建 新 编著

网络安全初阶

——黑客技术揭秘与防范

上海科学技术出版社

网络安全初阶

——黑客技术揭秘与防范

建 新 编著

上海科学技术出版社

网 络 安 全 初 阶
——黑客技术揭秘与防范
建 新 编著
上海科学技术出版社出版、发行
（上海瑞金二路450号 邮政编码200020）
新华书店上海发行所经销 常熟市第六印刷厂印刷
开本 787×1092 1/16 印张 12 字数 280 000
2000年7月第1版 2000年7月第1次印刷
印数 1—3 000
ISBN 7-5323-5588-8/TP·146
定价：20.00元

内 容 提 要

网络安全是现今一个非常热门而重要的问题。本书就这一问题进行了介绍和讨论。

书中主要内容包括：IP 欺骗攻击与防范，Sniffer 与防范，端口扫描与防范，口令破解器与防范，特洛伊木马与防范，缓冲区溢出攻击与防范，以及黑客进行攻击的一般步骤和实例，入侵 Windows NT 的预防方法，计算机病毒的防范措施等内容。

本书可以作为计算机爱好者，特别是对 Linux 和编程有兴趣的读者阅读。同时也是网络管理员和安全软件开发人员、软件测试人员的实用参考书。

前　言

笔者编写本书的第一个动机，是希望能让所有接触信息技术的人都能正确地了解网络安全，了解它的重要性。

网络安全技术的重要性是显而易见的。我们时常可以看到许多“黑客”的新闻：他们突破了哪个金融机构的计算机系统，获取非法帐号，为自己谋取暴利，以致触犯法律等等。可见，网络安全对维护正常的社会、经济秩序是至关重要的。

需要强调的是网络安全并非仅仅是技术问题，它还是一个社会问题。现在，有一种误解，以为网络安全是纯粹技术上的事，网络安全是高深莫测的事，是技术人员的事。其实，网络安全，人人有责。比如，在一家公司里，你有一个帐号能访问公司服务器上的一些商业机密文件，但无意中你将帐号和密码泄露了出去，或你的帐号和密码能很容易地被“黑客”攻破，这就对你公司的网络安全构成了威胁。如果你在网上购物，没有注意到相关的安全事项，可能会对你的经济带来损失。

那为什么说网络安全又是一个社会问题呢？

其实技术本身并没有过错，而问题主要在于使用技术的人。对于网络安全技术来讲，有道德的人就能用它来保护社会利益，而邪恶之徒则用它来从事非法活动。这些都是由一个人的社会价值取向来决定的，从而使得网络安全也成为了一个社会问题。

以前人们有一种对科学技术的蔑视和恐惧，害怕掌握技术的人会给社会带来危害。于是，笔者就产生了编写本书的第二个动机，希望通过对“黑客”技术的揭秘，能采取正确可行的防范措施，以确保网络的安全运行。

对网络安全技术进行深入探讨，但绝对不把这些技术用于非法和不道德的目的，绝对禁止传播各种威胁网络安全的工具。笔者认为，如果仅仅因为害怕这些技术会给社会造成危害，而禁止去讨论这些技术本身，这无疑是愚蠢的。

第三个动机，就有点自私了——是在享受科学技术提供的智慧和思辩。在收集和学习这些资料的时候，笔者觉得是在和一些充满智慧的人在谈话、在交流。通过这些程序、这些文字，使笔者不断地体会着科学技术的美的所在。他们的思想、方法都溶入了笔者的精神当中，觉得自己在不断地吸收着人类的精华，并逐步地升华。

那么，怎样介绍网络安全技术又是件很头痛的事。

从理论上讲，没有一个计算机系统是安全的。除非你不开机，或将它放在保险箱里，与外界没有任何联系。因此，为了验证系统的安全性，最好的方法就是想办法突破这个系统，发现这个系统的缺陷。那么突破或者叫做入侵应该是讲解网络安全技术的基础。只有对入侵方法有了充分的认识后，才能对系统的缺陷进行分析，找出问题的所在，才能从根本上增强系统的安全性。

要想对网络安全技术有一个很深的研究，必须具备一些必要的知识。本书的前三章提供了一些操作系统、编程方法、网络协议和网络编程等基本概念。为以后各章内容的介绍打下良好的基础。在第一章中介绍了 Linux 上的编程。因为 Linux 操作系统对网络通信有很好的支持，而且带了 gcc 编译器和 gdb 调试器是最佳选择。在 Linux 上编写的 C 程序可以很短小，代码执行效率也很高。第二章介绍了 TCP/IP 协议，对 IP 和 TCP 的数据包格式作了简单的介绍。另外讲了 TCP 连接的三次握手。许多威胁网络安全的技术都是对协议的弱点的攻击。第三章介绍了网络编程，因为在测试一个网络是否安全的时候，通常需要编个程序来完成一个特殊的测试工作。

接着几章介绍了根据 TCP/IP 协议进行的攻击与防范，Sniffer 的工作原理与防范，端口扫描的几种技术与防范，特洛伊木马的侵入与防范，缓冲区溢出攻击与防范。

然后，对“黑客”攻击网络的步骤作了一个总结，并列举了“黑客”入侵 Windows NT 的过程，对前面介绍的方法的综合利用也做了介绍。

最后，介绍了计算机病毒的原理和防范措施以及 Perl 语言。

为了对某个原理进行介绍，本书中还提供了一些简单的源代码程序。这里的程序大多数是由 C 语言编写的 Linux 程序。

本书的最终目的是想通过对“黑客”技术的揭秘，说明“黑客”技术并不是像许多媒体描述的那样高深莫测。

本书可以作为计算机爱好者，特别是对 Linux 和编程有兴趣的读者阅读。同时也是网络管理员和安全软件开发人员、软件测试人员的一本参考书。

本书仅作抛砖引玉之用，书中难免有错误之处，希望能得到批评指正。

作 者

2000 年 6 月

目　录

第一章　常见的网络操作系统 1
§1.1　Linux 1
§1.2　Windows 98 12
§1.3　Windows NT 15

第二章　TCP/IP 协议 19
§2.1　TCP/IP 协议简介 19
§2.2　以太网 24
§2.3　Internet 地址 25
§2.4　IP 协议和路由 26
§2.5　TCP 协议 28

第三章　网络编程 31
§3.1　Linux 网络编程（Berkeley Sockets） 31
§3.2　Windows 网络编程（WinSock） 40
§3.3　MFC 中的编程 44

第四章　IP 欺骗攻击与防范 50
§4.1　IP 欺骗原理 50
§4.2　IP 欺骗的防范 55
§4.3　产生 IP 欺骗包的实例 56

第五章　Sniffer 与防范 62
§5.1　Sniffer 简介 62
§5.2　一个 Sniffer 源程序 63
§5.3　怎样探测 Sniffer 72
§5.4　怎样防止被 Sniffer 72

第六章　端口扫描与防范 74
§6.1　常用网络相关命令 74
§6.2　端口扫描器 81
§6.3　一个端口扫描器的源代码 84
§6.4　端口扫描的防范 86

第七章　口令破解与防范 87
§7.1　口令破解器 87
§7.2　口令破解器的工作原理 88

§7.3 注册码破解 96
§7.4 口令破解防范 97
第八章 特洛伊木马与防范 98
§8.1 什么是特洛伊木马 98
§8.2 特洛伊木马的一个简单实例 104
§8.3 特洛伊木马的防范 112
第九章 缓冲区溢出的攻击及防范 113
§9.1 缓冲区溢出原理简介 113
§9.2 制造缓冲区溢出 114
§9.3 通过缓冲区溢出获得用户 shell 119
§9.4 利用缓冲区溢出进行的系统攻击 124
§9.5 缓冲区溢出的攻击实例 127
第十章 黑客攻击实例及防范 143
§10.1 黑客攻击的一般步骤 143
§10.2 实例 144
第十一章 黑客入侵 Windows NT 的防范 152
§11.1 通过 NetBIOS 入侵 152
§11.2 口令破解 158
§11.3 后门 158
§11.4 本地攻击 160
§11.5 防范 162
第十二章 计算机病毒及防范 163
§12.1 计算机病毒原理 163
§12.2 计算机病毒实例 166
附录 Perl 语言简介 176

第一章　常见的网络操作系统

要对网络安全技术有一定的了解，首先必须熟悉目前较流行的一些操作系统。

本章主要介绍目前常见的操作系统。首先介绍Linux系统以及在Linux系统上开发C++程序和shell程序的过程。

随后对Windows 9x的MSDOS.SYS的设置以及Windows NT中的注册表作一个介绍。这些是了解网络安全技术的基础。

§1.1　Linux

Linux是一个自由软件，可在个人计算机上运行，是一个功能完整的UNIX操作系统。它是一个真正的32位多任务操作系统，完全符合POSIX 1.0标准，以前为UNIX编写的程序可以方便地移植过来。Linux对网络通信有很好的支持，并且支持图形用户界面，有一个完整的软件开发环境，用它来开发程序，可以将精力集中在具体功能的实现上，开发出的程序很简洁。

§1.1.1　Linux下的C++编程

目前发行的Linux中，包含了gcc编译器和gdb调试器。你可以很方便地用它来编写程序。

1．elf和a.out

在Linux下，有两种可执行文件格式：elf和a.out。你的Linux有可能只支持其中的一种，也有可能两种都支持。运行一下命令“file”，如果命令输出包含elf，则支持elf格式，如果包含Linux/i386，则支持a.out格式。

2．gcc版本

gcc是一个广泛使用的C++编译器。使用如下命令，可以知道它的版本：

```
gcc -v
```

3．gcc安装后的目录结构

由于Linux的版本不同，gcc会安装在不同的目录下。下面是可能出现的几种情况：/usr/lib/gcc-lib/target/version/（及子目录），编译器就在这个目录下。

/usr/bin/gcc，可以从命令行执行的二进制程序就在这个目录下。

/usr/target/(bin|lib|include)/，库和头文件在这个目录下。

/lib/,/usr/lib 和其他目录，系统的库在这些目录下。

4．符号定义

gcc 编译器内部预定义了一些符号。使用-v 开关，就能看到 gcc 定义的符号，参见下列实例：

$ echo 'main(){printf("hello world\n");}' | gcc -E -v –

键入上面一行后的输出结果如下：

```
Reading specs from /usr/lib/gcc-lib/i486-box-linux/2.7.2/specs
gcc version 2.7.2
/usr/lib/gcc-lib/i486-box-linux/2.7.2/cpp -lang-c -v -undef
-D__GNUC__=2 -D__GNUC_MINOR__=7 -D__ELF__ -Dunix -Di386 -Dlinux
-D__ELF__ -D__unix__ -D__i386__ -D__linux__ -D__unix -D__i386
-D__linux -Asystem(unix) -Asystem(posix) -Acpu(i386)
-Amachine(i386) -D__i486__ -
```

5．gcc 编译器使用简介

gcc 编译器在使用时通常需要后跟一些选项和文件名。gcc 命令的基本用法如下：

gcc [options] [filenames]

通过选择不同的编译选项可以指定编译器进行编译。

gcc 选项中有 100 多个编译选项。这些选项中的许多项可能很少用到，但一些主要的选项会经常遇到。很多 gcc 选项包括一个以上的字符，因此必须为每个选项指定各自的连字符（-）。例如，下面的两个命令是不同的：

gcc -p -g test.c

gcc -pg test.c

第一条命令告诉 gcc 编译 test.c 时为 prof 命令建立剖析（profile）信息，并且把调试信息加入到可执行的文件里。第二条命令只告诉 gcc 为 gprof 命令建立剖析信息。

没有选项时，gcc 会生成一个名为 a.out 的可执行文件。

使用-o 编译选项，将产生的可执行文件用指定的文件名来命名。例如，将一个叫 count.c 的 C 程序编译为名叫 count 的可执行文件，输入命令如下：

gcc -o count count.c

-c 选项告诉 gcc 仅把源代码编译为目标代码。缺省时 gcc 建立的目标代码文件有一个.o 的扩展名。

-S 编译选项告诉 gcc 在为 C 代码产生了汇编语言文件后停止编译。gcc 产生的汇编语言文件的缺省扩展名是.s。

-E 选项指示编译器仅对输入文件进行预处理。在使用这个选项时，预处理器的输出被送到标准输出，而不是储存在文件里。

用 gcc 编译 C 代码时，它会试着用最少的时间完成编译，并且使编译后的代码易于

调试。易于调试意味着编译后的代码没有经过优化。必要时，需让编译器对代码进行优化。

-O 选项告诉 gcc 对源代码进行基本优化。这些优化在大多数情况下都会使程序执行得更快。-O2 选项告诉 gcc 产生尽可能小和尽可能快的代码。-O2 选项将使编译的速度比使用-O 时慢，但通常产生的代码执行速度会更快。

gcc 支持数种调试和剖析选项，常用到的是-g 和-pg。

-g 选项告诉 gcc 产生能被 gnu 调试器使用的调试信息，以便调试你的程序。gcc 提供了一个很多其他 C 编译器里没有的特性，在 gcc 里，使-g 和-O（产生优化代码）能够联用。

-pg 选项告诉 gcc 在编译好的程序里加入额外的代码。运行程序时，产生 gprof 用的剖析信息以显示你的程序的耗时情况。

6．gdb 调试器使用简介

（1）用 gdb 调试 gcc 程序：Linux 包含了一个叫 gdb 的 gnu 调试程序。它在程序运行时能观察程序内部的结构和其内存使用情况。以下是 gdb 所提供的一些功能：

监视程序中变量的值；

设置断点，使程序在指定的代码行上停止执行；

一行行地执行代码。

为了用 gdb 调试程序，在编译时必须指定调试选项。在命令行上键入 gdb，并按回车键就可以运行 gdb 了。如果一切正常的话，gdb 将被启动，并在屏幕上显示：

GDB is free software and you are welcome to distribute copies of it under certain conditions; type "show copying" to see the conditions.

There is absolutely no warranty for GDB; type "show warranty" for details.

GDB 4.14 (i486-slakware-linux), Copyright 1995 Free Software Foundation, Inc.

(gdb)

你可以在启动 gdb 时，加入许多选项。也可以在这个命令后面直接指定要调试的程序：

gdb <fname>.

（2）gdb 基本命令：gdb 支持很多命令，这些命令有从简单的文件装入到复杂的允许检查所调用的堆栈内容等。下面列出了用 gdb 调试时需要用到的一些命令。

命令	描　述
file	装入想要调试的可执行文件。
kill	终止正在调试的程序。
list	列出产生执行文件的源代码的一部分。
next	执行下一行源代码但不进入函数内部。
step	执行下一行源代码而且进入函数内部。
run	执行当前正在调试的程序。
quit	终止 gdb。
watch	监视一个变量的值而不管它何时被改变。
break	在代码里设置断点，使程序执行到这里时被挂起。
make	不退出 gdb 就可以重新产生可执行文件。

shell　　不离开 gdb 就执行 unix shell 命令。

（3）gdb 应用举例：下面列出一个将被调试的程序，这个程序被称为 greeting。它先显示一个简单的问候，再用反序将它列出。

文件 test.c.gdb 调试例程如下：

```
#include   <stdio.h>

main ()
{
  char my_string[] = "hello there";

  my_print (my_string);
  my_print2 (my_string);
}

void my_print (char *string)
{
  printf ("The string is %s\n", string);
}

void my_print2 (char *string)
{
  char *string2;
  int size, i;
  size = strlen (string);
  string2 = (char *) malloc (size + 1);
  for (i = 0; i < size; i++)
    string2[size - i] = string[i];
  string2[size+1] = '\0';
  printf ("The string printed backward is %s\n", string2);
}
```

现在用如下的命令编译这个程序：

gcc -o –g greeting test.c

运行编译好的程序，显示如下：

The string is hello there

The string printed backward is

输出的第一行是正确的，但第二行的东西并不是我们所期望的。我们所设想的输出应该是：

The string printed backward is ereht olleh

由于某些原因，my_print2 函数没有正常工作。现用 gdb 查看问题究竟出在哪里。先键入如下命令：

gdb greeting

如果在输入命令时忘了把要调试的程序名作为参数传给 gdb，可以在 gdb 提示符下用 file 命令来载入它：

(gdb) file greeting

这个命令载入 greeting 可执行文件，就像在 gdb 命令行里指定启动 gdb 装入它一样。随后就能用 gdb 的 run 命令来运行 greeting。当它在 gdb 里被运行后，结果大约如下：

```
(gdb) run
Starting program: /root/greeting
The string is hello there
The string printed backward is
Program exited with code 041
```

这个输出和在 gdb 外面运行的结果一样。问题是，为什么反序输出没有工作。为了找出症结所在，我们可以在 my_print2 函数的 for 语句后设一个断点，具体的做法是在 gdb 提示符下键入 list 命令三次，列出源代码：

```
(gdb) list
(gdb) list
(gdb) list
```

第一次键入 list 命令的输出如下：

```
1       #include   <stdio.h>
2
3       main ()
4       {
5          char my_string[] = "hello there";
6
7          my_print (my_string);
8          my_print2 (my_string);
9       }
10
```

如果按下回车键，gdb 将再执行一次 list 命令，给出下列输出：

```
11      my_print (char *string)
12      {
13         printf ("The string is %s\n", string);
14      }
15
16      my_print2 (char *string)
```

```
17      {
18        char *string2;
19        int size, i;
20
```

再按一次回车键，将列出 greeting 程序的剩余部分：

```
21        size = strlen (string);
22        string2 = (char *) malloc (size + 1);
23        for (i = 0; i < size; i++)
24        string2[size - i] = string[i];
25        string2[size+1] = `\0';
26        printf ("The string printed backward is %s\n", string2);
27      }
```

根据列出的源程序，你能看到要设断点的地方在第 24 行，在 gdb 命令行提示符下键入如下命令设置断点：

```
(gdb) break 24
```

该命令的执行结果如下：

```
Breakpoint 1 at 0x139: file greeting.c, line 24
(gdb)
```

现在再键入 run 命令，将产生如下的输出：

```
Starting program: /root/greeting
The string is hello there
Breakpoint 1, my_print2 (string = 0xbfffdc4 "hello there") at greeting.c :24
24   string2[size-i]=string[i]
```

你能通过设置一个观察 string2[size - i]变量值的观察点，来观察错误是怎样产生的，方法是键入：

```
(gdb) watch string2[size - i]
```

执行结果如下：

```
Watchpoint 2: string2[size - i]
```

现在可以用 next 命令来一步步地执行 for 循环了：

```
(gdb) next
```

经过第一次循环后，gdb 告诉我们 string2[size - i]的值是`h`。这是执行 next 命令后的结果：

```
Watchpoint 2, string2[size - i]
Old value = 0 `\000'
New value = 104 `h'
my_print2(string = 0xbfffdc4 "hello there") at greeting.c:23
23 for (i=0; i<size; i++)
```

这个值正是期望的。后来数次循环的结果都是正确的。当 i=10 时，表达式 string2[size

- i]的值等于`e`，size - i 的值等于 1，最后一个字符已经拷到新的字符串里了。

如果再把循环执行下去，会看到已经没有值分配给 string2[0]了，而它是新的字符串的第一个字符，因为 malloc 函数在分配内存时把它们初始化为空(null)字符。所以 string2 的第一个字符是空字符。于是就发现了为什么在输出 string2 时没有任何输出了。

找出了问题所在，修正这个错误是很容易的。把代码里写入 string2 的第一个字符的偏移量改为 size - 1 即可（而不是 size）。这是因为 string2 的大小为 12，但起始偏移量是 0，字符串内的字符从偏移量 0 到偏移量 10，偏移量 11 为空字符保留。

为了使代码正常工作的修改办法有很多种。还有一种是另设一个比字符串的实际大小要小 1 的变量，下面是使用这种方法的程序。

```
#include   <stdio.h>
main ()
{
  char my_string[] = "hello there";
  my_print (my_string);
  my_print2 (my_string);
}
my_print (char *string)
{
  printf ("The string is %s\n", string);
}
my_print2 (char *string)
{
  char *string2;
  int size, size2, i;
  size = strlen (string);
  size2 = size -1;
  string2 = (char *) malloc (size + 1);
  for (i = 0; i < size; i++)
    string2[size2 - i] = string[i];
  string2[size] = `\0';
  printf ("The string printed backward is %s\n", string2);
}
```

§1.1.2 Linux shell 编程

shell 编程是指写一个包含一系列 UNIX 命令的程序，这个程序可以在命令行运行。能否用 shell 编程实现一些常用的功能，是判断一个程序员是不是高手的标准。

用下面的命令可以执行一个 shell 程序：

方式一：

```
$ sh cmd.file
```

方式二：

```
$ cmd.file;
```

方式三：

```
$ chmod u+x cmd.file
$ cmd.file
```

1．怎样创建和运行一个 shell 脚本

在一个编辑器里，写入一系列 UNIX 命令，现举个例子：

```
echo This is a shell program
echo Today I am going to
echo $1 $2 $3 $4 $5 $6 $7 $8 $9
```

保存这个文件，并将其命名为 ex1。然后用“chmod 700 ex1”命令，将该文件设置为可执行文件。做完上述步骤即创建了 shell 脚本。如果要看运行这个文件会出现什么结果，可以在命令行状态下键入：ex1 coffee bar in hangzhou。

上述程序中最后一行就是将 ex1 命令中的单词一一读入内存。$1 代表第一个单词，$2 代表第二个单词。

可见，shell 程序的目的是能批量处理命令，从而完成一些比较复杂的工作。

不同的 shell 有不同的启动文件，比如：

bash：.profile

sh：.profile

csh：.cshrc

tcsh：.cshrc

zsh：$ZDOTDIR/.zprofile and/or $ZDOTDIR/.zshrc

所有的这些启动文件都要读入.login 和.logout 文件中。

2．shell 程序设计

（1）注释操作符：“#”引入注释。

（2）数值操作符：

= 等于；

-n 不等于；

-gt 大于；

-lt 小于；

-le 小于等于。

exit 命令：用于结束 shell 脚本，可以带一个返回值。

expr 命令：以数值和算术运算符作为参数计算结果，并将其返回标准输出。

```
$ expr 4 + 5
```

9

$

合法算术运算符有+、-、*、/和%。在*和/之前必须冠以反斜线，以防被 shell 先行解释。

（3）if 操作符：为条件判断语句。

语法：

```
if [ 条件表达式 ]
then
    命令序列
fi
```

或

```
if [ 条件表达式 ]
then
    命令序列
else
    命令序列
fi
```

（4）for 操作符：为循环语句。

语法：

```
for $环境变量 in 字符串表
do
    语句序列
done
```

while 操作符：为循环语句。

语法：

```
while [ 条件表达式 ]
do
    语句序列
done
```

（5）case 操作符：为条件控制语句。

语法：

```
case $环境变量 in
    常量 1)
        语句序列 1
        ;;
    常量 2）
        语句序列 2
        ;;
    …… ……
```