi
40147a:       b8 00 00 00 00          mov    $0x0,%eax
40147f:       e8 2c f7 ff ff          callq  400bb0 <__isoc99_sscanf@plt>
```

（3）回到函数 phase_2，检验%rsp 是否为有效地址，否则引爆

```
400ec7:       83 3c 24 00             cmpl   $0x0,(%rsp)
400ecb:       79 05                   jns    400ed2 <phase_2+0x29>
```

开始进行计算部分，先改用%rbp 地址偏移，将%ebx 置 1，把%ebx 的值赋给%eax，再把（%rbp）的值传给%eax，最后比较%eax 和（%rbp）偏移四个的值，若若相等则到 400ee9 执行，否则引爆。

```
400ed2:       48 89 e5                mov    %rsp,%rbp
400ed5:       bb 01 00 00 00          mov    $0x1,%ebx
400eda:       89 d8                   mov    %ebx,%eax
400edc:       03 45 00                add    0x0(%rbp),%eax
400edf:       39 45 04                cmp    %eax,0x4(%rbp)
400ee2:       74 05                   je     400ee9 <phase_2+0x40>
400ee4:       e8 4d 05 00 00          callq  401436 <explode_bomb>
```

（4）接下来是循环判断部分，将%edx 加 1，在将%rbp 加 4，比较%ebx 和 6，不相等则返回 400eda 重复执行计算部分。

```
400ee9:       83 c3 01                add    $0x1,%ebx
400eec:       48 83 c5 04             add    $0x4,%rbp
400ef0:       83 fb 06                cmp    $0x6,%ebx
400ef3:       75 e5                   jne    400eda <phase_2+0x31>
400ef5:       48 8b 44 24 18          mov    0x18(%rsp),%rax
```

（5）分析可知 phase_2 为 sum[n+1]=sum[n]+n,答案为 sum[0]到 sum[5]六个数。

即 0 1 3 6 10 15

3.3 阶段 3 的破解与分析

密码如下：0 61 或 1 -818 或 2 38 或 3 -616 或 4 0 或 5 -616

破解过程：

（1）参数初始化

```
400f1e:       48 89 44 24 08          mov    %rax,0x8(%rsp)
400f23:       31 c0                   xor    %eax,%eax
400f25:       48 8d 4c 24 04          lea    0x4(%rsp),%rcx
400f2a:       48 89 e2                mov    %rsp,%rdx
```

（2）查看地址 0x40258f，是两个%d，可知密码是两个数字

```
400f2d:       be 8f 25 40 00          mov    $0x40258f,%esi
400f32:       e8 79 fc ff ff          callq  400bb0 <__isoc99_sscanf@plt>
```

（3） 接下来是两个检查，要求%eax=0，（%rsp）<=7；

```
400f37:       83 f8 01                cmp    $0x1,%eax
400f3a:       7f 05                   jg     400f41 <phase_3+0x30>
400f3c:       e8 f5 04 00 00          callq  401436 <explode_bomb>
400f41:       83 3c 24 07             cmpl   $0x7,(%rsp)
400f45:       77 65                   ja     400fac <phase_3+0x9b>
```

（4）接下来是类似于 switch 的语句

```
400f47:       8b 04 24                mov    (%rsp),%eax
400f4a:       ff 24 c5 30 24 40 00    jmpq   *0x402430(,%rax,8)
```

跳到地址为 0x402430+(%rax+8)的语句，并观察下面七个分支，没有类似 break 的跳转

```
400f51:       b8 6f 03 00 00          mov    $0x36f,%eax
400f56:       eb 05                   jmp    400f5d <phase_3+0x4c>
400f58:       b8 00 00 00 00          mov    $0x0,%eax
400f5d:       2d 58 03 00 00          sub    $0x358,%eax
400f62:       eb 05                   jmp    400f69 <phase_3+0x58>
400f64:       b8 00 00 00 00          mov    $0x0,%eax
400f69:       05 8e 02 00 00          add    $0x28e,%eax
400f6e:       eb 05                   jmp    400f75 <phase_3+0x64>
```

由此推测，跳转到任意分支后，会将下面剩余分支依次执行。

（5）最后的判断条件，可知（%rsp）不能大于 5，最后结果（%eax）应等于 0x4（%rsp）。

```
400fb6:       83 3c 24 05             cmpl   $0x5,(%rsp)
400fba:       7f 06                   jg     400fc2 <phase_3+0xb1>
400fbc:       3b 44 24 04             cmp    0x4(%rsp),%eax
400fc0:       74 05                   je     400fc7 <phase_3+0xb6>
400fc2:       e8 6f 04 00 00          callq  401436 <explode_bomb>
```

（6）综上分析可知，答案共有六组数，第一个数字是序号，决定跳过前 n 个开始依次执行，但 n 不能大于 5；第二个数是 switch 语句执行出的结果。

3.4 阶段 4 的破解与分析

密码如下：162 3 或 108 2

破解过程：

（1）参数初始化

```
401029:       48 89 44 24 08          mov    %rax,0x8(%rsp)
40102e:       31 c0                   xor    %eax,%eax
401030:       48 89 e1                mov    %rsp,%rcx
401033:       48 8d 54 24 04          lea    0x4(%rsp),%rdx
```

（2）查看内存 0x40258f，是两个%d，可知密码为两个数字

```
401038:       be 8f 25 40 00          mov    $0x40258f,%esi
40103d:       e8 6e fb ff ff          callq  400bb0 <__isoc99_sscanf@plt>
```

```
(gdb) x/2s 0x40258f
0x40258f:       "%d %d"
```

（3）%eax 先与 2 比较，再把（%rsp）赋值给%eax 并减 2 与 2 比较，可知第二个输入的数小于 4 大于等于 2，故第二个数可以输入 2 或 3

```
401042:       83 f8 02                cmp    $0x2,%eax
401045:       75 0b                   jne    401052 <phase_4+0x36>
401047:       8b 04 24                mov    (%rsp),%eax
40104a:       83 e8 02                sub    $0x2,%eax
40104d:       83 f8 02                cmp    $0x2,%eax
401050:       76 05                   jbe    401057 <phase_4+0x3b>
401052:       e8 df 03 00 00          callq  401436 <explode_bomb>
```

（4）接下来两次赋值，调用函数 func4，观察可知 func4 内两次调用自身，为递归函数，而且由%edi 控制递归次数

```
401057:       8b 34 24                mov    (%rsp),%esi
40105a:       bf 08 00 00 00          mov    $0x8,%edi
40105f:       e8 7d ff ff ff          callq  400fe1 <func4>
```

（5）观察 func4 内部，当%edi=0 跳转 401010，返回%eax=0，当%edi=1 时，跳转 40101a，返回%eax=（%rsp）。此部分为递归基础。

```
400fe1:       85 ff                    test   %edi,%edi
400fe3:       7e 2b                    jle    401010 <func4+0x2f>
400fe5:       89 f0                    mov    %esi,%eax
400fe7:       83 ff 01                 cmp    $0x1,%edi
400fea:       74 2e                    je     40101a <func4+0x39>

401010:       b8 00 00 00 00           mov    $0x0,%eax
401015:       c3                       retq
401016:       5b                       pop    %rbx
401017:       5d                       pop    %rbp
401018:       41 5c                    pop    %r12
40101a:       f3 c3                    repz retq
```

（6）%esi 赋给%ebp，%edi 赋给%ebx，%edi 减 1 后调用 func4 相当于 func4（n-1），此处为第一个递归的方向。

```
400ff0:       89 f5                    mov    %esi,%ebp
400ff2:       89 fb                    mov    %edi,%ebx
400ff4:       8d 7f ff                 lea    -0x1(%rdi),%edi
400ff7:       e8 e5 ff ff ff           callq  400fe1 <func4>
```

（7）%r12d=%rbp+%rax=%esi+%edi，%edi 减 2 后调用 func4，相当于 func4（n-2），此处为第二个递归方向。

```
400ffc:       44 8d 64 05 00           lea    0x0(%rbp,%rax,1),%r12d
401001:       8d 7b fe                 lea    -0x2(%rbx),%edi
401004:       89 ee                    mov    %ebp,%esi
401006:       e8 d6 ff ff ff           callq  400fe1 <func4>
```

（8）最后%eax 加上%r12d 跳转 401016，返回%eax，即返回%esi+%edi。

```
40100b:       44 01 e0                 add    %r12d,%eax
40100e:       eb 06                    jmp    401016 <func4+0x35>
```

（9）返回 phase_4，检查 0x4（%rsp）是否等于%eax，若相等则结束函数。

```
401064:       3b 44 24 04              cmp    0x4(%rsp),%eax
401068:       74 05                    je     40106f <phase_4+0x53>
40106a:       e8 c7 03 00 00           callq  401436 <explode_bomb>
40106f:       48 8b 44 24 08           mov    0x8(%rsp),%rax
401074:       64 48 33 04 25 28 00     xor    %fs:0x28,%rax
```

（10）利用 C 语言模拟 func4，得出当第二个输入的数，即%edi=3 时，返回值为 162；当%edi=2 时，返回值为 108。

3.5 阶段 5 的破解与分析

密码如下：yoapeg（每个字符大小写均可）

```
So you got that one.  Try this one.
yoapeg
Good work!  On to the next...
```

破解过程：

（1）参数初始化，把%rax 的值赋给 0x8（%rsp），将%eax 置 0。

```
40109a:       48 89 44 24 08          mov    %rax,0x8(%rsp)
40109f:       31 c0                   xor    %eax,%eax
4010a1:       e8 73 02 00 00          callq  401319 <string_length>
```

（2）查看函数 string_length，功能是将（%rdi）置 0，同时%eax 不断累加，记录了%rdi 指向字符串的长度。

```
401323:       48 83 c7 01             add    $0x1,%rdi
401327:       83 c0 01                add    $0x1,%eax
40132a:       80 3f 00                cmpb   $0x0,(%rdi)
```

判断%eax 是否等于 6，可知密码是一个六位的字符串。

```
4010a6:       83 f8 06                cmp    $0x6,%eax
```

（2.5）尝试用 gdb 查看内存 0x402470，结果发现了隐藏关卡的入口提示信息。。。。。。。。

```
(gdb) x/2s 0x402470
0x402470 <array.3597>:  "maduiersnfotvbylSo you think you can stop the bomb with
 ctrl-c, do you?"
0x4024b8:       "Curses, you've found the secret phase!"
(gdb)
```

（3）注意到地址 0x402470 后十六个字符 maduiersnfotvby，分别对应了 16 个地址，可能后面会用到的对应表。

```
0x402470 <array.3597>:  109 'm'
(gdb)
0x402471 <array.3597+1>:        97 'a'
(gdb)
0x402472 <array.3597+2>:        100 'd'
(gdb)
0x402473 <array.3597+3>:        117 'u'
(gdb)
0x402474 <array.3597+4>:        105 'i'
(gdb)
0x402475 <array.3597+5>:        101 'e'
(gdb)
0x402476 <array.3597+6>:        114 'r'
(gdb)
0x402477 <array.3597+7>:        115 's'
(gdb)
```

```
0x402478 <array.3597+8>:        110 'n'
(gdb)
0x402479 <array.3597+9>:        102 'f'
(gdb)
0x40247a <array.3597+10>:       111 'o'
(gdb)
0x40247b <array.3597+11>:       116 't'
(gdb)
0x40247c <array.3597+12>:       118 'v'
(gdb)
0x40247d <array.3597+13>:       98 'b'
(gdb)
0x40247e <array.3597+14>:       121 'y'
(gdb)
0x40247f <array.3597+15>:       108 'l'
(gdb)
```

（4）接下来是一个循环，将输入六个字符的十六进制的最后一位，加上0x402470，作为指针并将指向的字符传入(%rsp,%rax,1)中。

```
4010b0:       b8 00 00 00 00          mov    $0x0,%eax
4010b5:       0f b6 14 03             movzbl (%rbx,%rax,1),%edx
4010b9:       83 e2 0f                and    $0xf,%edx
4010bc:       0f b6 92 70 24 40 00    movzbl 0x402470(%rdx),%edx
4010c3:       88 14 04                mov    %dl,(%rsp,%rax,1)
4010c6:       48 83 c0 01             add    $0x1,%rax
4010ca:       48 83 f8 06             cmp    $0x6,%rax
4010ce:       75 e5                   jne    4010b5 <phase_5+0x2c>
```

（5）用 gdb 查看地址 0x40241e，发现是 flames。

```
4010d0:       c6 44 24 06 00          movb   $0x0,0x6(%rsp)
4010d5:       be 1e 24 40 00          mov    $0x40241e,%esi
4010da:       48 89 e7                mov    %rsp,%rdi
4010dd:       e8 55 02 00 00          callq  401337 <strings_not_equal>
4010e2:       85 c0                   test   %eax,%eax
4010e4:       74 05                   je     4010eb <phase_5+0x62>
4010e6:       e8 4b 03 00 00          callq  401436 <explode_bomb>
```

（6）最后根据 0x402470 后的表和 ASCII 码表，对照出密码为 yoapeg（每个字符大小写均可）。

3.6 阶段 6 的破解与分析

密码如下：4 1 2 3 5 6

```
Good work!  On to the next...
4 1 2 3 5 6
Congratulations! You've defused the bomb!
```

破解过程：

（1）可见答案为六个数字

```
401119:       48 89 44 24 58          mov    %rax,0x58(%rsp)
40111e:       31 c0                   xor    %eax,%eax
401120:       48 89 e6                mov    %rsp,%rsi
401123:       e8 30 03 00 00          callq  401458 <read_six_numbers>
```

（2）%r12 和%rbp 被赋以栈顶指针，并且检查指针所指的数是否大于等于 6

```
401128:       49 89 e4                mov    %rsp,%r12
40112b:       41 bd 00 00 00 00       mov    $0x0,%r13d
401131:       4c 89 e5                mov    %r12,%rbp
401134:       41 8b 04 24             mov    (%r12),%eax
401138:       83 e8 01                sub    $0x1,%eax
40113b:       83 f8 05                cmp    $0x5,%eax
40113e:       76 05                   jbe    401145 <phase_6+0x3f>
401140:       e8 f1 02 00 00          callq  401436 <explode_bomb>
```

（3）接下来开始第一个循环，以%r13d 计数，跳出循环条件为%r13d 等于 6。

```
401145:       41 83 c5 01             add    $0x1,%r13d
401149:       41 83 fd 06             cmp    $0x6,%r13d
40114d:       74 3d                   je     40118c <phase_6+0x86>
```

比较%rbp 指向的和%rsp 后 4*%rax 位置的，若相等则引爆。

```
40114f:       44 89 eb                mov    %r13d,%ebx
401152:       48 63 c3                movslq %ebx,%rax
401155:       8b 04 84                mov    (%rsp,%rax,4),%eax
401158:       39 45 00                cmp    %eax,0x0(%rbp)
40115b:       75 05                   jne    401162 <phase_6+0x5c>
40115d:       e8 d4 02 00 00          callq  401436 <explode_bomb>
```

接下来进入第二层循环，以%edx 计数，第二层每次循环 5-%r13d 次。第一层循环每次将%12 加 4。

```
401162:       83 c3 01                add    $0x1,%ebx
401165:       83 fb 05                cmp    $0x5,%ebx
401168:       7e e8                   jle    401152 <phase_6+0x4c>
40116a:       49 83 c4 04             add    $0x4,%r12
40116e:       eb c1                   jmp    401131 <phase_6+0x2b>
```

由此可见，第一个双层循环是检查输入的六个数字是否两两不相等。

(4)接下来是第二个循环，两层。

上一个循环跳出后，先判断（%rsp+%rsi）指向的内容是否大于 1。

```
40118c:       be 00 00 00 00          mov    $0x0,%esi
401191:       8b 0c 34                mov    (%rsp,%rsi,1),%ecx
401194:       b8 01 00 00 00          mov    $0x1,%eax
401199:       ba f0 32 60 00          mov    $0x6032f0,%edx
40119e:       83 f9 01                cmp    $0x1,%ecx
4011a1:       7f cd                   jg     401170 <phase_6+0x6a>
4011a3:       eb d6                   jmp    40117b <phase_6+0x75>
```

在 gdb 中查看地址 0x6032f0，发现 phase_6 中原有的链表。

```
0x6032f4 <node1+4>:     0x0060330000000001
(gdb) x/x 0x6032f0
0x6032f0 <node1>:       0x00000001000002ea
(gdb)
0x6032f8 <node1+8>:     0x0000000000603300
(gdb)
0x603300 <node2>:       0x00000002000002e7
(gdb)
0x603308 <node2+8>:     0x0000000000603310
(gdb)
0x603310 <node3>:       0x0000000300000276
(gdb)
0x603318 <node3+8>:     0x0000000000603320
(gdb)
0x603320 <node4>:       0x00000004000003cd
(gdb)
0x603328 <node4+8>:     0x0000000000603330
(gdb)
0x603330 <node5>:       0x0000000500000186
(gdb)
0x603338 <node5+8>:     0x0000000000603340
(gdb)
0x603340 <node6>:       0x00000006000000c0
```

回到第二个循环，此部分将原链表的数据，移动到输入的数字指定的位置。

```
401170:       48 8b 52 08             mov    0x8(%rdx),%rdx
401174:       83 c0 01                add    $0x1,%eax
401177:       39 c8                   cmp    %ecx,%eax
401179:       75 f5                   jne    401170 <phase_6+0x6a>
```

循环判定条件。

```
401180:       48 83 c6 04             add    $0x4,%rsi
401184:       48 83 fe 18             cmp    $0x18,%rsi
401188:       75 07                   jne    401191 <phase_6+0x8b>
40118a:       eb 19                   jmp    4011a5 <phase_6+0x9f>
```

由此可见，第二个循环是将原链表的数据按照输入的数据排序，而且推测输入的是一到六按照一定顺序排列。

（5）第三个循环，一层。

将第二个循环打乱的链表指针重新排序

```
4011a5:       48 8b 5c 24 20          mov    0x20(%rsp),%rbx
4011aa:       48 8d 44 24 20          lea    0x20(%rsp),%rax
4011af:       48 8d 74 24 48          lea    0x48(%rsp),%rsi
4011b4:       48 89 d9                mov    %rbx,%rcx
4011b7:       48 8b 50 08             mov    0x8(%rax),%rdx
4011bb:       48 89 51 08             mov    %rdx,0x8(%rcx)
4011bf:       48 83 c0 08             add    $0x8,%rax
4011c3:       48 89 d1                mov    %rdx,%rcx
4011c6:       48 39 f0                cmp    %rsi,%rax
4011c9:       75 ec                   jne    4011b7 <phase_6+0xb1>
```

（6）第四个循环，单层。

检查排序后的数据，是否按照从大到小的顺序

```
4011d3:       bd 05 00 00 00          mov    $0x5,%ebp
4011d8:       48 8b 43 08             mov    0x8(%rbx),%rax
4011dc:       8b 00                   mov    (%rax),%eax
4011de:       39 03                   cmp    %eax,(%rbx)
4011e0:       7d 05                   jge    4011e7 <phase_6+0xe1>
4011e2:       e8 4f 02 00 00          callq  401436 <explode_bomb>
4011e7:       48 8b 5b 08             mov    0x8(%rbx),%rbx
4011eb:       83 ed 01                sub    $0x1,%ebp
4011ee:       75 e8                   jne    4011d8 <phase_6+0xd2>
```

（7）综上所述，密码为六个数字，为 1-6 按照一定顺序排列。Phase_6 中原有链表的数据，按照输入的顺序排列后，会呈由大到小的顺序。

PS：phase_6 实在太长，不得不借用了微软+苹果的技术支持，见下图。。。。。

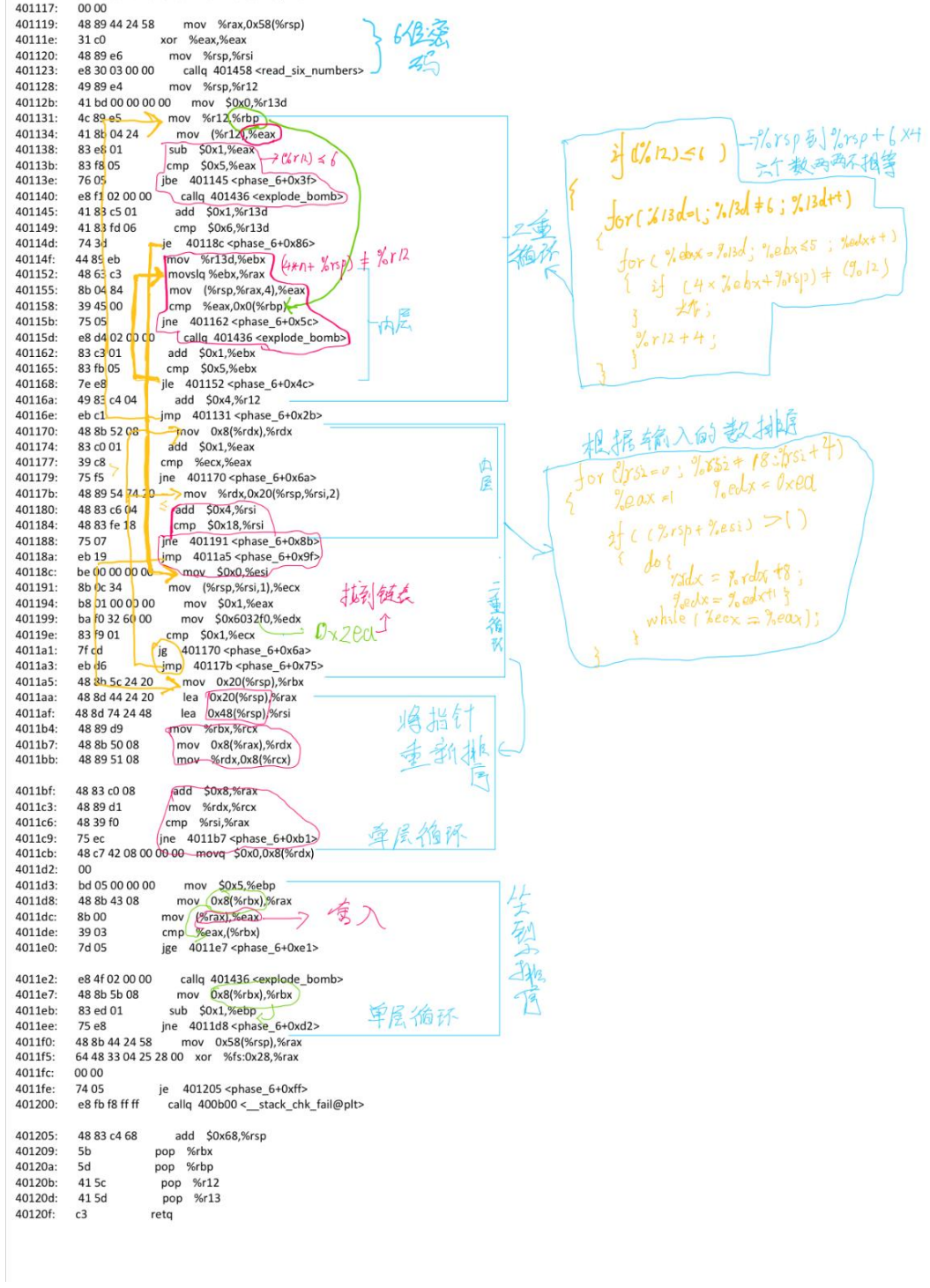
Phase_6

```
0000000000401106 <phase_6>:
 401106:    41 55              push   %r13
 401108:    41 54              push   %r12
 40110a:    55                 push   %rbp
 40110b:    53                 push   %rbx
 40110c:    48 83 ec 68        sub    $0x68,%rsp
 401110:    64 48 8b 04 25 28 00   mov    %fs:0x28,%rax
 401117:    00 00
 401119:    48 89 44 24 58     mov    %rax,0x58(%rsp)
 40111e:    31 c0              xor    %eax,%eax
 401120:    48 89 e6           mov    %rsp,%rsi
 401123:    e8 30 03 00 00     callq  401458 <read_six_numbers>
 401128:    49 89 e4           mov    %rsp,%r12
 40112b:    41 bd 00 00 00 00  mov    $0x0,%r13d
 401131:    4c 89 e5           mov    %r12,%rbp
 401134:    41 8b 04 24        mov    (%r12),%eax
 401138:    83 e8 01           sub    $0x1,%eax
 40113b:    83 f8 05           cmp    $0x5,%eax
 40113e:    76 05              jbe    401145 <phase_6+0x3f>
 401140:    e8 f1 02 00 00     callq  401436 <explode_bomb>
 401145:    41 83 c5 01        add    $0x1,%r13d
 401149:    41 83 fd 06        cmp    $0x6,%r13d
 40114d:    74 3d              je     40118c <phase_6+0x86>
 40114f:    44 89 eb           mov    %r13d,%ebx
 401152:    48 63 c3           movslq %ebx,%rax
 401155:    8b 04 84           mov    (%rsp,%rax,4),%eax
 401158:    39 45 00           cmp    %eax,0x0(%rbp)
 40115b:    75 05              jne    401162 <phase_6+0x5c>
 40115d:    e8 d4 02 00 00     callq  401436 <explode_bomb>
 401162:    83 c3 01           add    $0x1,%ebx
 401165:    83 fb 05           cmp    $0x5,%ebx
 401168:    7e e8              jle    401152 <phase_6+0x4c>
 40116a:    49 83 c4 04        add    $0x4,%r12
 40116e:    eb c1              jmp    401131 <phase_6+0x2b>
 401170:    48 8b 52 08        mov    0x8(%rdx),%rdx
 401174:    83 c0 01           add    $0x1,%eax
 401177:    39 c8              cmp    %ecx,%eax
 401179:    75 f5              jne    401170 <phase_6+0x6a>
 40117b:    48 89 54 74 20     mov    %rdx,0x20(%rsp,%rsi,2)
 401180:    48 83 c6 04        add    $0x4,%rsi
 401184:    48 83 fe 18        cmp    $0x18,%rsi
 401188:    75 07              jne    401191 <phase_6+0x8b>
 40118a:    eb 19              jmp    4011a5 <phase_6+0x9f>
 40118c:    be 00 00 00 00     mov    $0x0,%esi
 401191:    8b 0c 34           mov    (%rsp,%rsi,1),%ecx
 401194:    b8 01 00 00 00     mov    $0x1,%eax
 401199:    ba f0 32 60 00     mov    $0x6032f0,%edx
 40119e:    83 f9 01           cmp    $0x1,%ecx
 4011a1:    7f cd              jg     401170 <phase_6+0x6a>
 4011a3:    eb d6              jmp    40117b <phase_6+0x75>
 4011a5:    48 8b 5c 24 20     mov    0x20(%rsp),%rbx
 4011aa:    48 8d 44 24 20     lea    0x20(%rsp),%rax
 4011af:    48 8d 74 24 48     lea    0x48(%rsp),%rsi
 4011b4:    48 89 d9           mov    %rbx,%rcx
 4011b7:    48 8b 50 08        mov    0x8(%rax),%rdx
 4011bb:    48 89 51 08        mov    %rdx,0x8(%rcx)

 4011bf:    48 83 c0 08        add    $0x8,%rax
 4011c3:    48 89 d1           mov    %rdx,%rcx
 4011c6:    48 39 f0           cmp    %rsi,%rax
 4011c9:    75 ec              jne    4011b7 <phase_6+0xb1>
 4011cb:    48 c7 42 08 00 00 00   movq   $0x0,0x8(%rdx)
 4011d2:    00
 4011d3:    bd 05 00 00 00     mov    $0x5,%ebp
 4011d8:    48 8b 43 08        mov    0x8(%rbx),%rax
 4011dc:    8b 00              mov    (%rax),%eax
 4011de:    39 03              cmp    %eax,(%rbx)
 4011e0:    7d 05              jge    4011e7 <phase_6+0xe1>

 4011e2:    e8 4f 02 00 00     callq  401436 <explode_bomb>
 4011e7:    48 8b 5b 08        mov    0x8(%rbx),%rbx
 4011eb:    83 ed 01           sub    $0x1,%ebp
 4011ee:    75 e8              jne    4011d8 <phase_6+0xd2>
 4011f0:    48 8b 44 24 58     mov    0x58(%rsp),%rax
 4011f5:    64 48 33 04 25 28 00   xor    %fs:0x28,%rax
 4011fc:    00 00
 4011fe:    74 05              je     401205 <phase_6+0xff>
 401200:    e8 fb f8 ff ff     callq  400b00 <__stack_chk_fail@plt>

 401205:    48 83 c4 68        add    $0x68,%rsp
 401209:    5b                 pop    %rbx
 40120a:    5d                 pop    %rbp
 40120b:    41 5c              pop    %r12
 40120d:    41 5d              pop    %r13
 40120f:    c3                 retq
```



3.7 阶段 7 的破解与分析(隐藏阶段)

密码如下：36

```
But finding it and solving it are quite different...
36
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```

破解过程：

（1）首先找到 secret_phase 的入口，在 phase_4 结尾发现调用第三个输入（仅需两个输入），于是从 phase_defused 入手。

```
  40106f:       48 8b 44 24 08          mov    0x8(%rsp),%rax
```

在 phase_defused 中，观察到有检验输入是否为三个的过程，在六个 phase 中并无三个输入的密码，所以认定这是判断是否进入 secret_phase 的关键。

```
  4015ee:       bf 90 38 60 00          mov    $0x603890,%edi
  4015f3:       e8 b8 f5 ff ff          callq  400bb0 <__isoc99_sscanf@plt>
  4015f8:       83 f8 03                cmp    $0x3,%eax
```

查看地址 0x4025d9。

```
(gdb) x/2s 0x4025d9
0x4025d9:       "%d %d %s"
```

进一步比对第三个输入是否正确，查看地址 0x4025e2。

```
4015fd:       be e2 25 40 00          mov    $0x4025e2,%esi
401602:       48 8d 7c 24 10          lea    0x10(%rsp),%rdi
401607:       e8 2b fd ff ff          callq  401337 <strings_not_equal>
40160c:       85 c0                   test   %eax,%eax
40160e:       75 1e                   jne    40162e <phase_defused+0x71>
```

```
(gdb) x/2s 0x4025e2
0x4025e2:       "DrEvil"
```

在第四题输入 162 3 DrEvil，进入 secret_phase。

```
Starting program: /mnt/hgfs/hitics/lab3/bomb1170300821/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?
0 1 3 6 10 15
That's number 2.  Keep going!
0 61
Halfway there!
162 3 DrEvil
So you got that one.  Try this one.
yoapeg
Good work!  On to the next...
4 1 2 3 5 6
Curses, you've found the secret phase!
But finding it and solving it are quite different...
```

（2）调用函数 read_line()，从而可以推断出此关卡应该是一个字符串作为参数。

```
40124e:       53                      push   %rbx
40124f:       e8 43 02 00 00          callq  401497 <read_line>
```

（3）<__strtol_internal@plt>，推测为 strtol()函数。strtol()函数的参数原型为 long int strtol(const char *nptr,char **endptr,int base) ，作用是将一个字符串转换成一个长整数，推测这个函数是将输入的字符串转换成十进制长整数赋给%eax 作为返回值。

```
401254:       ba 0a 00 00 00          mov    $0xa,%edx
401259:       be 00 00 00 00          mov    $0x0,%esi
40125e:       48 89 c7                mov    %rax,%rdi
401261:       e8 2a f9 ff ff          callq  400b90 <strtol@plt>
401266:       48 89 c3                mov    %rax,%rbx
```

（4）%eax 的值应该是该函数的返回值，由 cmp 语句可知返回值应该为 0x3e8+1。再观察第二个函数的解析符号，jbe 表明 0<=%eax-1<=1000 ，所以初步推定输入的字符串应该是 1-1001 的任意一个数字。

```
401269:       8d 40 ff                lea    -0x1(%rax),%eax
40126c:       3d e8 03 00 00          cmp    $0x3e8,%eax
401271:       76 05                   jbe    401278 <secret_phase+0x2a>
401273:       e8 be 01 00 00          callq  401436 <explode_bomb>
```

（5）分析地址 0x603110，发现是 0x24。

```
401278:       89 de                   mov    %ebx,%esi
40127a:       bf 10 31 60 00          mov    $0x603110,%edi
40127f:       e8 8c ff ff ff          callq  401210 <fun7>
401284:       85 c0                   test   %eax,%eax
```

```
(gdb) x/s 0x603110
0x603110 <n1>:  "$"
(gdb) x/x 0x603110
0x603110 <n1>:  0x24
(gdb)
```

（5）接下来进入函数 fun7 分析。

移动指针，如果%rdi=0，返回 0xffffffff，%rsp+8。

```
401210:       48 83 ec 08             sub    $0x8,%rsp
401214:       48 85 ff                test   %rdi,%rdi
401217:       74 2b                   je     401244 <fun7+0x34>
```

将（%rdi）的值赋给%edx，即若%esi<=24,则跳转 40122c。

```
401219:       8b 17                   mov    (%rdi),%edx
40121b:       39 f2                   cmp    %esi,%edx
40121d:       7e 0d                   jle    40122c <fun7+0x1c>
```

%eax=0.若%esi=%edx，则返回。若不相等，则%rdi+10 再调用 fun7，%eax 乘 2 加 1。

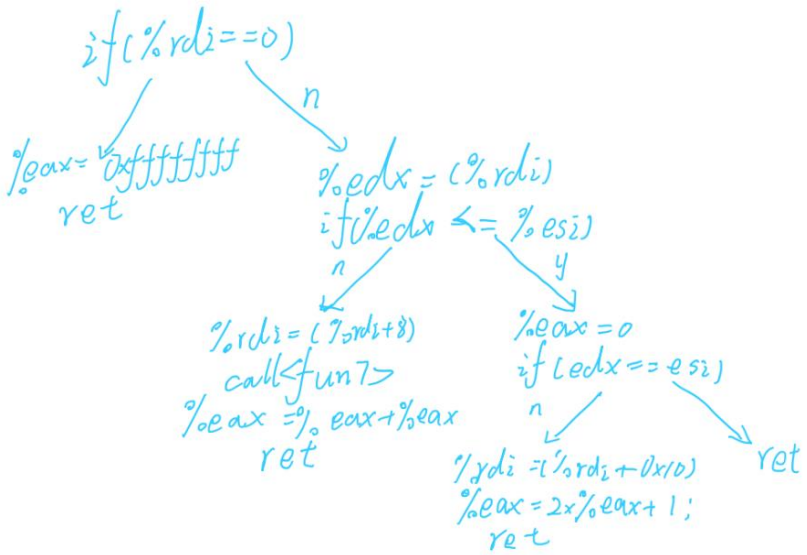
```
40122c:       b8 00 00 00 00          mov    $0x0,%eax
401231:       39 f2                   cmp    %esi,%edx
401233:       74 14                   je     401249 <fun7+0x39>

401235:       48 8b 7f 10             mov    0x10(%rdi),%rdi
401239:       e8 d2 ff ff ff          callq  401210 <fun7>
40123e:       8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
401242:       eb 05                   jmp    401249 <fun7+0x39>
```

（6）若%esi>24,则%rdi+8，%eax 乘 2，%rsp+8 返回。

```
40121f:       48 8b 7f 08             mov    0x8(%rdi),%rdi
401223:       e8 e8 ff ff ff          callq  401210 <fun7>
401228:       01 c0                   add    %eax,%eax
40122a:       eb 1d                   jmp    401249 <fun7+0x39>
```

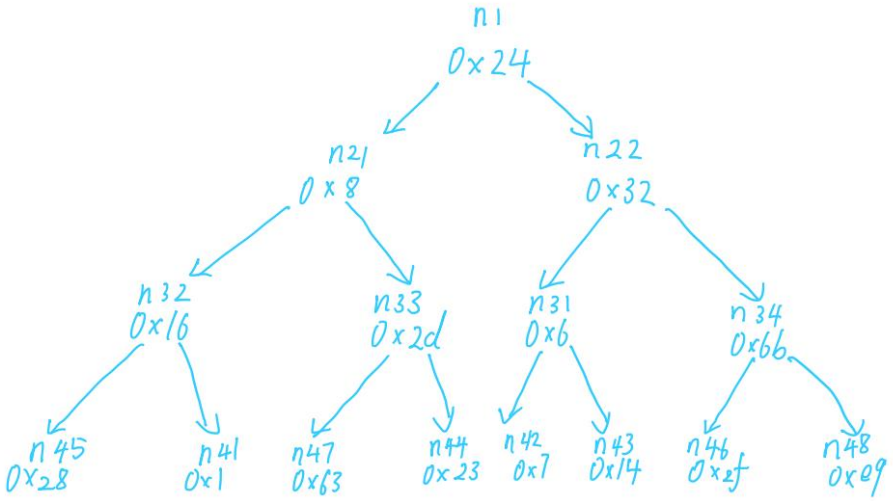
（7）整理 fun7 如下功能



（8）查询地址 0x603110 及其之后的地址。

```
(gdb) x/12xg 0x603110
0x603110 <n1>:  0x0000000000000024      0x0000000000603130
0x603120 <n1+16>:       0x0000000000603150      0x0000000000000000
0x603130 <n21>: 0x0000000000000008      0x00000000006031b0
0x603140 <n21+16>:      0x0000000000603170      0x0000000000000000
0x603150 <n22>: 0x0000000000000032      0x0000000000603190
0x603160 <n22+16>:      0x00000000006031d0      0x0000000000000000
(gdb) x/12xg 0x603170
0x603170 <n32>: 0x0000000000000016      0x0000000000603290
0x603180 <n32+16>:      0x0000000000603250      0x0000000000000000
0x603190 <n33>: 0x000000000000002d      0x00000000006031f0
0x6031a0 <n33+16>:      0x00000000006032b0      0x0000000000000000
0x6031b0 <n31>: 0x0000000000000006      0x0000000000603210
0x6031c0 <n31+16>:      0x0000000000603270      0x0000000000000000
(gdb) x/12xg 0x6031d0
0x6031d0 <n34>: 0x000000000000006b      0x0000000000603230
0x6031e0 <n34+16>:      0x00000000006032d0      0x0000000000000000
0x6031f0 <n45>: 0x0000000000000028      0x0000000000000000
0x603200 <n45+16>:      0x0000000000000000      0x0000000000000000
0x603210 <n41>: 0x0000000000000001      0x0000000000000000
0x603220 <n41+16>:      0x0000000000000000      0x0000000000000000
(gdb) x/12xg 0x603230
0x603230 <n47>: 0x0000000000000063      0x0000000000000000
0x603240 <n47+16>:      0x0000000000000000      0x0000000000000000
0x603250 <n44>: 0x0000000000000023      0x0000000000000000
0x603260 <n44+16>:      0x0000000000000000      0x0000000000000000
0x603270 <n42>: 0x0000000000000007      0x0000000000000000
0x603280 <n42+16>:      0x0000000000000000      0x0000000000000000
(gdb) x/12xg 0x603290
0x603290 <n43>: 0x0000000000000014      0x0000000000000000
0x6032a0 <n43+16>:      0x0000000000000000      0x0000000000000000
0x6032b0 <n46>: 0x000000000000002f      0x0000000000000000
0x6032c0 <n46+16>:      0x0000000000000000      0x0000000000000000
0x6032d0 <n48>: 0x00000000000003e9      0x0000000000000000
0x6032e0 <n48+16>:      0x0000000000000000      0x0000000000000000
```

整理为二叉树如下。



（9）根据返回值得出应输入 36。

第 4 章 总结

4.1 请总结本次实验的收获

（1）熟练掌握计算机系统的 ISA 指令系统与寻址方式

（2）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

（3）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

（4）

4.2 请给出对本次实验内容的建议

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.