# 第8章　各种网络应用

成功配置IP和解析器之后，必须转向准备提供的网络服务。本章将全面讨论如何配置简单的网络应用，其中包括inetd服务器和源于rlogin家族的几个程序。另外，还将为大家简要介绍用作“网络文件系统”（NFS）和“网络信息系统”（NIS）的“远程进程调用”。但是，由于NFS和NIS配置涉及到的内容非常广泛，我们将分别在独立的章节中就此进行讨论。它们也适用于电子邮件和netnews新闻组。

## 8.1　inetd超级服务器

通常情况下，服务都是由daemon来执行的。所谓daemon，是一种程序，用于打开特定的端口，并等待进入的连接。如果有连接接入，它就会创建一个用于接受该连接的子进程，父进程则继续监听别的连接请求。这种方式的缺点是：对提供的每项服务而言， daemon都必须运行并在端口上监听连接请求，这通常意味着某些系统资源（比如交换空间）的浪费。

因此，几乎所有的安装过程都会运行一个“超级服务器”，这个超级服务器针对大量的服务创建套接字，并利用select(2)系统调用，同步监听所有的服务。在远程主机请求其中一项服务时，这个超级服务器就会注意到这种情况，并为该端口生成特定服务器。

常用的超级服务器是inetd，即Internet Daemon。它是在系统启动时，开始运行的，它从一个名为/etc/inetd.conf的启动文件中，选出自己准备管理的服务列表。除了调用相关的服务器外， inetd本身还要调用内部服务所涉及的大量日常服务。其中包括“ chargen”（只生成一个字符串）和“ daytime”（返回系统日期和时间）。

这个文件中的条目只有一行，这一行由下列字段组成：

```
service type protocol wait user server cmdline
```

各字段的含义参见表8-1。

**表8-1　字段定义**

| | |
|---|---|
| service | 提供服务名。必须在/etc/service文件内查找这个服务名，并把它译为端口号。有关 /etc/services文件的详情，参见第10章。 |
| type | 指定套接字类型，要么是stream（用于面向连接的协议），要么是dgram（用于数据报传输协议）。因此，基于TCP的服务应该一直采用stream，而基于UDP的服务则应该一直采用dgram。 |
| protocol | 对服务所用的传输协议进行命名。这个字段值必须是一个能够在 protocols文件内找到的有效协议名，稍后将进一步讨论。 |
| wait | 该字段只用于dgram套接字。即可以是wait，也可以是nowait。如果指定的是wait，inetd在任何时候，对指定的端口，都只执行一个服务器。如若不然，它就会在执行服务器之后，立即返回端口，监听接入连接。这对单线程服务器来说是非常有用的。所谓单线程服务器，是单纯地读取所有进入的数据报，直到再已没有数据报进入时，才退出。多数 RPC（远程进程调用）服务器都是这种类型的服务器，因此多数情况下都应该指定为 wait。与单线程服务器相反的是多线程服务器，它允许同时运行多个实例进程，数额不限。这类服务器较少使用，如果有，应该指定为 nowait。stream套接字应该始终采用nowait。 |
| user | 这是用户的登录ID，进程就在这个ID下执行。其值通常应该是根用户，但有些服务可采用不同的账号。 |

(续)

| | |
|---|---|
| server | 给出准备执行的服务器之完美路径名。内部服务通常用 internal 关键字标注。 |
| cmdline | 准备传给服务器的命令行。其中包括参数0，也就是命令名。通常情况下，这是一个服务器程序名(除非该程序的行为有别于用另一个名字调用时)。如果是内部服务，这个字段就为空。 |

下面是一个/etc/inetd.conf文件示例。

**清单8-1 /etc/inetd.conf文件示例**

```
#
# inetd services
ftp         stream tcp nowait root   /usr/sbin/ftpd     in.ftpd -l
telnet      stream tcp nowait root   /usr/sbin/telnetd  in.telnetd -b/etc/issue
#finger      stream tcp nowait bin    /usr/sbin/fingerd  in.fingerd
#tftp        dgram  udp wait   nobody /usr/sbin/tftpd    in.tftpd
#tftp        dgram  udp wait   nobody /usr/sbin/tftpd    in.tftpd /boot/diskless
login       stream tcp nowait root   /usr/sbin/rlogind  in.rlogind
shell       stream tcp nowait root   /usr/sbin/rshd     in.rshd
exec        stream tcp nowait root   /usr/sbin/rexecd   in.rexecd
#
#       inetd internal services
#
daytime     stream tcp nowait root internal
daytime     dgram  udp nowait root internal
time        stream tcp nowait root internal
time        dgram  udp nowait root internal
echo        stream tcp nowait root internal
echo        dgram  udp nowait root internal
discard     stream tcp nowait root internal
discard     dgram  udp nowait root internal
chargen     stream tcp nowait root internal
chargen     dgram  udp nowait root internal
```

finger服务被标示出来，因此它是不可用的。这样做通常是出于安全方面的考虑，因为黑客常通过这一方式获得各个网络用户名。

tftp也被标示出来。tftp实行的是“原始文件传输协议”，该协议允许在不经过密码验证的情况下，把你的系统文件传播出去。这一点对/etc/passwd文件来说，危害性不言而喻。

无盘客户机和X终端常用tftp从启动服务器下载自己的代码。如果你由于这个原因，需要运行tftpd，务必保证将其限定在客户机将从中获取文件的目录内，这是通过将这些目录名加入tftpd命令行来完成的。显示在清单8-1中的第二个tftp行。

## 8.2 tcpd访问控制工具

由于通过计算机访问网络涉及到安全方面的问题，所以在设计应用程序时应该全盘考虑。但是，有些应用程序仍然存在安全方面的不足（以RTM Internet蠕虫的表现尤为突出），或不能区分安全主机（该类主机请求的特定服务将被接受）和非安全主机（其请求将被拒绝）。我们前面已谈过tftp和finger服务，因此，应该把对这些服务的访问限定于“信任主机”，常规设置是不可能做到这一点的，这种情况下，inetd要么将该服务提供给所有客户机，要么一个也

不提供。

另一个非常有用的访问控制工具是 tcpd，该工具是 Wietse Venema（邮件地址：wietse@wzv.win.tue.nl）编写的，是一个所谓的 daemon封装器。如果你打算监控或保护的是TCP服务，就可以弃服务器程序而调用它。 tcpd将请求记入系统日志 daemon，查看是否允许远程主机使用该项服务，而且只有得到肯定的答复之后，才执行真正的服务器程序。注意，这一点不适用于基于UDP的服务。

例如，要封装finger daemon，必须将inetd.conf中的相应行改为

```
# wrap finger daemon
finger  stream  tcp  nowait  root  /usr/sbin/tcpdin.fingerd
```

没有增加任何的访问控制，除了所有请求都被记入系统日志的授权设备外，客户机上进行了一个常规的finger设置。

访问控制的实施是通过两个文件来实现的，它们是 /etc/hosts.allow和/etc/hosts.deny。这两个文件中分别包含接受和拒绝访问特定服务和主机的条目。 tcpd从一个名为biff.foobar.com的客户机主机，处理finger之类的服务请求时，它将在 hosts.deny和hosts.allow这两个文件中查看与该服务和客户机主机相符的条目。如果在 hosts.allow中找到了与请求的客户机主机相符的条目，就认可访问权，不管 hosts.deny内的情况如何。如果在 hosts.deny中找到与请求服务相符的条目，这个请求就会被拒绝，其方式是取消连接。反之，则接受服务请求。

访问文件中的条目是这样的：

```
servicelist: hostlist [:shellcmd]
```

servicelist和hostlist的定义如表8-2所示。

**表8-2 servicelist和hostlist的定义**

| | |
|---|---|
| servicelist | 这是取自/etc/services或关键字ALL的服务列表。要使之与除了finger和tftp之外的所有服务对应，利用“ ALL EXCEPT finger tftp ”即可 |
| hostlist | 这是主机名或IP地址、或ALL、LOCAL或UNKNOWN这几个关键字的列表。 ALL对应任何一台主机， LOCAL对应不包含句点的主机名（一般说来，只有通过查找 /etc/hosts得来的本地主机名中才不包含句点）。UNKNOWN对应任何一台无法找到其地址的主机。以句点开头的主机名字串对应所有其域等同于这个名字串的主机。例如， .foobar.com对应的是biff.foobar.com。另外，还为IP网络地址和子网编号准备了相应的关键字，详情参见 hosts_access手册 |

要拒绝除了本地主机以外的机器访问 finger和tftp服务，将下面一行放入/etc/hosts.deny内，让/etc/hosts.allow文件为空。

```
in.tftpd.  in.fingerd: ALL EXCEPT LOCAL, .your.domain
```

仅供选择的shellcmd字段中可以包含一个shell命令，这个命令将在已找到匹配请求条件的条目时调用。最好布置一些小把戏，引诱黑客上钩：

```
in.ftpd: ALL EXCEPT LOCAL, .vbrew.com :
echo "request from %d@%h"  /var/log/finger.log;
if [ %h != "vlager.vbrew.com" ]; then
    finger -l @%h  /var/log/finger.log
fi
```

%h和%d参数是tcpd派生出来的，分别代表客户机主机名和服务名。详情参阅 hosts_access手册。

## 8.3 服务和协议文件

对提供特定“标准”服务的端口来说，其端口号是在“已分配编号” RFC中定义的。要将服务器和客户机程序用于服务名到这些端号之间的转换，每台主机上至少得保留一份服务编号对应列表，该列表保存在一个名为 /etc/services的文件内。其中的条目结构如下：

```
service port/protocol   [aliases]
```

这里，service指的是服务名，port定义的是上面准备提供服务的端口，而 protocol定义的是准备采用的传输协议。通常，不是 udp，就是tcp。为一项服务提供多个协议是行得通的，同样，在同一个端口上提供不同的服务也是可行的，只要各项服务采用的协议不同。 aliases字段允许为同一项服务指定备用名。

一般说来，没必要更改随网络软件一起安装在系统中的 services文件。虽然如此，我们还是从中摘录了一小段代码，如下所示：

```
# The services file:
#
# well-known services
echo            7/tcp                   # Echo
echo            7/udp                   #
discard         9/tcp   sink null       # Discard
discard         9/udp   sink null       #
daytime        13/tcp                   # Daytime
daytime        13/udp                   #
chargen        19/tcp   ttytst source   # Character Generator
chargen        19/udp   ttytst source   #
ftp-data       20/tcp                   # File Transfer Protocol (Data)
ftp            21/tcp                   # File Transfer Protocol (Contr
telnet         23/tcp                   # Virtual Terminal Protocol
smtp           25/tcp                   # Simple Mail Transfer Protocol
nntp          119/tcp   readnews        # Network News Transfer Protoco
#
# UNIX services
exec          512/tcp                   # BSD rexecd
biff          512/udp   comsat          # mail notification
login         513/tcp                   # remote login
who           513/udp   whod            # remote who and uptime
shell         514/tcp   cmd             # remote command, no passwd use
syslog        514/udp                   # remote system logging
printer       515/tcp   spooler         # remote print spooling
route         520/udp   router routed   # routing information protocol
```

注意，echo服务是端口7提供的，采用的协议是 TCP和UDP，而端口512用于两项不同服务，即UDP上的COMSAT daemon（通知最近收到邮件的用户，参见 xbiff(1x)）和利用TCP的远程执行（rexec(1)）。

与services文件类似，networking库也需要一种方法将协议名（比如 services文件中所用的那些）翻译为协议编号，以便其他主机上的 IP层能够识别。这是通过在 /etc/protocols文件中查找协议名的方式得来的。该文件中每行包含一个条目，每个条目中包含一个协议名以及关联的协议编号。其示例文件如下：

```
ip    0   IP      # internet protocol, pseudo protocol number
icmp  1   ICMP    # internet control message protocol
igmp  2   IGMP    # internet group multicast protocol
tcp   6   TCP     # transmission control protocol
pup   12  PUP     # PARC universal packet protocol
udp   17  UDP     # user datagram protocol
idp   22  IDP     # WhatThis?
raw   255 RAW     # RAW IP interface
```

## 8.4 远程过程调用

对客户机-服务器应用程序来说，它们所用的常见机制是“远程进程调用”（Remote Procedure，Call RPC）包提供的。RPC是有Sun微系统公司开发的，是一个工具和库函数集。内置于RPC包之上的重要应用程序分别是网络文件系统 (NFS)和网络信息系统(NIS)，两者将在后续章节中讨论。

RPC服务器由一系列进程和进程参数组成，客户机可通过向服务器发送 RPC请求的方式，调用这些进程。服务器将调用客户机这一端指定的进程，如果有返回值的话，再将它们传回客户机。为了做到数据的传送与机器无关，客户机和服务器之间交换的所有数据都将由发送端将它们转换为一种所谓的“外部数据表示法”格式，再由接收端把数据转换成本地表示法格式。

有时，对RPC应用程序的改进往往与进程调用接口内的变动不相容。当然，只更改服务器将导致仍然在原地踏步的所有应用程序崩溃。因此， RPC程序为接口分配了不同版本号，一般从1开始，版本越新，这个数就越大。通常，服务器可以同时提供好几个不同版本；客户机根据服务请求中的版本号，作出应答。

RPC服务器和客户机之间的网络通信有些特殊。 RPC服务器提供一个或多个进程集；每个进程集就叫做一个程序，而且有一个唯一的程序编号。将服务名映射为相应程序编号的列表通常保存在/etc/rpc文件内，下面是摘自这个文件的片段：

```
#ident    "@(#) rpc          1.11          95/07/14 SMI"          /* SVr4.0 1.2
*/
#
#          rpc
#
rpcbind             100000   portmap sunrpc rpcbind
rstatd              100001   rstat rup perfmeter
rusersd             100002   rusers
nfs                 100003   nfsprog
ypserv              100004   ypprog
mountd              100005   mount showmount
ypbind              100007
walld               100008   rwall shutdown
yppasswdd           100009   yppasswd
etherstatd          100010   etherstat
rquotad             100011   rquotaprog quota rquota
sprayd              100012   spray
3270_mapper         100013
rje_mapper          100014
selection_svc       100015   selnsvc
```

```
database_svc        100016
rexd                100017    rex
alis                100018
sched               100019
llockmgr            100020
nlockmgr            100021
x25.inr             100022
statmon             100023
status              100024
ypupdated           100028    ypupdate
keyserv             100029    keyserver
bootparam           100026
sunlink_mapper      100033
tfsd                100037
nsed                100038
nsemntd             100039
showfhd             100043    showfh
ioadmd              100055    rpc.ioadmd
NETlicense          100062
sunisamd            100065
debug_svc           100066    dbsrv
ypxfrd              100069    rpc.ypxfrd
bugtraqd            100071
kerbd               100078
event               100101    na.event      # SunNet Manager
logger              100102    na.logger     # SunNet Manager
sync                100104    na.sync
hostperf            100107    na.hostperf
activity            100109    na.activity   # SunNet Manager
hostmem             100112    na.hostmem
sample              100113    na.sample
x25                 100114    na.x25
ping                100115    na.ping
rpcnfs              100116    na.rpcnfs
hostif              100117    na.hostif
etherif             100118    na.etherif
iproutes            100120    na.iproutes
layers              100121    na.layers
snmp                100122    na.snmp snmp-cmc snmp-synoptics snmp-unisys snmp-
utk
traffic             100123    na.traffic
nfs_acl             100227
sadmind             100232
nisd                100300    rpc.nisd
nispasswd           100303    rpc.nispasswdd
ufsd                100233    ufsd
pcnfsd              150001
amd                 300019    amq
bwnfsd              545580417
fypxfrd             600100069 freebsd-ypxfrd
```

TCP/IP网络中，RPC的作者面临着如何将程序编号映射为常见的网络服务这一难题。他们的解决办法是：让每台服务器为每个数据报和每个版本同时提供 TCP端口和UDP端口。一

般说来，RPC应用程序将在发送数据时，采用 UDP，而在传输的数据不能装入一个单一的UDP数据报时，转而采用TCP。

当然，客户机程序必须有一种方法，能够找出程序编号对应的是哪个端口。采用配置文件未免太死板，由于RPC应用程序不采用保留端口，因此无法保证原本打算供我们的数据库应用程序所用的端口没有被别的进程占用。鉴于此，RPC应用程序选出了自己能够得到的端口，并将其注册为所谓的portmapper daemon（端口映射器程序）。后者充当的是所有运行于这台机器上的RPC服务器的“经纪人”：对希望将服务和指定程序编号联系起来的客户机来说，先在服务器的主机上查询端口映射器，后者将返回请求服务能够抵达的 TCP和UDP端口。

上面这种方法有个明显的不足，比标准 Berkeley服务所用的inetd daemon更甚。因为在端口映射器失效时，所有的 RPC端口信息也会丢失；这通常意味着你不得不手工启动所有 RPC服务器或重启整台机器。

端口映射器名为rpc.portmap，驻留于/usr/sbin中。

**注意** 对Red Hat 6来说，端口映射器位于/sbin/portmap。

## 8.5 r 命令的配置

可在远程主机上执行的命令有许多。它们是： rlogin、rsh、rcp和rcmd。它们在远程主机上创建一个外壳，允许用户执行这些命令。当然，客户机需要拥有相应的主机账号，以便能在这些远程主机上执行命令。所以，所有这些命令都要执行一个身份验证进程。一般情况下，客户机将用户的登录名告诉服务器，服务器再依次请求以常规方式验证的密码。

```
#
# /etc/rpc - miscellaneous RPC-based services
#
portmapper      100000  portmap sunrpc
rstatd          100001  rstat rstat svc rup perfmeter
rusersd         100002  rusers
nfs             100003  nfsprog
ypserv          100004  ypprog
mountd          100005  mount showmount
ypbind          100007
walld           100008  rwall shutdown
yppasswdd       100009  yppasswd
bootparam       100026
ypupdated       100028  ypupdate
```

但有些时候，需要对特定的用户发放免检通行证。比如，如果你需要频频登录到局域网内的其他机器，你肯定希望免去每次的键入密码之苦。

我们建议取消身份验证只限于其密码数据库同步更新的少数主机，或少数特权用户（由于管理方面的原因，他们需要访问多台计算机）。只要你想允许人们未指定任何一个登录 ID或密码，就进入你的主机，一定要先保证不会偶然将访问权授予任何人！

对r命令来说，取消身份验证的方法有两种。其一是针对超级用户而言，是允许特定或所有主机上的特定或所有用户（毫无疑问，“所有”将是非常糟糕的选择）在不要求提供密码的情况下，登录进来。这种访问是由一个名为 /etc/hosts.equiv的文件控制的。该文件中包含一系列主机和用户名，这里的主机名和用户名被视为与本地主机上的用户等同。其二是针对普通

用户而言，授权特定主机上的其他用户可以访问自己的账号。这种访问控制包含在用户主目录的file.rhosts文件中。出于安全方面的考虑，该文件必须为用户或超级用户专有，而且不能是象征性的链接，否则就是无效的（在 NFS环境中，可能还需要为它提供 444的保护级，因为超级用户只能通过NFS访问磁盘上的文件）。

客户机请求r服务时，将先后在/etc/hosts.equiv和.rhosts文件内查找准备用来登录的主机和用户名。比如，Janet在Gauss这台机器上工作，想登录到 Joe在Euler主机上的账号。下面的示例将以Janet作为客户机用户，Joe作为本地用户为例。现在，Janet键入

```
$ rlogin -1 joe euler
```

在Gauss上，如果Janet被授予自由访问的话，服务器将先搜索 hosts.equiv（注意，在有人试图以根的身份登录时，hosts.equiv文件是不能搜索的），如果搜索失败，它就会试着在 Joe的根目录内的.rhosts文件中查找Janet。

Euler上的hosts.equiv文件如下所示：

```
gauss
euler
-public
quark.physics.groucho.edu      andres
```

该文件中的一个条目由一个主机名构成，用户名是可选的。如果主机名自行显示出所有的用户，该主机的所有用户都得以允许在无须验证的情况下，进入他们的本地账号。在上面的示例中，Janet得以允许从Gauss登录到她自己的账号，除了根之外的其他用户也如此。但如果Janet想以Joe的身份登录，就会像往常一样，提示她输入密码。

如果主机名后面跟有用户名，就像示例文件中的最后一行那样，该用户就可以无须任何验证，自由地访问除了根账号外的所有账号。

主机名前面还可以加上一个负号－，例如“ － public ”。它要求公开对所有的账号进行身份验证，不管用户在自己的 .rhosts文件内的授权如何。

.rhosts文件的结构和 hosts.equiv完全相同，但其含义有些差别。比如 Euler机器上，Joe的.rhosts文件是这样的；

```
chomp.cs.groucho.edu
gauss      janet
```

第一个条目授权Joe可以从chomp.cs.groucho.edu自由访问所有的账号，但该条目并没有影响Euler或Chomp上的其他账号。第二个条目是前者的变体，它授权 Janet可以从Gauss自由访问Joe的账号。

注意，客户机的主机名是通过将呼叫方的地址逆向映射为主机名这一方式获得的。所以如果解析器不知道主机的话，就无法使用这一特性。在下面这两种情况下，客户机的主机名被认为对应于hosts文件内的主机名：

- 从字面上来讲，客户机的规范主机名（而不是别名）对应于文件内的主机名。
- 如果客户机的主机名是一个完整资格域名（比如是在运行 DNS时，解析器返回的），从字面上来讲，它不对应hosts文件内的主机名，它利用本地域名，对主机名进行了扩展。

**注意** 勿庸置疑，启用r命令实在不是个好主意。相比而言，为了更好地保护自己的数据，应该采用非常严格的SSH加密和用户身份验证软件。令人遗憾的是，由于美国政府的保密策略，Linux分销商不能将SSH之类的优秀软件包含在他们的光盘内。