Linux Administrators Security Guide

LASG - 0.0.8

By Kurt Seifried (seifried@seifried.org)
Available at: https://www.seifried.org/lasg/

Table of contents
Preface
Forward by the author
What this guide is and isn't
How to determine what to secure and how to secure it
Safe installation of Linux
    Choosing your install media
    It ain't over 'til...
The Linux kernel
    Upgrading and compiling the kernel
    Kernel versions
General concepts, server verses workstations, etc
Physical / Boot Security
    Physical access
    The computer BIOS
    LILO
Shadow passwords
Safe user / system administration
    User / System information
    login.defs
    sudo
    Super
    YaST
    COAS
    Linuxconf
Log files and other forms of monitoring
    sysklogd/klogd
    bash
RPM
dpkg
tarballs / tgz
PAM
File / Filesystem Security
Virii, Trojan Horses, Worms, and Social Engineering
TCP-IP and network security
PPP security
Basic network service security
    What is running and who is it talking to?
    PS Output
    Netstat Output
    lsof
Network services config files
    inetd.conf
    TCP WRAPPERS
Network services
    Sendmail
    Qmail *
    Postfix *
    Bind
    WuFTPD

- ProFTPD *
- BeroFTPD *
- tftp
- Apache
- INN
- D News *
- NFSD
- Telnetd
- POPD/IMAPD
- Finger
- SSHD
    - psst
    - Fresh Free FiSSH
- RSH, REXEC, RCP
- SQUID
- DHCPD
- bootp *
- Identd
- Webmin
- LPD *
- xntpd *
- cu-snmp
- CVS
- rsync

SAMBA
- SWAT *

Novell *

Network based authentication
- NIS / NIS+
- Kerberoes
- Radius

Firewalling
- IPFWADM
- IPCHAINS

Encrypting services / data
- Encrypting network services
    - SSL
    - HTTP - SSL
    - Telnet - SSL
    - FTP - SSL
    - Virtual Private Network Solutions
    - IPSec
    - PPTP
    - CIPE
    - ECLiPt
- Encrypting data
    - PGP
    - GnuPG
    - CFS

Scanning and intrusion testing tools

- Network scanners
  - Strobe
  - Nmap
- Intrusion scanners
  - Nessus
  - Saint
  - Cheops
- Exploits

Scanning and intrusion detection tools
- Host based attack detection
  - Firewalling
  - TCP-WRAPPERS
  - Klaxon
  - Host Sentry
  - Logcheck
  - Port Sentry
  - Packet sniffing
- Network based attack detection
  - NFR

Packet sniffers
- tcpdump
- sniffit
- Other sniffers

Conducting baselines

Conducting audits

Backups
- Tar and Gzip
- Commercial Backup Programs for Linux
  - BRU
  - Quickstart
  - Backup Professional *
  - Arkeia
- Pro's and Con's of Backup Media

X Window system

Distribution specific tools
- RedHat *
- Debian *
- Slackware *
- Caldera *
- SuSE

Distribution specific errata and security lists
- RedHat
- Debian
- Slackware
- Caldera
- SuSE

Appendix A: Books, Magazines and other

Appendix B: ftp/www sites and online resources

Glossary / Term Definitions

Version History

Preface

Since this is an electronic document, changes will be made on a regular basis, so feel free to comment/critique/other. The author is available at:

Kurt Seifried
seifried@seifried.org
(780) 453-3174

My Verisign Class 2 digital ID


```
-----BEGIN CERTIFICATE-----
MIIDtzCCAyCgAwIBAgIQO8AwExKJ74akljwwoX4BrDANBgkqhkiG9w0BAQQFADCB
uDEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xHzAdBgNVBAsTFlZlcmlTaWduIFRy
dXN0IE5ldHdvcmsxRjBEBgNVBAsTPXd3dy52ZXJpc2lnbi5jb20vcmVwb3NpdG9y
eS9SUEEgSW5jb3JwLiBCeSBSZWYuLExJQUIuTFREKGMpOTgxNDAyBgNVBAMTK1Zl
cmlTaWduIENsYXNzIDIgQ0EgLSBJbmRpdmlkdWFsIFN1YnNjcmliZXIwHhcNOTgx
MDIxMDAwMDAwWhcNOTkxMDIxMjM1OTU5WjCB6TEXMBUGA1UEChMOVmVyaVNpZ24s
IEluYy4xHzAdBgNVBAsTFlZlcmlTaWduIFRydXN0IE5ldHdvcmsxRjBEBgNVBAsT
PXd3dy52ZXJpc2lnbi5jb20vcmVwb3NpdG9yeS9SUEEgSW5jb3JwLiBieSBSZWYu
LExJQUIuTFREKGMpOTgxJzAlBgNVBAsTHkRpZ2l0YWwgSUQgQ2xhc3MgMiAtIE1p
Y3Jvc29mdDEWMBQGA1UEAxQNS3VydCBTZWlmcmllZDEkMCIGCSqGSIb3DQEJARYV
c2VpZnJpZWRAc2VpZnJpZWQub3JnMFswDQYJKoZIhvcNAQEBBQADSgAwRwJAZsvO
hR/FIDH8V2MfrIU6edLc98xk0LYA7KZ2xx81hPPHYNvbJe0ii2fwNoye0DThJal7
bfqRI2OjRcGRQt5wlwIDAQABo4HTMIHQMAkGA1UdEwQCMAAwga8GA1UdIASBpzCA
MIAGC2CGSAGG+EUBBwEBMIAwKAYIKwYBBQUHAgEWHGh0dHBzOi8vd3d3LnZlcmlz
aWduLmNvbS9DUFMwYgYIKwYBBQUHAgIwVjAVFg5WZXJpU2lnbiwgSW5jLjADAgEB
Gj1WZXJpU2lnbidzIENQUyBpbmNvcnAuIGJ5IHJlZmVyZW5jZSBsaWFiLiBsdGQu
IChjKTk3IFZlcmlTaWduAAAAAAAAMBEGCWCGSAGG+EIBAQQEAwIHgDANBgkqhkiG
9w0BAQQFAAOBgQAwfnV6AKAetmcIs8lTkgp8/KGbJCbL94adYgfhGJ99M080yhCk
yNuZJ/o6L1VlQCxjntcwS+VMtMziJNELDCR+FzAKxDmHgal4XCinZMHp8YdqWsfC
wdXnRMPqEDW6+6yDQ/pi84oIbP1ujDdajN141YLuMz/c7JKsuYCKkk1TZQ==
-----END CERTIFICATE-----
```


I sign all my email with that certificate, so if it isn't signed, it isn't from me. Feel free to encrypt email with my certificate, I'm trying to encourage world-wide secure email (doesn't seem to be working though).

To receive updates about this book please subscribe to the announcements email list, expect an email everytime I release a new version of the guide. Send an email to **lasg-announce-request@seifried.org** with the Subject line containing the word "**subscribe**" (no quotes) and you will automatically be placed on the list. To unsubscribe send an email with the word "unsubscribe" (no quotes) in the Subject line. Otherwise take a look at https://www.seifried.org/lasg/ once in a while to see if I announce anything.

Forward by the author

I got my second (our first doesn't count, a TRS-80 that died after a few months) computer in Christmas of 1993, blew windows away 4 months later for OS/2, got a second computer in spring of 1994, loaded Linux on it (Slackware 1.?) in July of 1994. I ran Slackware for about 2-3 years and switched to RedHat after being introduced to it, after 2-3 months of RedHat exposure I switched over to it. Since then I have also earned an MCSE and MCP+Internet (come to the dark side Luke...). Why did I write this guide? Because no-one else would it seems. Why is it free? Because I want to reach the largest audience possible.

What this guide is and isn't

This guide is not a general security document. This guide is specifically about securing the Linux Operating System against specific and general threats. If you need a general overview of security please go buy "Practical Unix and Internet Security" available at www.ora.com. O'Reilly an associates is by far my favorite publisher of computer books, listed in the appendix are their books that I recommend.

How to determine what to secure and how to secure it

Are you protecting data (proprietary, confidential or otherwise), are you trying to keep certain services up (your mail server, www server, etc.), do you simply want to protect the physical hardware from damage? What are you protecting it against? Malicious damage (8 Sun Ultrasparcs), deletion (survey data), changes (a hospital with medical records), exposure (confidential internal communications concerning the lawsuit), and so on. What are the chances of a bad event happening, network probes (happen to me daily), physical intrusion (hasn't happened to me yet), social engineering ("Hi, this is Bob from IT, I need your password so we can reset it….").

You need to list out the resources (servers, services, data and other components) that contain data, provide services, make up your company infrastructure, and so on. The following is a short list:

- Physical server machines
- Mail server and services
- DNS server and services
- WWW server and services
- File server and services
- Internal company data such as accounting records and HR data
- Your network infrastructure (cabling, hubs, switches, routers, etc.)
- Your phone system (PBX, voicemail, etc.)

You then need to figure out what you want to protect it against:

- Physical damage (smoke, water, food, etc.)
- Deletion / modification of data (accounting records, defacement of your www site, etc.)
- Exposure of data (accounting data, etc.)
- Continuance of services (keep the email/www/file server up and running)
- Prevent others from using your services illegally/improperly (email spamming, etc.)

Finally what is the likelihood of an event occurring?

- Network scans – daily is a safe bet
- Social engineering – varies, usually the most vulnerable people tend to be the ones targeted
- Physical intrusion – depends, typically rare, but a hostile employee with a pair of wire cutters could do a lot of damage in a telecom closet
- Employees selling your data to competitors – it happens
- Competitor hiring skilled people to actively penetrate your network – no-one ever talks about this one but it also happens

Once you have come up with a list of your resources and what needs to be done you can start implementing security. Some techniques (physical security for servers, etc.) pretty much go without saying, in this industry there is a baseline of security typically implemented (passwording accounts, etc.). The vast majority of security problems are usually human generated, and most problems I have seen are due to a lack of education/communication

between people, there is no technical ‘silver bullet’, even the best software needs to be installed, configured and maintained by people.

Now for the stick. A short list of possible results from a security incident:

- Loss of data
- Direct loss of revenue (www sales, file server is down, etc)
- Indirect loss of revenue (email support goes, customers vow never to buy from you again)
- Cost of staff time to respond
- Lost productivity of IT staff and workers dependant on IT infrastructure
- Legal Liability (medical records, account records of clients, etc.)
- Loss of customer confidence
- Media coverage of the event

Safe installation of Linux

A proper installation of RedHat is the first step to a stable, secure system. There are various tips and tricks to make the install go easier, as well as some issues that are best handled during the install (such as disk layout).

Choosing your install media

This is the #1 issue that will affect speed of install and to a large degree safety. My personal favorite is ftp installs since popping a network card into a machine temporarily (assuming it doesn't have one already) is quick and painless, and going at 1+ megabyte/sec makes for quick package installs. Installing from CD-ROM is generally the easiest, as they are bootable, RedHat finds the CD and off you go, no pointing to directories or worrying about case (in the case of an HD install). This is also original RedHat media and you can be relatively sure it is safe, if you are paranoid however feel free to check the signatures on the files.

- FTP - quick, requires network card, and an ftp server (Windows box running something like warftpd will work as well).
- Samba - quick, good way if you have a windows machine (share the cdrom out).
- NFS - not as quick, but since nfs is usually implemented in most existing UNIX networks (and NT now has an NFS server from MS for free) it's mostly painless.
- CDROM - if you have a fast cdrom drive, your best bet, pop the cd and boot disk in, hit enter a few times and you are done.
- HardDrive - generally the most painful, windows kacks up filenames/etc, installing from an ext2 partition is usually painless though (catch 22 for new users however).

It ain't over 'til...

So you've got a fresh install of RedHat 5.2, (please, please, DO NOT install 5.0 or 5.1, it's more hassle then it's worth to upgrade, buy a new CD set), but chances are there is a lot of extra software installed, and packages you might want to upgrade or things you had better upgrade if you don't want the system compromised in the first 15 seconds of uptime (in the case of BIND/Sendmail/etc.). Keeping a local copy of the updates directory for RedHat is a good idea (ftp://updates.redhat.com/), and making it available via nfs/ftp or burning it to CD is generally the quickest way to make it available. As well there are other items you might want to upgrade, for instance I use a chroot'ed, non-root version of Bind 8.1.2, available on the contrib server (ftp://contrib.redhat.com/), instead of the stock, non-chrooted, run as root Bind 8.1.2 that ships with RedHat Linux. You will also want to remove sharp objects such as rsh, rlogin, zgv (anything that allows access over the network with insufficient authentication, or is setuid/setgid).

The Linux kernel

Linux (or according to Stallman GNU/Linux) is actually just the kernel of the operating system. the kernel is the core of the system, it handles access to all the harddrive, security mechanisms, networking and pretty much everything. It had better be secure or you are screwed.

In addition to this we have problems like the Pentium F00F bug and the inherent problems with the TCP-IP protocol, the Linux kernel has it's work cut out for it. Kernel versions are labeled as X.Y.Z, Z are minor revision numbers, Y define if the kernel is a test (odd number) or production (even number), and X defines the major revision (we have had 0, 1 and 2 so far). I would highly recommend running kernel 2.2.x, as of April 1999 this is 2.2.6. The 2.2 series of kernel has major improvements over the 2.0 series, and don't even consider the 0 or 1 series of kernels. Using the 2.2.x kernels also allows you access to newer features such as ipchains (instead of ipfwadm) and other advanced security features.

Upgrading and Compiling the Kernel

Upgrading the kernel consists of getting a new kernel and modules, editing /etc/lilo.conf, rerunning lilo to write a new MBR. The kernel will typically be placed into /boot, and the modules in /lib/modules/kernel.version.number/.

Getting a new kernel and modules can be accomplished 2 ways, by downloading the appropriate kernel RPM from ftp://updates.redhat.com/, and installing it, or by downloading the source code from ftp://ftp.kernel.org/, and compiling it.

Compiling a kernel is straightforward:

```
cd /usr/src
```

there should be a symlink called "`linux`" pointing to the directory containing the current kernel, remove it if there is, if there isn't one no problem. You might want to '`mv`' the linux directory to /usr/src/linux-kernel.version.number and create a link pointing /usr/src/linux at it.

Unpack the source code using tar and gzip as appropriate so that you now have a `/usr/src/linux` with about 50 megabytes of source code in it. The next step is to create the linux kernel configuration (/usr/src/linux.config), this can be achieved `using "make config`", "`make menuconfig`" or "`make xconfig`", my preferred method `is "make menuconfig`" (for this you will need ncurses and ncurses devel libraries). This is arguably the hardest step, there are hundreds options, which can be categorized into two main areas: hardware support, and service support. For hardware support make a list of hardware that this kernel will be running on (i.e. P166, Adaptec 2940 SCSI Controller, NE2000 ethernet card, etc.) and turn on the appropriate options. As for service support you will need to figure out which filesystems (fat, ext2, minix ,etc.) you plan to use, the same for networking (firewalling, etc.).

Once you have configured the kernel you need to compile it, the following commands makes dependencies ensuring that libraries and so forth get built in the right order, then cleans out any information from previous compiles, then builds a kernel, the modules and installs the modules.

`make dep` (makes dependencies)
`make clean` (cleans out previous cruft)
`make bzImage` (make zImage pukes if the kernel is to big, and 2.2.x kernels tend to be pretty big)
`make modules` (creates all the modules you specified)
`make modules_install` (installs the modules to /lib/modules/kernel.version.number/)

you then need to copy `/usr/src/linux/arch/i386/boot/bzImage` (zImage) `to /boot/vmlinuz-kernel.version.number`. Then edit /etc/lilo.conf, adding a new entry for the new kernel and setting it as the default image is the safest way (using the `default=X` command, otherwise it will boot the first kernel listed), if it fails you can reboot and go back to the previous working kernel. Run lilo, and reboot.

Kernel Versions

Currently we are in a stable kernel release series, 2.2.X. I would highly recomend running the latest stable kernel (currently 2.2.6 as of April 1999) as there are several nasty security problems (network attacks and denial of service attacks) that affect all kernels up to 2.0.35, 2.0.36 is patched, and the later 2.1.X test kernels to 2.2.3. Upgrading from the 2.0.X series of stable kernels to the 2.2.X series is relatively painless if you are careful and follow instructions (there are some minor issues but for most users it will go smoothly). Several software packages must be updated, libraries, ppp, modutils and others (they are covered in the kernel docs / rpm dependencies / etc.). Additionally keep the old working kernel, add an entry in lilo.conf for it as "linuxold" or something similar and you will be able to easily recover in the event 2.2.X doesn't work out as expected. Don't expect the 2.2.X series to be bug free, 2.2.6 will be found to contain flaws and will be obsoleted, like every piece of software in the world.

General concepts, server verses workstations, etc

There are many issues that affect actually security setup on a computer. How secure does it need to be? Is the machine networked? Will there be interactive user accounts (telnet/ssh)? Will users be using it as a workstation or is it a server? The last one has a big impact since "workstations" and "servers" have traditionally been very different beasts, although the line is blurring with the introduction of very powerful and cheap PC's, as well as operating systems that take advantage of them. The main difference in today's world between computers is usually not the hardware, or even the OS (Linux is Linux, NT Server and NT Workstation are close family, etc.), it is in what software packages are loaded (apache, X, etc) and how users access the machine (interactively, at the console, and so forth). Some general rules that will save you a lot of grief in the long run:

1. Keep users off of the main servers. That is to say: do not give them interactive login shells, unless you absolutely must.
2. Lock down the workstations, assume users will try to 'fix' things (heck, they might even be hostile, temp workers/etc).
3. Use encryption wherever possible to keep plain text passwords from lying around.
4. Regularly scan workstations/etc for open ports/installed software/etc that shouldn't be.

Remember: security is not a solution, it is a way of life.

Generally speaking workstations/servers are used by people that don't really care about the underlying technology, they just want to get their work done and retrieve their email in a timely fashion. There are however many users that will have the ability to change their workstation, for better or worse (install packet sniffers, warez ftp sites, www servers, irc bots, etc). To add to this most users have physical access to their workstations, meaning you really have to lock them down if you want to do it right.

1. Use BIOS passwords to lock users out of the BIOS (they should never be in here, also remember that older BIOS's have universal passwords.)
2. Set the machine to boot from the appropriate harddrive only.
3. Password the LILO prompt.
4. Do not give the user root access, use sudo to tailor access to privileged commands as needed.
5. Use firewalling so even if they do setup services they cannot make them accessible to the world.
6. Regularly scan the process table, open ports, installed software, and so on for oddities.
7. Have a written security policy that users can understand, and enforce it.
8. Remove all sharp objects (compilers, etc) unless needed.

Remember: security in depth.

Properly setup a Linux workstation is almost user proof, and generally a lot more stable then a comparable Wintel machine. With the added joy of remote administration you can keep your users happy and productive.

Servers are a different ball of wax together, and generally more important then workstations (one workstation dies, one user is affected, the email/www/ftp/etc server dies and your boss phones up in a bad mood). Unless there is a strong need, keep the number of users with

interactive shells (bash, pine, lynx based, whatever) to a bare minimum. Segment services up (have a mail server, a www server, and so on) to minimize single point of failure. Generally speaking a properly setup server will run and not need much maintenance (I have one email server at a client location that has been in use for 2 years with about 10 hours of maintenance in total). Any upgrades should be planned carefully and executed on a test. Some important points to remember with servers:

1. Restrict physical access to servers.
2. Policy of least privilege, they can break less things this way.
3. MAKE BACKUPS!
4. Regularly check the servers for changes (ports, software, etc), automated tools are great for this.
5. Software changes should be carefully planned/tested as they can have adverse affects (like kernel 2.2 no longer uses ipfwadm, wouldn't that be embarrassing if you forgot to install ipchains).

Minimization of privileges means giving users (and administrators for that matter) the minimum amount of access required to do their job. Giving a user "root" access to their workstation would make sense if all users were Linux savvy, and trustworthy, but they aren't. And even if they were it would be a bad idea as chances are they might install some software that is broken/insecure or other. If all a user access needs to do is shutdown/reboot the workstation then that is the amount of access they should be granted. You certainly wouldn't leave accounting files on a server with world readable permissions so that the accountants can view them, this concept extends across the network as a whole. Limiting access will also limit damage in the event of an account penetration (have you ever read the post-it notes people put on their monitors?).

Physical / Boot security

Physical Access

This threat is covered in depth in the "Practical Unix and Internet Security" book, but I'll cover the basics anyways. Someone turns your main accounting server off, turns it back on, boots it from a specially made floppy disk and transfers payroll.db to a foreign ftp site. Unless your accounting server is locked up what is to prevent a malicious user (or the cleaning staff of your building, the delivery guy, etc.) from doing just that? I have heard horror stories of cleaning staff unplugging servers so that they could plug their cleaning equipment in. I have seem people accidentally knock the little reset switch on power bars and reboot their servers. It just makes sense to lock your servers up in a secure room (or even a closet). It is also a very good idea to put the servers on a raised surface to prevent damage in the event of flooding (be it a hole in the roof or a super gulp slurpee.

The Computer BIOS

The computer's BIOS is on of the most powerful components, it controls how the computer boots and a variety of other things. Older bios's are infamous for having universal passwords, make sure your bios is recent and does not contain such a password. The bios can be used to lock the boot sequence of a computer to C: only, ie the first harddrive, this is a very good idea. You should also use the bios to disable the floppy drive (typically a server will not need to use it), and it can prevent users from copying data off of the network onto floppy disks. You may also wish to disable the serial ports in users machines so that they cannot attach modems, most modern computers use PS/2 keyboard and mice, there is very little reason for a serial port anymore (plus they eat up IRQ's). Same goes for the parallel port, allowing users to print in a fashion that bypasses your network, or giving them the chance to attach an external CDROM burner or harddrive can increase security greatly. As you can see this is an extension of the policy of least privilege and can decrease risks considerably, as well as making network maintenance easier (less IRQ conflicts).

LILO

Once the computer has decided to boot from C:, LILO (or whichever bootloader you use) takes over. Most bootloaders allow for some flexibility in how you boot the system, LILO especially so, this is a two edged sword. You can pass LILO arguments at boot time, the most damaging (from a security point of view) being "linux single" which boots Linux into single user mode, and by default in most distributions dumps you to root access in a command shell with no prompting for passwords or other pesky security mechanisms. Several techniques exist to minimize this risk.

`delay=X`
this controls how long (in tenths of seconds) LILO waits for user input before booting to the default selection. One of the requirements of C2 security is that this interval be set to 0 (obviously a dual boot machines blows most security out of the water). It is a good idea to set this to 0 unless the system dual boots something else.

`prompt`
forces the user to enter something, LILO will not boot the system automatically. This could be useful on servers as a way of disabling reboots without a human attendant, but typically if the hacker has the ability to reboot the system they could rewrite the MBR with new boot options. If you add a timeout option however the system will continue booting after the timeout is reached.

`restricted`
requires a password to be used if boot time options (such as "linux single") are passed to the boot loader. Make sure you use this one on each image.

`password=XXXXX`
requires user to input a password, used in conjunction with restricted, also make sure lilo.conf is no longer world readable, or any user will be able to read the password.

Here is an example of lilo.conf from one of my servers (the password has been of course changed).

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=100
default=linux
restricted
password=some_password
image=/boot/vmlinuz-2.2.5
        label=linux
        root=/dev/hda1
        read-only
```

This boots the system using the `/boot/vmlinuz-2.2.5` kernel, stored on the MBR of the first IDE harddrive of the system, the prompt keyword would normally stop unattended rebooting, however there is a 10 second timeout set (100 tenths of a second), so if you want to go into 'linux single' you have 10 seconds to type it in, at which point you would be prompted for the password ("some_password"). Combine this with a BIOS set to only boot from C: and passworded and you have a pretty secure system.

Shadow passwords

In all UNIX like operating systems there are several constants, and one of them is the file /etc/passwd and how it works. For user authentication to work properly you need (minimally) some sort of file(s) with UID to username mappings, GID to groupname mappings, passwords for the users, and other misc. info. The problem with this is that everyone needs access to the passwd file, everytime you do an ls it gets checked, so how do you store all those passwords safely, yet keep them world readable? For many years the solution has been quite simple and effective, simply hash the passwords, and store the hash, when a user needs to authenticate take the password they enter it, hash it, if it matches it was obviously the same password. The problem with this is that computing power has grown enormously, and I can now take a copy of your passwd file, and try to brute force it open in a reasonable amount of time. So to solve this several solutions exist:

- Use a 'better' hashing algorithm. Problem: can break a lot of things.
- Store the passwords elsewhere. Problem: the system/users still need access to them.

Several OS's take the first solution, Linux has implemented the second for quite a while now, it is called shadow passwords. In the passwd file your passwd is simply replaced by an 'x', which tells the system to check your passwd against the shadow file. Anyone can still read the passwd file, but only root has read access to the shadow file (the same is done for the group file and it's passwords). Seems simple enough but until recently implementing shadow passwords was a royal pain. You had to recompile all your programs that checked passwords (login, ftpd, etc, etc) and this obviously takes quite a bit of effort. This is where RedHat shines through.

To implement shadow passwords you must do two things. The first is relatively simple, changing the password file, but the second can be a pain. You have to make sure all your programs have shadow password support, which in the case of Slackware can mean recompiling a LOT of software. Some distributions make it easier however.

Because of RedHat's reliance on PAM for authentication, to implement a new authentication scheme all you need to do it add a PAM module that understand it, and edit the config file for whichever program (say login) allowing it to use that module to do authentication. No recompiling, and a minimal amount of fuss and muss, right? Without 2 utilities RedHat ships it would be extremely painful for most users to go into /etc/pam.d/ and start editing config files, however RedHat ships two utilities that do this quickly and painlessly for you.
To use shadow passwords in RedHat you need to do the following:
Install the shadow passwords utility RPM

On the RedHat 5.1 CD this is:

shadow-utils-980403-3.i386.rpm

Elsewhere just look for an rpm/tarball named:

shadow-utils-(version-release-architecture).rpm or .tar.gz or .tgz

To install it using RPM type in:

```
rpm -ivh filename.rpm
```

and then run:

```
/usr/sbin/pwconv
/usr/sbin/grpconv
```

And you are basically done. Now for an attacker to look at the hashed passwords they must go to quite a bit more effort then simply copying the /etc/passwd file. Also make sure to occasionally run pwconv and grpconv (daily from a script in /etc/cron.daily) in order to ensure all passwords are in fact shadowed. Sometimes passwords will get left in /etc/passwd, and not be sent to /etc/shadow as they should be by some utilities.

Safe user / system administration

User administration is a relatively important part of any system, the tools with which to accomplish these tasks (adding, deleting, modifying user accounts and related items) are plentiful in Linux. In addition to this Linux stores all it's configuration information in plain text files, such as /etc/passwd, making manipulation very easy and automating tasks even easier. There are also tools that provide simple interfaces to system administration tasks, and tools that enable users to safely administer the system, so far none of them are truly great, but they can make life easier.

User / System Information

User information is help primarily in two files, `/etc/passwd` and `/etc/group`, additionally user information can be held in `/etc/shadow` if you have shadow passwords enabled. The `/etc/shadow` file will also (depending on the system) allow you to set account expires and other related policies. You can manipulate these files directly, with a text editor or automated tools such as perl, grep and awk, this is only advisable if you are intimately familiar with the structure of the files and the tools you are using.

Generally speaking all 3 files contain one record per line with a return at the end. You CANNOT put comments in the file, nor can you leave blank lines, the first blank line encountered will generally be taken as the end of the file, so all the account info beyond that blank line is inactive. The fields of records are separated by full colons ":"'s, making it easy to work with and read.

Several utilities exist to help users manipulate these files to change settings such as password entry, and account data such as login shell and name. It is advisable to disable these tools if possible and only allow users to access "`passwd`", the tool used to change passwords. For the administrator there are tools such as "`useradd`" and "`userdel`" which allow you to easily add and delete user accounts, "`usermod`" which lets you modify account settings. Simply type in the command with no options, or "`man <command>`" for more details on what they allow you to do. For manipulating the group settings we have "`groupadd`", "`groupdel`" and "`groupmod`", which are self evident and provide help if you type in the command or "`man <command>`".

As mentioned previously in the shadow passwords section, do not allow users the ability to grab passwords (even if they are encrypted) of other users from the password file. The vast majority of users choose bad passwords.

login.defs

This file (`/etc/logins.def`) allows you to define some useful default values for various programs such as useradd and password expiry. It tends to vary slightly across distributions and even versions, but typically is well commented and tends to contain sane default values.

sudo

Sudo gives a user setuid access to a program(s), and you can specify which host(s) they are allowed to login from (or not) and have sudo access (thus if someone breaks into an account, but you have it locked down damage is minimized). You can specify what user a command will run as, giving you a relatively fine degree of control. If granting users access be sure to specify the hosts they are allowed to log in from and execute sudo, as well give the full pathnames to binaries, it can save you significant grief in the long run (i.e. if I give a user setuid access to "adduser", there is nothing to stop them editing their path statement, and copying "bash" into /tmp). This tool is very similar to super but with slightly less fine control. Sudo is available for most distributions as a core package or a contributed package.

Super

Super is one of the very few tools that can actually be used to give certain users (and groups) varied levels of access to system administration. In addition to this you can specify times and allow access to scripts, giving setuid access to even ordinary commands could have unexpected consequences (any editor, any file manipulation tools like chown, chmod, even tools like lp could compromise parts of the system). Debian ships with super, and there are rpm's available in the contrib directory (buildhost is listed as "localhost", you might want to find the source and compile it yourself). This is a very powerful tool (it puts sudo to shame), but requires a significant amount of effort to implement properly, I think it is worth the effort though. The head end distribution site for super is at: ftp://ftp.ucolick.org/pub/users/will/.

YaST

YaST (Yet Another Setup Tool) is a rather nice command line graphical interface (very similar to scoadmin) that provides an easy interface to most administrative tasks. It does not however have any provisions for giving users limited access, so it is really only useful for cutting down on errors, and allowing new users to administer their systems. Another problem is unlike Linuxconf it is not network aware, meaning you must log into each system you want to manipulate.

COAS

The COAS project (Caldera Open Administration System) is a very ambitious project to provide an open framework for administering systems, from a command line (with semi graphical interface), from within X (using the qt widget set) to the web. It abstracts the actual configuration data by providing a middle layer, thus making it suitable for use on disparate Linux platforms. Unfortunately it doesn't seem to be progressing very fast or far. Caldera started this project over a year ago and it is still in extreme pre beta stages due to a series of interruptions. If this project ever delivers what it is trying to do however it will be a boon to Linux administrators everywhere. The COAS site is at: http://www.coas.org/.

Linuxconf

Linuxconf is a general purpose Linux administration tool that is usable from the command line, from within X, or via it's built in www server. It is my preferred tool for automated system administration (I primarily use it for doing strange network configurations), as it is relatively light from the command line (it is actually split up into several modules). From

within X it provides an overall view of everything that can be configured (PPP, users, disks, etc.). To use it via a www browser you must first run Linuxconf on the machine and add the host(s) or network(s) you want to allow to connect (Conf > Misc > Linuxconf network access), save changes and quit, then when you connect to the machine (by default Linuxconf runs on port 98) you must enter a username and password, it only accepts root as the account, and Linuxconf doesn't support any encryption, so I would have to recommend very strongly against using this feature. Linux ships with RedHat Linux and is available at: http://www.solucorp.qc.ca/linuxconf/. Linuxconf also doesn't seem to ship with any man pages/etc, the help is contained internally which is slightly irritating.

Log files and other forms of monitoring

One integral part of any UNIX system are the logging facilities. The majority of logging in Linux is provided by two main programs, sysklogd and klogd, the first providing logging services to programs and applications, the second providing logging capability to the Linux kernel. Klogd actually sends most messages to the syslogd facility but will on occasion pop up messages at the console (i.e. kernel panics). Sysklogd actually handles the task of processing most messages and sending them to the appropriate file or device, this is configured from within `/etc/syslog.conf`. By default most logging to files takes place in `/var/log/`, and generally speaking programs that handle their own logging (such as apache) log `to /var/log/progname/`, this centralizes the log files and makes it easier to place them on a separate partition (some attacks can fill your logs quite quickly, and a full / partition is no fun). Additionally there are programs that handle their own interval logging, one of the more interesting being the `bash` command shell. By default bash keeps a history file of commands executed in `~username/.bash_history`, this file can make for extremely interesting reading, as oftentimes many admins will accidentally type their passwords in at the command line. Apache handles all of it's logging internally, configurable from `httpd.conf` and extremely flexible with the release of Apache 1.3.6 (it supports conditional logging). Sendmail handles it's logging requirements via syslogd but also has the option (via the command line -X switch) of logging all SMTP transactions straight to a file. This is highly inadvisable as the file will grow enormous in a short span of time, but is useful for debugging. See the sections in network security on apache and sendmail for more information.

sysklogd/klogd

In a nutshell klogd handles kernel messages, depending on your setup this can range from almost none to a great deal if for example you turn on process accounting. It then passes most messages to syslogd for actual handling, i.e. placement in a logfile. the man pages for sysklogd, klogd and syslog.conf are pretty good with clear examples. One exceedingly powerful and often overlooked ability of syslog is to log messages to a remote host running syslog. Since you can define multiple locations for syslog messages (ie send all kern messages to the /var/log/messages file, and to console, and to a remote host or multiple remote hosts) this allows you to centralize logging to a single host and easily check log files for security violations and other strangeness. There are several problems with syslogd and klogd however, the primary ones being the ease of which once an attacker has gained root access to deleting/modifying log files, there is no authentication built into the standard logging facilities. There are however secure versions of syslogd, available at http://www.core-sdi.com/ssyslog/ (these guys generally make good tools and have a good reputation, in any case it is open source software for those of you truly paranoid).

The standard log files that are usually defined in syslog.conf are:

```
/var/log/messages
/var/log/secure
/var/log/maillog
/var/log/spooler
```

The first one (messages) gets the majority of information typically, user login's, tcp_wrappers dumps information here, ip firewall packet logging typically dumps information here and so on. The second typically records entries for events like users changing their UID/GID (via su,

sudo, etc.), failed attempts when passwords are required and so on. The maillog file typically holds entries for every pop/imap connection (user login and logout), and the header of each piece of email that goes in or out of the system (from whom, to where, msgid, status, and so on). The spooler file is not often used anymore as the number of people running usenet or uucp has plummeted, uucp has been basically replaced with ftp and email, and most usenet servers are typically extremely powerful machines to handle a full, or even partial newsfeed, meaning there aren't many of them (typically one per ISP or more depending on size). Most home users and small/medium sized business will not (and should not in my opinion) run a usenet server, the amount of bandwidth and machine power required is phenomenal, let alone the security risks.

You can also define additional log files, for example you could add:

```
kern.*                                                  /var/log/kernel-log
```

Which would result in all kernel messages being logged to /var/log/kernel-log, this is useful on headless servers since by default kernel messages go to /dev/console (i.e. someone logged in at the machines).

bash

I will also cover bash since it is the default shell in most Linux installations, and thus it's logging facilities are generally used. bash has a large number of variables you can configure at or during run time that modify how it behaves, everything from the command prompt style to how many lines to keep in the log file.

`HISTFILE`
name of the history file, by default it is ~username/.bash_history

`HISTFILESIZE`
maximum number of commands to keep in the file, it rotates them as needed.

`HISTSIZE`
the number of commands to remember (ie when you use the up arrow key).

The variables are typically set in `/etc/profile`, which configures bash globally for all users, the values can however be over-ridden by users with the `~username/.bash_profile` file, and/or by manually using the export command to set variables such as export `EDITOR=emacs`. This is one of the reasons user directories should not be world readable, as the bash_history file can contain a lot of valuable information to a hostile party. You can also set the file itself non world readable, set your bash_profile not to log, set the file non writeable (thus denying bash the ability to write and log to it) or link it to /dev/null (this is almost always a sure sign of suspicious user activity, or a paranoid user). For the root account I would highly recomend setting the HISTFILESIZE and HISTSIZE to a low value such as 10.

RPM

RPM is a software management tool originally created by RedHat, and later GNU'ed and given to the public (www.rpm.org). It forms the core of administration on most systems, since one of the major tasks for any administrator is installing and keeping software up to date. Various estimates place most of the blame for security break-ins on bad passwords, and old software with known vulnerabilities. This isn't exactly surprising one would think, but with the average server probably containing 200-400 software packages, one begins to see why keeping software up to date can be a major task.

The man page for RPM is pretty bad, there is no nice way of putting it. The book "**Maximum RPM**" (ISBN: 0-672-31105-4) on the other hand is really wonderful (freely available at http://www.rpm.org/ in post script format). I would suggest this book for any RedHat administrator, and can say safely that it is required reading if you plan to build RPM packages. The basics of RPM are pretty self explanatory, packages come in an rpm format, with a simple filename convention:

package_name-package_version-rpm_version-architecture.rpm

nfs-server-2.2beta29-5.i386.rpm

Command	**Function**
-q	Queries Packages / Database for info
-i	Install software
-U	Upgrades or Installs the software
-e	Extracts the software from the system (removes)
-v	be more Verbose
-h	Hash marks, a.k.a. done-o-dial
Command Example	**Function**
rpm -ivh package.rpm	Install 'package.rpm', be verbose, show hash marks
rpm -Uvh package.rpm	Upgrade 'package.rpm', be verbose, show hash marks
rpm -qf /some/file	Check which package owns a file
rpm -qpi package.rpm	Queries 'package.rpm', lists info
rpm -qpl package.rpm	Queries 'package.rpm', lists all files
rpm -qa	Queries RPM database lists all packages installed
rpm -e package-name	Removes 'package-name' from the system (as listed by rpm -qa)

RedHat 5.1 ships with 528 packages, and RedHat 5.2 ships with 573, which when you think about it is a heck of a lot of software (SuSE 6.0 ships on 5 CD's). Typically you will end up with 2-300 packages installed (more apps on workstations, servers tend to be leaner, but this is not always the case). So which of these should you install and which should you avoid if possible (like the r services packages). One thing I will say, the RPM's that ship with RedHat distributions are usually pretty good, and typically last 6-12 months before they are horridly broken.

There is a list of URL's and mailing lists where distribution specific errata is later on in this document.

dpkg

The Debian package system is a similar package to RPM, however lacks some of the functionality, although overall it does an excellent job of managing software packages on a system. Combined with the dselect utility (being phased out) you can connect to remote sites, scroll through the available packages, install them, run any configuration scripts needed (like say for gpm), all from the comfort of your console. The man page for dpkg "`man dpkg`" is quite extensive.

The general format of a Debian package file (.deb) is:

packagename_packageversion-debversion.deb

ncftp2_2.4.3-2.deb

Unlike rpm files .deb files are not labeled for architecture as well (not a big deal but something to be aware of).

Command	**Function**
-I	Queries Package
-i	Install software
-l	List installed software (equiv. to rpm -qa)
-r	Removes the software from the system
Command Example	**Function**
dpkg -i package.deb	Install package.deb
dpkg -I package.deb	Lists info about package.deb (rpm -qpi)
dpkg -c package.deb	Lists all files in package.deb (rpm -qpl)
dpkg -l	Shows all installed packages
rpm -r package-name	Removes 'package-name' from the system (as listed by dpkg -l)

Debian has 1500+ packages available with the system. You will learn to love dpkg (functionally it has everything necessary, I just miss a few of the bells and whistles that rpm has, on the other hand dselect has some features I wish rpm had).

There is a list of URL's and mailing lists where distribution specific errata is later on in this document.

tarballs / tgz

Most modern Linux distributions use a package management system to install, keep track of and remove software on the system. There are however many exceptions, Slackware does not use a package management system per se, but instead has precompiled tarballs (a compressed tar file containing many files) that you simply unpack from the root directory to install. Or perhaps you want to try the latest copy of X, and no-one has yet gotten around to making a nice .rpm or .deb file, so you must grab the source code (also usually in a tarball), unpack it and install it. This present no more real danger then a package as most tarballs have MD5 and/or PGP signatures associated with them you can download and check. The real security concern with these is the difficulty in sometimes tracking down whether or not you have a certain piece of software installed, determining the version, and then removing or upgrading it. I would advise against using tarballs if at all possible, if you must use them it is a good idea to make a list of files on the system before you install it, and one afterwards, and then compare them using '`diff`' to find out what file it placed where. Simply run '`find /* > /filelist.txt`' before and '`find /* > /filelist2.txt`' after you install the tarball, and use '`diff -q /filelist.txt /filelist2.txt > /difflist.txt`' to get a list of what changed. Alternatively a '`tar -tf blah.tar`' will list the contents of the file, but like most tarballs you'll be running an executable install script/compiling and installing the software, so a simple file listing will not give you an actual picture of what was installed/etc.

PAM

"Pluggable Authentication Modules for Linux is a suite of shared libraries that enable the local system administrator to choose how applications authenticate users." Straight from the PAM documentation, I don't think I could have said it any better. But what does this actually mean? Take a 'normal' program, say login, when a user connects to a tty (via modem or telnet) a getty program answers the call (as it were) and usually starts up the 'login' program, login then requests a username, followed by a password, which it checks against the /etc/passwd file. So what happens if you have a spiffy new digital card authentication system? Well you have to recompile login (and any other apps that will use it) so they support the new system. As you can imagine this is quite laborious and prone to errors. PAM introduces a layer of middleware (nice buzzword huh?) between the application and the actual authentication mechanism. Once a program is PAM'ified, any authentication methods PAM supports will be usable by the program. In addition to this PAM can handle account, and session data which is something 'normal' authentication mechanisms don't do well. Using PAM for example you can easily disallow login access by normal users between 6pm and 6am, thus preventing security risks. By default RedHat systems are PAM aware, I'm not sure of any other distributions that are, thus in RedHat all I have to do to say implement shadow passwords is convert the password and group files, and possibly add one or two lines to some config files (if they weren't already added). Essentially PAM gives you a great deal of flexibility when handling user authentication, and will support other features in future such as transparent chroot'ing of users (be they telneting in, or ftping). This kind of flexibility will be required if Linux is to be an enterprise class operating system. Distributions that do not ship as "pam aware" can be made so but it requires a lot of effort (you must recompile all your programs with PAM support, install PAM, etc), it is probably easier to switch straight to a PAM'ified distribution if this will be a requirement. PAM usually comes with complete documentation, and if you are looking for a good overview you should visit: http://www.sun.com/software/solaris/pam/.

File / Filesystem security

A solid house needs a solid foundation, otherwise it will collapse. In Linux's case this is the ext2 (EXTended, version 2) filesystem. Pretty much your everyday standard UNIX-like filesystem. It supports file permissions (read, write, execute, sticky bit, suid, guid and so on), file ownership (user, group, other), and other standard things. Some of it's drawbacks are: no journaling, and especially no Access Control Lists, which are rumored to be in the upcoming ext3. On the plus side Linux has excellent software RAID, supporting Level 0, 1 and 5 very well (RAID isn't security related, but it certainly is safety/stability related).

The basic utilities to interact with files are: ls, chown, chmod, and find. Others include ln (for creating links), stat (tells you about a file) and many more. As for creating and maintaining the filesystems themselves we have fdisk (good old fdisk), mkfs (MaKe FileSystem, format, supports ext2, dos, minix, etc.), and fsck (FileSystem ChecK, scandisk that works).
So, what is it we are trying to prevent hostile people (usually users, and or network daemons fed bad info) from doing? A Linux system can be easily compromised if access to certain files is gained, for example the ability to read a non shadowed password file results in the ability to run the encrypted passwords against crack, easily finding weak password, this is a common goal of attackers coming in over the network (poorly written CGI scripts seem to be a favorite). Alternatively if an attacker can write to the password file he/she can at the least seriously disrupt the system, or (arguably worse) get whatever level of access they want. These conditions are commonly caused by "tmp races", where a setuid program (one running with root privileges) writes temporary files, typically in /tmp, however far to many do not check for the existence of a file, thus allowing an attacker to make a hard link in /tmp pointing to the password file, and when the setuid program is run, kaboom, /etc/passwd is wiped out or possibly appended to. There are many more attacks similar to this, so how can we prevent them?

Simple, set the file system up correctly when you install. The two common directories that users have write access to are /tmp and /home, splitting these off onto separate partitions also prevents users from filling up any critical filesystem (a full / is very bad indeed). A full /home could result in users not being able to log in at all (why root is in /root). Putting /tmp and /home on separate partitions is pretty much mandatory if users have shell access to the server, putting /etc, /var, and /usr on separate partitions is also a very good idea.

The primary tools for getting information about files/filesystems are all relatively simple and easy to use. df (shows disk usage) will also show inode usage (df -i, inodes contain information about files, you can run out of these before you run out of disk space, resulting in error messages of "disk full" when 'df' claims otherwise. This is similar to file allocation entries in Windows, with vfat it actually stores names in 8.3 format, using multiple entries for long filenames, with a max of 512 8.3 entries per directory, to many long filenames and the directory is 'full'. du will tell you the size of directories, very useful for finding out where all that disk space has disappeared to, usage is 'du' (lists everything in ./) or 'du dirname', optionally -s for a summary which is useful for dirs like /usr/src/linux. To gain information about specific files the primary tool is ls (similar to DOS's 'dir' command), 'ls' shows just file/dir names, 'ls -l' shows information such as file perms, size and so on, and 'ls -la' shows directories and files beginning in .'s, typical for config files/etc. ls has a few dozen options for sorting based on size, date, in reverse order and so forth, 'man ls' for all the details. For details on a particular file (creation date, last access, inode, etc) there is 'stat', it simply tells all the vital statistics on a given file(s), very useful to see if a file is in use/etc.

To manipulate files and folders we have the typical utilities like cp, mv, rm (CoPy, MoVe and ReMove), as well as tools for manipulating security information. chown is responsible for CHanging OWNership of files, the user and group a given file belongs to (the group other is always other, similar to Novell or NT's 'everyone' group). chmod (CHange MODe) changes a files attributes, the basic ones being read, write and execute, as well there is setuid, setguid (set user and group id the program is run as to the ones that own it, often times root), sticky bit and so forth. With proper use of assigning users to groups, chmod and chown you can emulate ACL's to a degree, but it is far less flexible then Sun/AIX/NT's file permissions (although this is rumored for ext3). Please be especially careful with setuid/setguid as any problems in that program/script can be magnified greatly.

I thought I would also mention find, it find's files, and can also filter based on permissions/ownership. A couple of quick examples for hunting down those setuid/guid programs:

find all setuid programs:
```
find / -perm +4000
```
find all setgid programs:
```
find / -perm +2000
```

The biggest part of file security however is user permissions. In Linux a file is 'owned' by 3 separate entities, a User, a Group, and Other (everyone else). You can set who is the user owner, and who is the group owner by:
```
chown user:group object
```

where object is a file, directory, etc. If you want to deny execute access to all of the 3 owners simply:
```
chmod x="" object
```

where x is a|g|u|o (All/User/Group/Other), force the permissions to be equal to "" (null, nothing, no access at all) and object is a file, directory, etc. This is by far the quickest and most effective way to rip out permissions and totally deny access to users/etc (="" forces it to clear it). Remember that root can **ALWAYS** change file perms and view/edit/run the file, Linux does not yet provide safety to users from root (which many would argue is a good thing). Also whoever owns the directory the object is in (be they a user/group/other with appropriate perms on the parent directory) can also potentially edit permissions (and since root owns / it can make changes that can traverse down the filesystem to any location).

Virii, Trojan Horses, Worms, and Social Engineering

Linux is not susceptible to virii in the same ways that a Dos/Windows or Mac platform is. In UNIX security controls are a fundamental part of the operating system, things like not allowing users to write promiscuously to any location in memory that they choose to, something that Dos/Windows and the Mac allow. To be fair there are viruses for UNIX, however the only Linux one I have seen was called "bliss", had an uninstall option ("--uninstall-please") and had to be run as root to be effective. Or to quote an old Unix favorite "if you don't know what an executable does, don't run it as root". Worms are much more prevalent in the UNIX world, the first major occurrence being the Morris Internet worm which exploited a vulnerability in sendmail. Current worms for Linux exploit broken versions of imapd, sendmail, Wu-FTPD and other daemons, the simplest fix is to keep up to date, and not make daemons accessible unless necessary. These attacks can be very successful especially if they find a network(s) that are not up to date, but typically their effectiveness fades out as people upgrade their daemons. In general I would not specifically worry about these two items, and there is definitely no need to buy anti virus software for Linux.

Social engineering on the other hand can be very effective, as no matter how many safe guards, security probes and patches you apply, humans can provide a wonderfully weak link to exploit. Case in point:

A customer bet me I couldn't crash his NT server (the bet was pizza and beer). The main problem in attacking his NT server was I had no idea what it's name or IP address was. Now I could have scanned the ISP's network using tools like ntinfosec, nbtstat and the like to look for likely netbios names, this would have taken several hours and annoyed the ISP's security officer (who I know quite well). The simplest solution was to ask the helpdesk. "Well you know I shouldn't be telling you this" and I was given his IP address after claiming I needed to fix the server. One mangled ping packet after business hours and I got my free pizza and beer. "Hi this is Bob from the IT department, we need to reinitialize all the accounts so please change your password to "temporary"". You get the idea. This is by far one of the more difficult threats to protect against, the only real answer is user education, which doesn't work very well (case in point the Melissa macro virus, many users have been told not to open documents with macros yet they did).

Worms have a long and proud tradition in the UNIX world, by exploiting known security holes (generally, very few exploit new/unknown holes), and replicating they can quickly mangle a network(s). There are several worms currently making their way around Linux machines, mostly exploiting old Bind 4.X and old IMAP software, defeating them is as easy as keeping software up to date.

Trojan horses are also popular, recently ftp.win.tue.nl was broken into and the tcp_wrappers package (among others) was modified to email passwords to an anonymous account. This was detected when someone checked the pgp signature of the package and found that it wasn't quite kosher. Moral of the story? Use software from trusted sites, and check the pgp signature(s).

Disinfection of virii / worms / trojans.

Back up your data, format and reinstall the system from known good media. Once an attacker has root on a Linux system they can literally do anything, from compromising gcc/egcs to loading interesting kernel modules at boot time.

TCP-IP and network security

TCP-IP was created in a time and place where security wasn't a very strong concern. Initially the 'Internet' (then called Arpanet) consisted of very few hosts, all were academic sites, big corporations or government in nature. Everyone knew everyone else, and getting on the Internet was a pretty big deal. the TCP-IP suite of protocol is remarkably robust (it hasn't failed yet), but unfortunately it has no real provisions for security, i.e. authentication, verification, encryption and so on. Spoofing packets, intercepting packets, reading data payloads, and so is remarkably easy in today's Internet. The most common attacks are denial of service attacks since they are the easiest to execute and the hardest to defeat, followed by packet sniffing, port scanning, and other related activities.

Hostnames don't always point at the right IP addresses and IP addresses don't always reverse lookup to the right hostname. Do not use hostname based authentication if possible, as DNS cache poisoning is relatively easy, relying on IP addresses for authentication reduces the problem to one of spoofing, but is also inherently flawed. There are no mechanisms in wide spread use to verify who sent data and who is receiving it (CIPE/IPv6 and VPN technology is starting to gain momentum however).

You can start by denying inbound data that claims to originate from your network(s), as this data is obviously spoofed, and to prevent your users from spoofing attacks you should block all outbound data that is not from your ip addresses. This is relatively simple and easy to manage but the vast majority of networks do not do it (I spent about a year pestering my ISP before they started). If everyone on the Internet had egress filters (that is restricted outbound traffic to that which is from their internal ip addresses) spoofing attacks would be impossible, and thus tracing attackers back to source would be far easier. You should also block the reserved networks (127.*, 10.*, etc.), I have noticed many attacks from the Internet with packets labeled as from those IP's, if you use network address translation (like IPMASQ) and do not have it properly firewalled you could be easily attacked, or used to relay an attack to a third party.

If you must communicate securely with people consider using VPN technology, the 'best' is probably IPSec, it is an open standard supported by all major vendors, and most major vendors have actual working implementations. Please see Appendix B or the Encrypting Services and Data section for more details.

PPP security

PPP provides TCP-IP (as well as IPX/SPX, and NetBEUI) connections over serial lines (which can of course be attached to modems). It is the primary method most people use to connect to the Internet (virtually all dial-up accounts are PPP). A PPP connection essentially consists of two computing devices (computer, a Palm Pilot, a terminal server, etc) connected over a serial link (usually via modems), both ends invoke PPP, authentication is handled (one of several ways), and the link is brought up. PPP has no real support for encryption, so if you require a secure link you must invest in some from of VPN software. Most systems invoke PPP in a rather kludgy way, you 'log in' to the equipment (terminal server, etc) and then as your login shell PPP is invoked, this of course means your username and password are sent in clear text over the line, and you must have an account on that piece of equipment, in this case PPP does not handle the authentication at all. A somewhat safer way of handling this is to use PAP (Password Authentication Protocol), where the authentication is handled by PPP, so you do not require a real account on the server, however the username and password is still sent in clear text, but the system at least is somewhat safer. The third (and best) method for authentication is to use CHAP (Challenge Handshake Authentication Protocol), each side exchanges a public key, and uses it to encrypt data sent for the authentication, thus your username and password are relatively safe from snooping, however actual data transfers are sent normally. One caveat with CHAP, Microsoft's implementation uses DES instead of MD5, making it slightly 'broken' if connecting with a Linux client, there are patches available however to fix this. PPP ships with almost every Linux distribution as a core part of the OS, the Linux PPP-HOWTO is available at: http://www.interweft.com.au/other/ppp-howto/ppp-howto.html.

Basic network service security

What is running and who is it talking to?

You can't start securing services until you know what is running. For this task ps and netstat are invaluable, ps will tell you what is currently running (httpd, inetd, etc), netstat will tell you what the status of ports are (at this point we're interested in ports that are open and listening, that is waiting for connections), and finally we can take a look at the various config files that control services.

PS Output

The program ps shows us process status (information available in the /proc/ virtual filesystem). The options most commonly used are -xau, which show pretty much all the information you'd ever want to know, please note these options vary across UNIX systems, Solaris, SCO, etc all behave differently (which is incredibly annoying). The following is typical output from a machine.

```
USER       PID %CPU %MEM  SIZE   RSS TTY STAT START   TIME COMMAND
bin        320  0.0  0.6   760   380  ?  S    Feb 12   0:00 portmap
daemon     377  0.0  0.6   784   404  ?  S    Feb 12   0:00 /usr/sbin/atd
named     2865  0.0  2.1  2120  1368  ?  S     01:14   0:01 /usr/sbin/named -u named -g named -
t /home/named
nobody     346  0.0 18.6 12728 11796  ?  S    Feb 12   3:12 squid
nobody     379  0.0  0.8  1012   544  ?  S    Feb 12   0:00 (dnsserver)
nobody     380  0.0  0.8  1012   540  ?  S    Feb 12   0:00 (dnsserver)
nobody     383  0.0  0.6   916   416  ?  S    Feb 12   0:00 (dnsserver)
nobody     385  0.0  0.8  1192   568  ?  S    Feb 12   0:00 /usr/bin/ftpget -S 1030
nobody     392  0.0  0.3   716   240  ?  S    Feb 12   0:00 (unlinkd)
nobody    1553  0.0  1.8  1932  1200  ?  S    Feb 14   0:00 httpd
nobody    1703  0.0  1.8  1932  1200  ?  S    Feb 14   0:00 httpd
root         1  0.0  0.6   776   404  ?  S    Feb 12   0:04 init [3]
root         2  0.0  0.0     0     0  ?  SW   Feb 12   0:00 (kflushd)
root         3  0.0  0.0     0     0  ?  SW   Feb 12   0:00 (kswapd)
root         4  0.0  0.0     0     0  ?  SW   Feb 12   0:00 (md_thread)
root        64  0.0  0.5   736   348  ?  S    Feb 12   0:00 kerneld
root       357  0.0  0.6   800   432  ?  S    Feb 12   0:05 syslogd
root       366  0.0  1.0  1056   684  ?  S    Feb 12   0:01 klogd
root       393  0.0  0.7   852   472  ?  S    Feb 12   0:00 crond
root       427  0.0  0.9  1272   592  ?  S    Feb 12   0:19 /usr/sbin/sshd
root       438  0.0  1.0  1184   672  ?  S    Feb 12   0:00 rpc.mountd
root       447  0.0  1.0  1180   644  ?  S    Feb 12   0:00 rpc.nfsd
root       458  0.0  1.0  1072   680  ?  S    Feb 12   0:00 /usr/sbin/dhcpd
root       489  0.0  1.7  1884  1096  ?  S    Feb 12   0:00 httpd
root       503  0.0  0.4   724   296   2 S    Feb 12   0:00 /sbin/mingetty tty2
root       505  0.0  0.3   720   228  ?  S    Feb 12   0:02 update (bdflush)
root       541  0.0  0.4   724   296   1 S    Feb 12   0:00 /sbin/mingetty tty1
root      1372  0.0  0.6   772   396  ?  S    Feb 13   0:00 inetd
root      1473  0.0  1.5  1492  1000  ?  S    Feb 13   0:00 sendmail: accepting connections on
port 25
root      2862  0.0  0.0   188    44  ?  S     01:14   0:00 /usr/sbin/holelogd.named
/home/named/dev/log
root      3090  0.0  1.9  1864  1232  ?  S     12:16   0:02 /usr/sbin/sshd
root      3103  0.0  1.1  1448   728  p1 S     12:16   0:00 su -
root      3104  0.0  1.3  1268   864  p1 S     12:16   0:00 -bash
root      3136  0.0  1.9  1836  1212  ?  S     12:21   0:04 /usr/sbin/sshd
```

The interesting ones are: portmap, named, squid (and it's dnsserver, unlinkd and ftpget children processes), httpd, syslogd, sshd, rpc.mountd, rpc.nfsd, dhcpd, inetd, and sendmail (this server appears to be providing gateway services, email and NFS file sharing). The easiest way to learn how to read ps output is go over the ps man page and learn what the various fields are (most are self explanatory, such as %CPU, some like SIZE are a bit obscure (SIZE is the number of 4k memory 'pages' a program is using)). To figure out what the running programs are a safe bet is 'man <command_name>', which almost always gives you the manual page pertaining to that service (such as httpd). You will notice that services like telnet, ftpd, identd and several others do not show up even though they are on, this is because they are run from inetd, the 'superserver', to find these services look at /etc/inetd.conf, or your netstat output.

Netstat Output

netstat tells us pretty much anything network related that you can imagine, however it is especially good at listing active connections and sockets. Using netstat we can find which ports on which interfaces are active. The following output is from a typical server using netstat –an.

```
Active Internet connections (including servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 24.108.11.200:80        205.253.183.122:3661    ESTABLISHED
tcp        0      0 0.0.0.0:1036            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 10.0.0.10:53            0.0.0.0:*               LISTEN
tcp        0      0 28.208.55.254:53        0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:139             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:25              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:2049            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:635             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
udp        0      0 127.0.0.1:1031          0.0.0.0:*
udp        0      0 0.0.0.0:1029            0.0.0.0:*
udp        0      0 0.0.0.0:800             0.0.0.0:*
udp        0      0 0.0.0.0:1028            0.0.0.0:*
udp        0      0 10.0.0.10:53            0.0.0.0:*
udp        0      0 28.208.55.254:53        0.0.0.0:*
udp        0      0 127.0.0.1:53            0.0.0.0:*
udp        0      0 10.1.0.1:138            0.0.0.0:*
udp        0      0 10.1.0.1:137            0.0.0.0:*
udp        0      0 10.0.0.10:138           0.0.0.0:*
udp        0      0 10.0.0.10:137           0.0.0.0:*
udp        0      0 0.0.0.0:138             0.0.0.0:*
udp        0      0 0.0.0.0:137             0.0.0.0:*
udp        0      0 0.0.0.0:2049            0.0.0.0:*
udp        0      0 0.0.0.0:635             0.0.0.0:*
udp        0      0 0.0.0.0:514             0.0.0.0:*
udp        0      0 0.0.0.0:111             0.0.0.0:*
raw        0      0 0.0.0.0:1               0.0.0.0:*
raw        0      0 0.0.0.0:6               0.0.0.0:*
```

Numeric output is in my opinion easier to read, once you memorize /etc/services anyways. The interesting fields for us are the first field, type of service, the fourth field which is the IP address of the interface and the port, the foreign address (if not 0.0.0.0.* means someone is actively talking to it), and the port state. The first line is a remote client talking to the web server on this machine (port 80). We then see the www server listening on 0.0.0.0:80 which means all interfaces, port 80, followed by the DNS server running on all 3 interfaces, a samba

server (139), a mail server (25), an NFS server (2049) and so on. You will notice the ftp server (21) listed, even though it is run out of inetd, and not currently in use (i.e. no one is actively ftping in), it is listed in the netstat output. This makes netstat especially useful for finding out what is active on a machine, making an inventory of active and inactive network related software on the server much easier.

lsof

lsof is a handy program similar in idea to ps, except that it prints out what files/etc are open, which can include network sockets. Unfortunately your average lsof puts out a lot of information, so you will need to use grep or redirect it through less to make sense of it.

```
squid      9726     root    4u  inet      78774                TCP localhost:2074-
>localhost:2073 (ESTABLISHED)
squid      9726     root    5u  inet      78777                TCP localhost:2076-
>localhost:2075 (ESTABLISHED)
squid      9726     root    6u  inet      78780                TCP localhost:2078-
>localhost:2077 (ESTABLISHED)
squid      9726     root    7w   CHR        1,3               6205 /dev/null
squid      9726     root   14u  inet      78789                TCP host1:3128 (LISTEN)
squid      9726     root   15u  inet      78790                UDP host1:3130
squid      9726     root   16u  inet      78791                UDP host1:3130
squid      9726     root   12u  inet     167524                TCP host1:3128->host2:3630
(ESTABLISHED)
squid      9726     root   17u  inet     167528                TCP host1:3424-
>www.playboy.com:http (SYN_SENT)
```

Shows for example that we have squid running, listening on ports 3128 and 3130, the last two lines for example shows an open connection from an internal host to squid, and the resulting action squid takes to fulfill it (going to www.playboy.com). host1 is the squid server and host2 is the client machine making the request. This is an invaluable tool for getting a precise image of what is going on network wise with your server. You can get lsof with some distributions, please not versions of lsof compiled for kernel version 2.0 will not work with kernel 2.2 and vice versa, there were to many changes. The head end site for lsof is at: ftp://vic.cc.purdue.edu/pub/tools/unix/lsof/.

Network services config files

There are several important config files that control what services Linux runs and how they run. Unfortunately many of them are in differing locations depending on what/how you installed Linux and the services. Typical locations are:

Inetd server config file:
```
/etc/inetd.conf
```

Initial start up files in various forms
```
/etc/rc.d/*
/etc/*
```

The best thing to do is figure out which services you want to run, and disable/remove the rest. On an RPM based system (RedHat, Caldera, SuSE) this is simply achieved with an rpm –e <package_name> (such as `rpm –e apache` to remove the apache www server).

inetd.conf

inetd.conf is responsible for starting services, typically ones that do not need to run continuously, or are session based (such as telnet, or ftpd). This is because the overhead of running a service constantly (like telnet) would be higher then the occasional start up cost (or firing in.telnetd up) when a user wants to use it. For some services (like DNS) that service many quick connections the overhead of starting the service every few seconds would be higher then constantly running it (also with services such as DNS and email time is critical, a few seconds delay starting an ftp session won't hurt much). The man page for inetd.conf covers the basics:

```
man inetd.conf
```

the service itself is called inetd and is run at boot time, so you can easily stop/start/reload it by manipulating the inetd process. Whenever you make changes to inetd.conf you must restart inetd to make the changes effective, killall -1 inetd will restart it properly. Lines in inetd.conf can be commented out with a # as usual (this is a very simple and effective way of disabling services like rexec). It is advisable to disable as many services in inetd.conf as possible, typically the only ones in use will be ftp, pop and imap, telnet and r services should be replaced with ssh, and services like systat/netstat and finger give away far to much information. Access to programs started by inetd can be easily controlled by the use of tcp_wrappers.

TCP WRAPPERS

Using tcp_wrappers makes securing your servers against outside intrusion is a lot simpler and painless then you would expect. TCP_WRAPPERS is controlled from two files:

```
/etc/hosts.allow
/etc/hosts.deny
```

hosts.allow is checked first, and the rules are checked from first to last, when it finds a rule that explicitly allows you in (i.e. a rule allowing your host, domain, subnet mask, etc) it lets you connect to the service, if it fails to find any rules that pertain to you in hosts.allow, it then

goes to check hosts.deny for a rule denying you entry. Again it checks the rules in hosts.deny from first to last, and the first rule it finds that denies you access (i.e. a rule disallowing your host, domain, subnet mask, etc) means it doesn't let you in. If it fails to find a rule denying you entry it then by default lets you. If you are paranoid like me the last rule (or only rule if you are going to a default policy of non-optimistic security) should be:
in hosts.deny:

```
ALL: 0.0.0.0/0.0.0.0
```

which means all services, all locations, i.e. a default deny policy. You might also want to just default deny access to say telnet, and leave ftp wide open to the world, to do this you would have:

in hosts.allow:

```
in.telnetd: 10.0.0.0/255.255.255.0      # allow access from my internal
network of 10.*.*.*
in.ftpd: 0.0.0.0/0.0.0.0                # allow access from anywhere in the
world
```

in hosts.deny:

```
in.telnetd: 0.0.0.0/0.0.0.0             # deny access to telnetd from
anywhere
```

or if you wish to be really safe:

```
ALL: 0.0.0.0/0.0.0.0            # deny access to everything from everywhere
```

This may affect services such as ssh and nfs, so be careful!
You may wish to simply list all the services you are using separately:

```
in.telnetd: 0.0.0.0/0.0.0.0
ipop3d: 0.0.0.0/0.0.0.0
```

If you leave a service on that you shouldn't have in inetd.conf, and DO NOT have a default deny policy, you could be up the creek. It is safer (and a bit more work, but in the long run less work then rebuilding the server) to have default deny rules for firewalling and tcp_wrappers, thus is you leave something on by accident, by default there will be no access to it. If you install something that users need access and you forget to put allow rules in, they will quickly complain that they can't get access and you will be able to rectify the problem quickly. Better safe then sorry. The man pages for tcp_wrappers are very good and available by:

```
man hosts.allow
```

and/or (they are the same man page):

```
man hosts.deny
```

One minor caveat with tcp_wrappers that recently popped up on bugtraq, tcp_wrappers interprets lines in hosts.allow and hosts.deny in the following manner:

strip off all \'s (line continuations), making all the lines complete (also note the max length of a line is about 2k, better to use multiple lines in some cases).

strip out lines starting with #'s, i.e. all commented out lines. Thus:

```
# this is a test
# in.ftpd: 1.1.1.1 \
in.telnetd: 1.1.1.1
```

means the "`in.telnetd: 1.1.1.1`" line would be ignored to.

Network services

Sendmail

Sendmail is another one of those services most of us have a love/hate relationship with. We hate to admin it and we'd love to replace it. This is however not 100% possible, as sendmail has many features and stability, with the downside being it runs as root. Newer mailer packages such as qmail, address these problems however they also suffer problems, qmail lacks certain features (anti spam/UBE) that sendmail has, as well as a no binary distribution clause (actually you can make binary distributions but the restrictions are very heavy and you must gain the authors permission, which he will probably not grant if you do not follow his restrictions to the letter).

Sendmail has earned itself a very bad reputation for security, however I find it hard to blame software when I find systems running versions of sendmail like 8.6.X, we are now up to 8.9.1. The root of the problem is that almost everyone runs sendmail (estimates say %70 of all Internet email passes through a sendmail box on it's journey), so as soon as a bug is found, finding a system to exploit isn't all that hard. The last few releases of sendmail have been quite good, no root hacks, etc, and with the new anti spam features sendmail is finally to come of age.

Chroot'ing sendmail is a good option, but a lot of work, and since it runs as root, rather debatable as to the effectiveness of this (since root can break out of a chrooted jail). I find that by keeping sendmail up to date, you can pretty much account yourself safe, of course if a new exploit comes out and the patch takes 24 hours we're up poop creek, the same applies to any piece of software though. Also by using Sendmail's advanced anti spam features you can effectively block 99% of all spam coming to you, and prevent other from using your system to relay spam.

Keeping sendmail up to date is relatively simple, I would recommend minimally version 8.9.2 (the 8.9 series has more anti-spam features, however RedHat's 8.8.7 rpm has most of these features as well). RedHat currently ships 8.8.7, and 8.8.8 and 8.9.X series are available in their contrib directory for hurricane/etc. You can also get the source from ftp.sendmail.org, but compiling sendmail is not for the faint of heart or those that do not have a chunk of time to devote to it.

Sendmail only needs to be accessible from the outside world if you are actually using it to receive mail from other machines and deliver the mail locally. If you only want to run sendmail so that local mail delivery works (ie a stand alone workstation, test server or other) and so mail can easily be sent to other machines simply firewall off sendmail, or better, do not run it in daemon mode. Sendmail can be run in a queue flushing node, where it simply 'wakes' up once in a while and processes local mail, either delivering it locally, or sending it off on it's way across the 'net. To set Sendmail to run in queue mode:

edit /etc/rc.d/init.d/sendmail
and change the line:

```
daemon /usr/sbin/sendmail -bd -q1h
```

to:

```
daemon /usr/sbin/sendmail -q1h
```

note if you use your system to send lots of email you may wish
to set the queue flush time lower, perhaps -q15m (15 minutes)
now outbound and system internal mail will behave just fine,
which unless you run a mail server, is perfect.

Now for all those wonderful anti-spam features in sendmail. Sendmail's configuration files consist of (this applies to Sendmail 8.9.X):

`/etc/sendmail.cf`
Primary config file, also tells where other config files are

`/etc/mail/`
You can define the location of configuration files in sendmail.cf, typically people place them in /etc/ or /etc/mail/ (which keeps it a little less cluttered).

`access`
Access list database, allows you to reject email from certain sources (IP or domain), and control relaying easily. My access file looks like this:

```
10.0.0          RELAY
spam.com                REJECT
```

which means 10.0.0.* (hosts on my internal network) are allowed to use the email server to send email to wherever they want, and that all email to or from *.spam.com is rejected. There are lists online of known spammers, typically they are 5-10,000 entries long, this can seriously impede sendmail performance (as each connection is checked against this list), on the other hand having your sendmail machine used to send spam is even worse.

`aliases`
aliases file, allows you to control delivery of mail local to the system, useful for backing up incoming users email to a separate spool. SmartList uses this file to get mail sent to lists delivered to the programs that actually process them.

`domaintable`
domain table (adding domains) that you handle, useful for virtual hosting.

`majordomo`
configuration file for majordomo, I would personally recommend SmartList over Majordomo.

`sendmail.cw`
file containing names of hosts for which we receive email, useful if you host more then one domain.

`sendmail.hf`
location of help file (telnet to port 25 and type in "HELP")

`virtusertable`
Virtual user table, maps incoming users, i.e. maps sales@example.org to john@example.org.

Sendmail 8.9.X (and previous versions) do not really support logging of all email very nicely (something required in today's world for legal reasons by many companies). This is one feature being worked on for the release of Sendmail 8.10.X. Until then there are 2 ways of logging email, the first is somewhat graceful and logs email coming IN to users on a per user

basis. The second method is not graceful and involves a simple raw log of all SMTP transactions into a file, you would have to write some sort of processor (probably in perl) to make the log useful.

Mail (incoming SMTP connections to be more precise) is first filtered by the `access` file, in here we can REJECT mail from certain domains/ip's, and RELAY mail from certain hosts (i.e. your internal network of windows machines). Any local domains you actually host mail for will need to go into `sendmail.cw`. Assuming mail has met the rules and is queued for local delivery the next file that gets checked is virtusertable, this is a listing of email addresses mapped to the account name/other email address. i.e.:

```
seifried@seifried.org                   alias-seifried
listuser@seifried.org                   listuser
@seifried.org                           mangled-emails
```

The last rule is a catch all so mangled email addresses do not get bounced, and instead sent to a mailbox. Then the aliases file is checked, if an entry is found it does what it says to, otherwise it attempts to deliver the mail to a local users mailbox, my aliases file entry for seifried is:

```
alias-seifried:         seifried, "/var/backup-spool/seifried"
```

This way my email gets delivered to my normal mailbox, and to a backup mailbox (in case I delete an email I really didn't mean to), or god forbid, Outlook decides to puke someday and hose my mailboxes. This would also be useful for corporations, as you now have a backup of all incoming email on a per user basis, and can allow them (or not) to access the file containing the backed up mail.

One caveat, when using a catch all rule for a domain (ie `@seifried.org`) you must create an alias for EACH account, and for mailing lists. Otherwise when it looks through the list and doesn't find a specific entry (for say mailing-list@seifried.org) it will send it to the mailbox specified by the catch all rule. For this reason alone you might not wish to use a catch all rule.

The second method is very simple, you simply start sendmail with the `-X` option and specify a file to log all transactions to. This file will grow very large very quickly, I would NOT recomend using this method to log email unless you absolutely must.

Qmail

Not written yet.

Postfix

Not written yet.

Bind

DNS is an extremely important service for IP networks, I would not hesitate to say probably the **MOST** important network service (without no-one can find anything). It also requires connections coming in from the outside world, and due to the nature and structure of DNS the information DNS servers claim to have may not be true. The main provider of DNS server software (named, the de facto standard) is currently looking at adding a form of DNS information authentication (basically using RSA to cyrptographically sign the data, proving it is 'true').

RedHat has been shipping Bind 4.X up until 5.2, which ships 8.1.2 (finally!), this is a good step, however they should have shipped it setup for non root chroot use by default. A chrooted one that runs as a non root user easily. Making the switch is easy however:

`-u`
specifies which UID bind will switch to once it is bound to port 53 (I like to use a user called 'named' with no login permissions, similar to 'nobody').

`-g`
specifies which GID bind will switch to once it is bound to port 53 (I like to use a group called 'named', similar to 'nobody').

`-t`
specifies the directory that bind will chroot itself to once started. /home/named is a good bet, in this directory you should place all the libraries and config files bind will require.

An even easier way of running bind chroot'ed is to download the bind-chroot package, available in ftp://contrib.redhat.com/ and install it. Before installation you will need a user and group called name (which is what the bind server changes it UID/GID to). Simply use `groupadd` and `useradd` to create the user/group. This package uses holelogd to log bind information to /var/log/messages (as bind would normally do). In addition to this the default configuration file for bind is setup securely (ie you cannot query bind for the version information).

Another aspect of bind is the information it contains about your network(s). When a person queries a DNS server for information they typically send a small request for one piece of information, i.e.: what is the ip address for www.seifried.org? And there are domain transfers, where a dns server requests all the information for say seifried.org, and grabs it and can then make it available to other (in the case of a secondary dns server). This is potentially very dangerous, it can be as or more dangerous then shipping a company phone directory to anyone that calls up and asks for it. Bind version 4 didn't really have much security, you could limit transfers to certain server, but not selectively enough to be truly useful. This has changed in Bind 8, documentation is available at **http://www.isc.org/bind.html**. To make a long story short in Bind 8 there are global settings and most of these can also be applied on a per domain basis. You can easily restrict transfers AND queries, log queries, set maximum data sizes, and so on. Remember, when restricting zone queries you must secure ALL name servers (master and the secondaries), as you can transfer zones from a secondary just as easily as a master.

Here is a relatively secure named.conf file (stolen from the bind-chroot package at ftp.tux.org):

```
options {
        // The following paths are necessary for this chroot
        directory "/var/named";
        dump-file "/var/tmp/named_dump.db";             // _PATH_DUMPFILE
        pid-file "/var/run/named.pid";                  // _PATH_PIDFILE
        statistics-file "/var/tmp/named.stats";         // _PATH_STATS
        memstatistics-file "/var/tmp/named.memstats";   // _PATH_MEMSTATS
        // End necessary chroot paths
        check-names master warn;                        /* default. */
        datasize 20M;
};

zone "localhost" {
        type master;
        file "master/localhost";
        check-names fail;
        allow-update { none; };
        allow-transfer { any; };
};

zone "0.0.127.in-addr.arpa" {
        type master;
        file "master/127.0.0";
        allow-update { none; };
        allow-transfer { any; };
};

// Deny and log queries for our version number except from localhost
zone "bind" chaos {
        type master;
        file "master/bind";
        allow-query {localhost; };
};

zone "." {
        type hint;
        file "named.zone";
};

zone "example.org" {
       type master;
       file "zones/example.org";
      allow-transfer {
                10.2.1.1;
                10.3.1.1;
      };
};
```

WuFTPD

FTP used to be the most used protocol on the Internet by sheer data traffic until it was surpassed by HTTP a few years ago (yes, there was a WWW free Internet once upon a time). FTP does one thing, and it does it well, transferring of files between systems. The protocol itself is insecure, passwords, data, etc is transfered in cleartext and can easily be sniffed, however most ftp usage is 'anonymous', so this isn't a huge problem. One of the main problems typically encountered with ftp sites is improper permissions on directories that allow people to use the site to distribute their own data (typically copyrighted material, etc). Again with telnet you should use an account for ftping that is not used for administrative work.
Problems with ftp in general include:

- Clear text authentication, username and password.
- Clear text of all commands.
- Password guessing attacks (minimal, will end up in the log files)
- Improper server setup and consequent abuse of gained privileges
- Several nasty Denial of Service attacks still exist in WU-FTPD

Securing FTP isn't to bad, between firewalling and tcp_wrappers you can restrict access based on IP address / hostname quite well. In addition ftp runs chrooted by default for anyone logging in as anonymous, or an account defined as guest. With some work amount of work you can set all users that are ftping in to be chrooted to say their home directory. You can also run an ftp daemon that encrypts the data (such as the SSL add on package/etc) however this means your ftp clients must speak the encryption protocol and this isn't always to practical. Also make very sure you have no publicly accessible directories on your ftp server that are both readable and writeable, otherwise people will exploit it to distribute their own stuff (typically warez or porn).
An example of firewalling rules:

```
ipfwadm -I -a accept -P tcp -S 10.0.0.0/8 -D 0.0.0.0/0 21
ipfwadm -I -a accept -P tcp -S some.trusted.host -D 0.0.0.0/0 21
ipfwadm -I -a deny -P tcp -S 0.0.0.0/0 -D 0.0.0.0/0 21

or

ipchains -A input -p tcp -j ACCEPT -s 10.0.0.0/8 -d 0.0.0.0/0 21
ipchains -A input -p tcp -j ACCEPT -s some.trusted.host -d 0.0.0.0/0 21
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -d 0.0.0.0/0 21
```

An example of the same using tcp_wrappers:
In /etc/hosts.allow

```
in.ftpd: 10.0.0.0/255.0.0.0, some.trusted.host
```

And in /etc/hosts.deny

```
in.ftpd: ALL
```

There are several encrypted alternatives to ftp as mentioned before, SSLeay FTPD, and other third party utils. Since most ftp accounts are not used as admin accounts (BAD IDEA!), and

hopefully run chrooted, the security risk is minimized. Now that we have hopefully covered all the network based parts of ftp, lets go over securing the user accounts and environment. The main mechanism for user security in ftpd is the use of chroot. For example by default all people logging in as anonymous have /home/ftp/ set as their 'root'. They cannot get out of this and say look at the contents of /home/ or /etc/. The same can be applied to groups of users and/or individuals, for example you could set all users to be chroot'ed to /home/ when they ftp in, or in extreme cases of user privacy (say on a www server hosting multiple domains) set each user chroot'ed to within their own home directory. This is accomplished through the use of /etc/ftpaccess and /etc/passwd (man ftpaccess has all the info). I will give a few examples of what needs to be done to accomplish this since it can be quite confusing at first. As well ftpd checks /etc/ftpusers and if the user attempting to login is listed in that file (like root should be) it will not let the user login via ftp.

To chroot users as they login into the ftp server is rather simple, but poorly documented. The ftp server check ftpaccess for 'guestgroup's, which are simply "guestgroup some-group-on-the-system" ie "guestgroup badusers", and the groupname needs to be defined in /etc/group and have members added. As well you need to edit their passwd file line so that the ftp server knows where to dump them. And since they are now chrooted into that directory on the system, they do not have access to /lib, etc so you must copy certain files into their dir for things like ls to work properly (always a nice touch).

Setting up a user (billybob) so that he can ftp in, and ends up chroot'ed in his home directory (because he keeps threatening to take the sysadmin possum hunting). In addition to this billybob can telnet in and change his password, but nothing else because he keeps trying to run ircbots on the system. The system he is on uses shadowed passwords, so that's why there is an 'x' in billybob's password field.

First off billybob needs a properly setup user account:
```
billybob:x:500:500:Billy Bob:/home/billybob/./:/usr/bin/passwd
```

this means that the ftp server will chroot billybob into
/home/billybob/ and chdir him into what is now / (/home/billybob to
the rest of us). The ftpaccess man file covers this bit ok, and of
course /usr/sbin/passwd needs to be listed in /etc/shells

Secondly for the ftp server to know that he is being chrooted he needs
to be a member of a group (badusers, ftppeople, etc) that is defined in /etc/group. And then that group must be listed in /etc/ftpaccess.

Now you need to copy some libs/binaries otherwise billybob won't be
able to do a whole lot once he has ftp'ed in. The files needed are in
the "anonftp" rpm package (RedHat 5.1 ships with anonftp-2.5-1). Simply
rpm -i this in and the files will be copied to /home/ftp/lib, you will
notice there is an etc/passwd, this is simply uses to map UID's to
usernames, if you want billybob to see his username and not UID, add
a line for him (ie copy his line from the real /etc/passwd to this one).
The same applies to the group file.

without "billybob:*:500:500:::" in /home/billybob/etc/passwd:
```
drwxr-xr-x   2 500      500          1024 Jul 14 20:46 billybob
```

and with the line added:

```
drwxr-xr-x   2 billybob 500          1024 Jul 14 20:46 billybob
```

and with a line for billybob's group added to the group file:

```
drwxr-xr-x   2 billybob billybob     1024 Jul 14 20:46 billybob
```

Billybob can now ftp into the system, upload and download files
from /home/billybob to his hearts content, change his password
all on his own, and do no damage to the system, nor download
the passwords file or other nasty things.
FTP is also a rather special protocol in that the clients connect to port 21 (typically) on the ftp server, and then port 20 of the ftp server connects to the client and that is the connection that the actual data is sent over. This means that port 20 has to make outgoing connections, keep this in mind when setting up a firewall either to protect ftp servers or clients using ftp. As well there is 'passive' ftp and usually used by www browsers/etc, and involves incoming connections to the ftp server on high port numbers (instead of using 20 they agree on something else). If you intend to have a public ftp server put up a machine that JUST does the ftp serving, and nothing else, preferably outside of your internal LAN (see Practical Unix and Internet Security for discussions of this 'DMZ' concept).

ProFTPD

Not yet written.

BeroFTPD

Not yet written.

tftp

tftp (Trivial File Transfer Protocol) is used for devices that require information from a network server, typically at boot time. It is an extremely simple form of ftp, with most of the security and advanced commands stripped off, it basically allows a device to retrieve (and upload) files from a server in a very simple manner. tftp is almost exclusively used for diskless workstations, router configuration data, and any device that boots up, and requires information it cannot store permanently. As such it presents a rather large security hole, just imagine if someone were to connect to your tftp server and grab the boot file for your main Cisco router. Additionally tftp defaults to granting complete access to the file system, traditionally this meant people would scan large blocks of hosts, and retrieve the `/etc/passwd` file among others. Fortunately modern versions of tftp can be locked down, they accept a directory name that they essentially limit access to, and TCP_WRAPPERS can be used to limit access to certain hosts only. By default tftp (at least for RedHat) defaults to giving access to the `/tftpboot` directory (which usually doesn't exist, so create it if you need it). It is a very good idea to keep the tftp directory as separate form the system as possible. This is done by specifying the directory or directories you want tftp to have access to after the tftp command in `inetd.conf`. Th following example starts tftp normally and grants it access to the `/tftpboot` directory and the `/kickstart` directory.

```
tftp   dgram   udp     wait    root    /usr/sbin/tcpd  in.tftpd /tftpboot
/kickstart
```

Also remember tftp uses UDP, so a '`ps xau`' won't necessarily show who is logged in or what they are doing (as opposed to ftp which shows up) unless they are currently downloading a file (since most tftp applications resolve around small files it is unlikely you will catch someone in the act as it were). The best place to monitor tftp is from syslog, but even then tftp doesn't log ip addresses or anything truly useful.

```
nobody     744  0.0  0.6   780   412  ?  R    14:31   0:00 in.tftpd
/tftpboot
```

Is an example of some ps output, not very useful, and following is some syslog output:

```
Apr 21 14:31:15 sundog tftpd[744]: tftpd: trying to get file: testfile
Apr 21 14:31:15 sundog tftpd[744]: tftpd: serving file from /tftpboot
```

TFTP can be easily restricted using TCP_WRAPPERS and firewalling, tftp runs on port 69, UDP so simply restrict access to that needed by your various diskless workstations, routers and the like, it is also a good idea to block all tftp traffic at your network borders, as there is no need for a machine to remote boot using tftp across the Internet/etc. Also tftp runs as the user nobody, but since no authentication is done and all devices accessing the tftp server are doing so as 'nobody' file level security is pretty well useless. All in all a very insecure protocol.

Apache

What can I say about securing Apache? Not much actually. By default Apache runs as the user 'nobody', giving it very little access to the system, and by and large the Apache team has done an excellent job of avoiding buffer overflows/etc. In general most www servers simply retrieve data off of the system and send it out, most of the danger come not from Apache but from sloppy programs that are executed via Apache (CGI's, server side includes, etc).
If going with Apache I would recommend using the 1.3 series unless you have some strange reason for sticking to 1.2, the active development is now on 1.3, and includes many new features from security, usability, stability and performance viewpoints. Most servers based upon Apache (RedHat Secure Server, Stronghold, etc.) are generally just as bug free but there are occasionally problems.

If you want to be paranoid I would suggest running Apache in a chrooted environment, this however is sometimes more trouble then it is worth. Doing this will break a great many things. You must also install numerous libraries, perl, and any other utilities that your apache server will be using, as well as any configuration files you wish to have access to. Any CGI scripts and other things that interact with the system will be somewhat problematic and generally harder to troubleshoot.

The simplest way to setup apache chrooted is to simply install it and move/edit the necessary files. A good idea is to create a directory (such as `/chroot/apache/`), preferably on a separate filesystem from /, /usr, etc (symlinks, accidental filling of partitions, etc...), and then create a file structure under it for apache. The following is an example, simply replace /chroot-http/ with your choice. You must of course execute these steps as root for it to work. RPM supports this with the --root dir directive, simply install apache and the needed libs using rpm (thus gaining it's support for dependencies/etc, making your life easier).

Apache logs requests and so forth internally, so you don't have to worry about setting up holelogd or any other strangeness in order to get your log files behaving.

About the simplest way to secure apache and insure that it doesn't have unnecessary access to your filesystem is to create a /www/ or similar directory and place ALL the websites, webcontent, cgi's and so forth under it. Then you can simply set access.conf up to deny any access to /, and enable access to /www/ and it's various cgi-bin directories.

Example for access.conf:

```
<Directory />
Options None
AllowOverride None
</Directory>

<Directory /www >
Options Indexes FollowSymLinks Includes
AllowOverride None
</Directory>
```

INN

The usenet server INN has had a long and varied history, for a long period there were no official releases and it seemed to be in a state of limbo, however it is back for good now it would seem. The server software is responsible for handling a potentially enormous load, if you take a full newsfeed the server must process several hundred articles per second, some several kilobytes in size. It must index these articles, write them to disk, and hand them out to clients that request them. INN itself is relatively secure, since it handles data with a directory and generally doesn't have access outside of that, however as with any messaging system if you use it for private/confidential material you must be careful.

One of the main security threats with INN is resource starvation on the server, if someone decides to flood your server with bogus articles, or there is a sudden surge of activity you might be in trouble if capacity is lacking. INN has had several bad security holes in past, but with today's environment the programmers seem to have chased down and eliminated all of them (none have surfaced recently). It is highly recommended (for more then security reasons alone) that you place the news spool on a separate disk system, let alone partition, you might also wish to use ulimit to restrict the amount of memory available so that it cannot bring the server to it's knees.

As for access you should definitely not allow public access, any news server that is publicly accessible will be quickly hammered by people using it to read news, send spam and the like. Restrict reading of news to your clients/internal network, and if you are really worried force people to login. Client access to INN is controlled via the `nnrp.access` file, you can specify ip address(s), domain names and domains (such as *.me.com), as well as there access levels (read and post), the newsgroups they do or don't have access to and you can also specify a username and password, but since this is linked to the host/domain it gets somewhat messy.

example of nnrp.access:

```
*:: -no - : -no- :!*
# denies access from all sites, for all actions (post and read), to all
groups.
*.me.com::Read Post:::*
# hosts in me.com have full access to all groups
*.them.com::Read:::*, !me.*
# hosts in them.com have read access to everything but the me hierarchy
*.aol.com:Read Post:myname:mypassword:*
# give me access from my aol account using a username and password
```

If you are going to run a news server I highly recomend the O'Reilly book "Managing Usenet", as Usenet is similar to Sendmail, a total beast to get running smoothly and keep happy.

D News

Not yet written.

NFSD

NFS stands for Network File System and is just that, it is a good way to distribute filesystems, read only and read/write, while maintaining a degree of security and control assuming your network is enclosed and secure. NFS is primarily meant for use in a high bandwidth environment (ie a LAN) where security risks are not high, or the information being shared is not sensitive (ie a small trusted LAN behind a firewall exchanging CAD/CAM diagrams, or a large university lab using nfs to mount /usr/. If you need a high level of security, ie encrypted data between hosts, NFS is not the best choice. I personally use it at across my internal LAN (this machine has 2 interfaces, guess which one is heavily firewalled), to share file systems containing rpm's, this website, etc. Safer alternatives include SAMBA (free) and now IBM is porting AFS to Linux (costly but AFS is a sweet hunk of code).

NFS has a few rudimentary security controls, the first one would be firewalling, using NFS across a large, slow public network like the Internet just isn't a good idea in any case, so firewall off port 2049, UDP. Since NFS runs as a set of daemons, tcp_wrappers are of no use unless NFS is compiled to support them. The config file for NFS actually has quite a few directives, the bulk of which deal with user id and group id settings (map everyone to nobody, perhaps map all the engineering clients to 'engineer', etc, etc) but no real mechanisms for authentication (your client claims to be UID 0, this is why root's id is squashed by default to nobody). NFS read only exports are pretty safe, you only have to worry about the wrong people getting a look at your info (if it is sensitive) and or creating a denial of service attack (say you have a directory world readable/etc for sharing kernel source, and some gomer starts sucking down data like crazy...). Writeable exports are a whole other ball game, and should be used with extreme caution, since the only 'authentication' is based on IP/hostname (both easily spoofable), and UID (you to can run Linux and be UID 0). Bounce a client down with a DOS attack, grab their IP, mount the writeable share and go to town. You say "but they'd have to know the ip and UID", packet sniffing is not rocket science folks, nor is 'showmount'.
So, how do we go about securing NFS? The first is to firewall it, especially if the machine is multi-homed, with an interface connected to a publicly accessible network (the Internet, the student lab, etc.). If you plan to run NFS over a publicly accessible network it better be read only, and you will be far better off with a different product then NFS. The second and most interesting part is the /etc/exports file. This controls what you allow clients to do, and how they do it.

A sample exports file.

```
#
# Allow a workstation to edit web content
/www                    10.0.0.11(rw,no_root_squash)
#
# Another share to allow a user to edit a web site
/www/www.bobo.org               10.0.0.202(rw,no_root_squash)
#
# Public ftp directory
/home/ftp                       *.example.org(ro,all_squash)
```

The structure of the exports file is pretty simple, directory you wish to export, client (always use ip's, hostnames can easily be faked), and any options. The client can be a single IP (10.0.0.1), hostname (gomer.poncho.net), a subnet (10.0.0.0/255.255.255.0), or a wildcard (*.bigdaddy.mil). Some of the more interesting (and useful) directives for the exports file are:

secure - the nfs session must originate from a privileged port, ie root HAS to be the one trying to mount the dir. This is useful if the server you are exporting to is secured well.

`ro` - a good one, Read Only, enough said.
`noaccess` - used to cut off access, ie export /home/ but do a noaccess on /home/root
`root_squash` - squashes root's UID to the anonymous user UID/GID (usually 'nobody'), very useful if you are exporting dirs to servers with admins you do not 100% trust (root can almost always read any file.... HINT)
`no_root_squash` - useful if you want to go mucking about in exported dirs as root to fix things (like permissions on your www site)
`squash_uids` and `squash_gids` - squash certain UID(s) or GID(s) to the anonymous user, in RedHat a good example would be 500-10000, allowing any users with lower UID's (ie special accounts) to access special things.
`all_squash` - a good one, all privileges are revoked basically and everyone is a guest.
`anonuid` and `anongid` - specifically set the UID / GID of the anonymous user (you might want something special like 'anonnfs').

The `man exports` page is actually quite good.

Beyond this there isn't much you can do to secure NFS apart from ripping it out and putting some other product in (like AFS, Coda, etc). NFS is good, every flavor of UNIX supports it, and is very easy to setup, work with and maintain. It's also 'old faithful', been around a long time. Just check "Practical Unix and Internet Security", they also state in bold not to use NFS if security is a real issue.

Telnetd

Telnet was one of the first services on what is now the Internet, it allows you to login to a remote machine interactively, issue commands and see their results. It is still the primary default tools for remote administration in most environments, and has nearly universal support (even NT has a telnet daemon and client). It is also one of the most insecure protocols, susceptible to sniffing, hijacking, etc. If you have clients using telnet to come into the server you should definitely chroot their accounts if possible, as well as restricting telnet to the hosts they use with tcp_wrappers. The best solution for securing telnet is to disable it and use SSL'ified telnet or ssh.

Problems with telnet include:

- Clear text authentication, username and password.
- Clear text of all commands.
- Password guessing attacks (minimal, will end up in the log files)

The best solution is to turn telnet off and use ssh. This is however not practical in all situations. If you must use telnet then I strongly suggest firewalling it, have rules to allow hosts/networks access to port 23, and then a general rule denying access to port 23, as well as using tcp_wrappers (which is more efficient because the system only checks each telnet connection and not every packet against the firewall rules) however using tcp_wrappers will allow people to establish the fact that you are running telnet, it allows them to connect, evaluates the connection, and then closes it if they are not listed as being allowed in.
An example of firewalling rules:

```
ipfwadm -I -a accept -P tcp -S 10.0.0.0/8 -D 0.0.0.0/0 23
ipfwadm -I -a accept -P tcp -S some.trusted.host/32 -D 0.0.0.0/0 23
ipfwadm -I -a deny -P tcp -S 0.0.0.0/0 -D 0.0.0.0/0 23
```

or in ipchains:

```
ipchains -A input -p all -j ACCEPT -s 10.0.0.0/8 -d 0.0.0.0/0 23
ipchains -A input -p all -j ACCEPT -s some.trusted.host/32 -d 0.0.0.0/0 23
ipchains -A input -p all -j DENY -s 0.0.0.0/0 -d 0.0.0.0/0 23
```

An example of the same using tcp_wrappers:
In /etc/hosts.allow

```
in.telnetd: 10.0.0.0/255.0.0.0, some.trusted.host
```

And in /etc/hosts.deny

```
in.telnetd: ALL
```

There are several encrypted alternatives to telnet as mentioned before, ssh, SSLeay Telnet, and other third party utils, I personally feel that the 'best' alternative if you are going to go to the bother of ripping telnet out and replacing it with something better is to use ssh.
To secure user accounts with respect to telnet there are several things you can do. Number one would be not letting root login via telnet, this is controlled by /etc/securetty and by default in

RedHat Linux 5.X root is restricted to logging on from the console (a good thing). For a user to successfully login their shell has to be valid (and a shell is evaluated against the list in /bin/shells) so setting up user accounts that are allowed to login is simply a matter of setting their shell to something valid, and for keeping users out as simple as setting their shell to /bin/false and making sure that is not listed in /etc/shells. Now for some practical examples of what you can accomplish by setting the user shell to things other then shells.
For an ISP that wishes to allow customers to change their password easily, but not allow them access to the system (my ISP uses Ultrasparcs and refuses to give out user accounts for some reason).

in /etc/shells list:
`/usr/bin/passwd`

and set the users shell to /usr/bin/passwd so you end
up with something like:
username:x:1000:1000::/home/username:/usr/bin/passwd

and voila, the user telnets to the server, logs in, and is
prompted to change his password, if he does so successfully
he gets the boot, if not, he gets the boot anyways and everyone is happy.

Telnet also displays a banner by default when someone connects. This banner typically contains systems information like the name, OS, release and sometimes other detailed information such as the kernel version. Historically this was useful if you had to work on multiple OS's, however in today's hostile Internet it is generally more harmful then useful. Telnetd displays the contents of the file `/etc/issue.net` (typically it is identical to `/etc/issue` which is displayed on terminals and so forth), this file is usually recreated at boot time in most Linux distributions, from the rc.local startup file. Simply edit the rc.local file, either modifying what it puts into `/etc/issue` and `/etc/issue.net`, or comment out the lines that create those files, then edit the files with some static information.

Typical RedHat `rc.local` contents pertaining to `/etc/issue` and `/etc/issue.net`:

```
# This will overwrite /etc/issue at every boot.  So, make any changes you
# want to make to /etc/issue here or you will lose them when you reboot.
echo "" > /etc/issue
echo "$R" >> /etc/issue
echo "Kernel $(uname -r) on $a $(uname -m)" >> /etc/issue

cp -f /etc/issue /etc/issue.net
echo >> /etc/issue
```

simply comment out the lines or remove the uname commands. If you absolutely must have telnet enabled for user logins make sure you have a disclaimer printed:

```
This system is for authorized use only. Trespassers will be prosecuted.
```

something like the above. Legally you are in a stronger position if someone cracks into the system or otherwise abuses your telnet daemon.

POPD/IMAPD

POP and IMAP are fundamental service on the Internet, allowing users to retrieve their email remotely. There exist a great number of POP and IMAP servers (literally dozens for Linux alone). POP is somewhat dated, allowing users to list, retrieve and delete mail, it was meant for one user, with one mailbox, usually attached via a LAN. IMAP is POP on steroids. It allows you to easily maintain multiple accounts, have multiple people access one account, leave mail on the server, just download the headers, or bodies and no attachments, and so on. IMAP is ideal for anyone on the go or with serious email needs. The default POP and IMAP servers that RedHat ships (bundled together into a single package named imapd oddly enough) fulfill most needs.

Unfortunately these two services are run as root, and cannot be easily set to run as a non root user since they have to open mailboxes, so they cannot drops privileges as soon as one would like. Nor can they easily be chrooted. The best policy is to keep up to date with the software, and if at all possible firewall pop and imap from the outside world, this works well if no-one is on the road and needs to collect their email via the Internet. If you have a higher need for security I would suggest looking into alternative IMAP servers such as Cyrus.

POP runs on ports 109 and 110 (109 is basically obsolete though), and IMAP runs on port 143. They also support tcp_wrappers natively, making them relatively easy to lock down. If you can, install a secure www server and a web based email reader on the mail server, this has several advantages for remote users:

- It is more secure, all transactions are encrypted
- Any decent www based mail reader/sender will not cache data, making it safe to use on public terminals
- All you need a browser capable of secure web browsing, which is any computer with an internet connection

Unfortunately I have yet to find any really decent www based mail reading/sending package, they either require PHP3 or are extremely temperamental cgi scripts written in Perl/C/etc. Once I narrow it down I will list 2-3 good ones. Until then I suggest you go to www.freshmeat.net or www.linuxapps.com and search.

Finger

Finger is one of those things most admins just disable and ignore. It is a useful tool on occasion, but if you want to allow other admins to figure out which of your users is currently trying to crack their machines, use identd. Finger lets out way to much info, and is a favorite tool for initial probes and data gathering on targets. There have also been several nasty DOS attacks released, basically consisting of sending hundreds of finger requests and in certain configurations just watching the server croak. Please don't run finger. I don't remember off hand if it is turned off by default in RedHat 5.X, but to quote inetd.conf:

```
# Finger, systat and netstat give out user information which may be
# valuable to potential "system crackers."  Many sites choose to disable
# some or all of these services to improve security.
```

If you still have the urge that you absolutely must run it use -u to deny finger @host requests that are only ever used to gather information for future attacks. Disable finger, really. Fingerd has also been the cause of a few recent and very bad denial of service attacks, especially if you run NIS with large maps, DO NOT, repeat NOT run fingerd.

SSHD

SSH is a secure protocol and set of tools to replace some common (insecure) ones. It was designed from the beginning to offer a maximum of security, and is designed for remote access of servers in a secure manner. SSH can be used to secure any network based traffic, by setting it up as a 'pipe', ie binding it to a certain port at both ends, this is quite kludgy but good for such things as using X across the Internet, in addition to this the server components runs on most UNIX systems, and NT, and the client components runs on pretty much anything. Unfortunately SSH is no longer free, however there is a project to create a free implementation of the SSH protocol.

There aren't any problems with SSH per se like there are with telnet, all session traffic is encrypted except of course and key exchange is done securely (alternatively you can preload keys at either end to prevent them from being transmitted), SSH typically runs as a daemon, and can easily be locked down by using the sshd_config file. You can also run sshd out of inetd, and thus use tcp_wrappers, and by default the ssh rpm's from ftp.replay.com have tcp_wrappers check option compiled into them. Thus using "sshd: blahblah" in hosts.allow and hosts.deny allows you to easily restrict access to ssh. Please note earlier versions of ssh do contain bugs, and sites have been hacked (typically with man in the middle attacks or problems with buffer overflows in the ssh code), but later version of ssh address these problems.

The firewalling rules for ssh are pretty much identical to telnet, and there is of course tcp_wrappers, the problem with tcp_wrappers being that an attacker connects to the port, but doesn't get a daemon, HOWEVER they know that there is something on that port, whereas with firewalling they don't even get a connection to the port. The following is an example of allowing people to ssh from internal machines, and a certain C class on the internet (say the C class your ISP uses for it's dial-up pool of modems).

```
ipfwadm -I -a accept -P tcp -S 10.0.0.0/8 -D 0.0.0.0/0 22
ipfwadm -I -a accept -P tcp -S isp.dial.up.pool/24 -D 0.0.0.0/0 22
ipfwadm -I -a deny -P tcp -S 0.0.0.0/0 -D 0.0.0.0/0 22

or

ipchains -A input -p tcp -j ACCEPT -s 10.0.0.0/8 -d 0.0.0.0/0 22
ipchains -A input -p tcp -j ACCEPT -s isp.dial.up.pool/24 -d 0.0.0.0/0 22
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -d 0.0.0.0/0 22
```

Or via tcp_wrappers:
hosts.allow:
```
sshd: 10.0.0.0/255.0.0.0, isp.dial.up.pool/255.255.255.0
```

hosts.deny:
```
sshd: 0.0.0.0/0.0.0.0
```

In addition to this ssh has a wonderful configuration file, in /etc/sshd/sshd_config by default in the RPM's available on ftp.replay.com. You can easily restrict who is allowed to login, which hosts, and what type of authentication they are allowed to use. The default configuration file is relatively safe but following is a more secure one with explanations. Please note all this info can be obtained by a 'man sshd' which is one of the few well written man pages out there.

```
Port 22
# runs on port 22, the standard
ListenAddress 0.0.0.0
# listens to all interfaces, you might only want to bind a firewall
# internally, etc
HostKey /etc/ssh/ssh_host_key
# where the host key is
RandomSeed /etc/ssh/ssh_random_seed
# where the random seed is
ServerKeyBits 768
# how long the server key is
LoginGraceTime 300
# how long they get to punch their credentials in
KeyRegenerationInterval 3600
# how often the server key gets regenerated
PermitRootLogin no
# permit root to login? hell no
IgnoreRhosts yes
# ignore .rhosts files in users dir? hell yes
StrictModes yes
# ensures users don't do silly things
QuietMode no
# if yes it doesn't log anything. yikes. we wanna log logins/etc.
X11Forwarding no
# forward X11? shouldn't have to on a server
FascistLogging no
# maybe we don't wanna log toto much.
PrintMotd yes
# print the message of the day? always nice
KeepAlive yes
# ensures sessions will be properly disconnected
SyslogFacility DAEMON
# who's doing the logging?
RhostsAuthentication no
# allow rhosts to be used for authentication? the default is no
# but nice to say it anyways
RhostsRSAAuthentication no
# is authentication  using rhosts or /etc/hosts.equiv sufficient
# not in my mind. the default is yes so lets turn it off.
RSAAuthentication yes
# allow pure RSA authentication? this one is pretty safe
PasswordAuthentication yes
# allow users to use their normal login/passwd? why not.
PermitEmptyPasswords no
# permit accounts with empty password to log in? hell no

Other useful sshd_config directives include:
AllowGroups - explicitly allow groups (/etc/group) to login using ssh
DenyGroups - explicitly disallows groups (/etc/groups) from logging in
AllowUsers - explicitly allow users to login in using ssh
DenyUsers - explicitly blocks users from logging in
AllowHosts - allow certain hosts, the rest will be denied
DenyHosts - blocks certain hosts, the rest will be allowed
IdleTimeout time - time in minutes/hours/days/etc, forces a logout
by SIGHUP'ing the process.
```

psst

psst is a free implementation of the SSH protocol, and looks very promising. It is not yet complete but it has most of the functionality provided by SSH. psst is available at: http://www.net.lut.ac.uk/psst/, and I would recomend it as an SSH replacement.

Fresh Free FiSSH

Most of us still have to sit in front of windows workstations, and ssh clients for windows are a pain to find. Fresh Free FiSSH is a free ssh client for Windows 95/NT 4.0, although not yet completed. I would recommend keeping your eye on it though if you are like me and have many Windows workstations, the URL is: http://www.massconfusion.com/ssh/.

RSH, REXEC, RCP

R services such as rsh, rcp, rexec and so forth are **very insecure**. There is simply no other way to state it. Their basis of security is based on the hostname/ip address of the machine connecting, which can easily be spoofed, or using techniques such as DNS poisoning otherwise compromised. By default they are not all disabled, please do so immediately. Edit /etc/inetd.conf and look for rexec, rsh and so on, and comment them out, followed by a "`killall -1 inetd`" to restart inetd.

If you absolutely must run these services use tcp_wrappers to restrict access, it's not much but it will help. Also make sure you firewall them as tcp_wrappers will allow an attacker to see that they are running, which might result in a spoofed attack, something tcp_wrappers cannot defend against if done properly.

If you need remote administration tools that are easy to use and similar to rsh/etc I would recommend nsh (Network SHell) or SSH, they both support encryption, and a much higher level of security. I will be reviewing this product soon and post a link to my review.

SQUID

SQUID is a powerful and fast Object Cache. It proxies FTP and WWW sessions, basically giving it many of the properties of an FTP and a WWW server, but it only reads and writes files within it's cache directory (or so we hope), making it relatively safe. Squid would be very hard to use to actually compromise the system it is on, in addition to it running as a non root user (typically 'nobody'), so generally it's not much to worry about. Your main worry with squid should be improper configuration, for example if squid is hooked up to your internal network (as is usually the case), and the internet (again, very common), it could actually be used to reach internal hosts (even if they are using non routed ip addresses). Hence proper configuration of squid is very important.

The simplest way to make sure this doesn't happen is to use squid's internal configuration and only bind it to the internal interface(s), not letting the outside world attempt to use it as a proxy to get at your internal LAN, in addition to this firewalling it is a good idea. Squid can also be used as an HTTP accelerator, perhaps you have an NT WWW Server on the internal network that you want to share with the world, in this case things get a bit harder to configure but it is possible to do relatively securely. Fortunately squid has very good ACL's (Access Control Lists) built into the squid.conf file, allowing you to lock down access by names, ip's, networks, time of day, actual day (perhaps you allow unlimited browsing on the weekends for people that actually come in to the office). Remember however that the more complicated an ACL is, the slower squid will be to respond to requests.

The ACL's work by defining rules, and then applying those rules, for example:

```
acl internalnet 10.0.0.0/255.0.0.0
http_access allow internalnet
http_access deny all
```

Which defines "internalnet" as being anything with a source of 10.0.0.0/255.255.255.0, allowing it access to the http caching port, and denying everything else. Remember that rules are read in the order given, just like ipfwadm, allowing you to get very complex (and make mistakes if you are not careful). Always start with the specific rules followed by more general rules, and remember to put blanket denials after specific allowals, otherwise it might make it through. It's better to accidentally deny something then to let it though, as you'll find out about denials (usually from annoyed users) faster then things that get through (when annoyed users notice accounting files from the internal www server appearing on the Internet). The squid configuration files (squid.conf) is well commented (to the point of overkill) and also has a decent man page.

DHCPD

DHCPD is something all network admins should use. It allows you to serve information to clients regarding their network settings/etc, typically meaning that the only client setup needed for networking is leaving the defaults and turning the machine on. It also allows you to reconfigure client machines (say move from using 10.0.1.0 to 10.0.2.0). In the long run (and short run) DHCP will save you enormous amounts of work, money and stress. I run it at home with only 8 client machines and have found life to be better even for a LAN this small.
Problems with DHCPD and RedHat 'out of the box':

- Nasty little root hack in previous version, that which ships with 5.2 is ok.
- It runs non chrooted, as root. This is very easy to fix.

I also highly recomend you run DHCPD version 2.X (3.X is in extreme pre alpha stages), it's got a lot of new features, and is easier to setup and work with. The absolute latest version(s) of this tend to be a bit neurotic however, be warned it is beta software. Definitely firewall DHCPD off from the Internet. DHCP traffic should only be on local segments, possibly forwarded to a DHCP server on another segment, but the only DHCP traffic you would see coming over the Internet would be an attack/DOS (they might reserve all your IP's, thus leaving your real clients high and dry). If you are forwarding DHCP traffic over the Internet, DON'T. This is a really bad idea for a variety of reasons (primarily performance / reliability, but security as well).
I recomend the DHCPD server be only a DHCP server, locked up somewhere, allowed to do it's job quietly, if you need to span subnets (ie you have multiple ethernet segments, only one of which has a DHCP server physically connected to it) use a DHCP relay (NT has one built in, the DHCP server has one, etc). There are also several known problems with NT and DHCP, NT RAS has a rather nasty habit of sucking up IP addresses like crazy (I have seen an NT server grab 64 and keep them indefinitely), because it is trying to reserve IP's for the clients that will be dialing in/etc. Either turn NT's RAS off, or put it on it's own subnet, the MAC address it sends to the DHCP server is very strange (and spells out RAS in the first few bytes) and is not easy to map out.
Chroot'ing DHCPD

DHCPD consists of 2 main executables:

- dhcpd - the DHCP
- dhcrelay - a DHCP relay (to relay requests to a central DHCP server since DHCP is based on broadcasts, which typically don't (and shouldn't) span routers/etc.

DHCPD requires 2 libraries:

- /lib/ld-linux.so.2
- /lib/libc.so.6

A config file:

- /etc/dhcpd.conf - configuration info, location of boot files, etc.

And a few other misc. files:

- /etc/dhcpd.leases - a list of active leases
- /etc/functions - a copy of /etc/rc.d/init.d/functions so that one can gracefully 'stop' and 'start' dhcpd

The latest DHCPD in rpm format for RedHat 5.X is at contrib.redhat.com, look for dhcpd-2.X.X, i.e. the latest (currently shipping with RedHat 5.2 is 2.0b1pl6-2).
The simplest way to setup named chrooted is to simply install dhcpd (latest one preferably) and move/edit the necessary files. A good idea is to create a directory (such as /chroot/dhcpd/), preferably on a separate filesystem from /, /usr, etc (symlinks...), and then create a file structure under it for dhcpd. The following is an example, simply replace /chroot/dhcpd/ with your choice. You must of course execute these steps as root for it to work.

```
# Install bind so we have the appropriate files
#
rpm -i dhcpd-2.0b1pl0-1.i386.rpm
#
# Create the directory structure
#
cd /chroot/dhcpd/            # or wherever
mkdir ./etc
mkdir ./usr/sbin
mkdir ./usr
mkdir ./var/dhcpd
mkdir ./var
mkdir ./lib
#
# Start populating the files
#
cp /usr/sbin/dhcpd ./usr/sbin/dhcpd
cp /etc/dhcpd.conf ./etc/dhcpd.conf
cp /etc/rc.d/init.d/dhcpd ./etc/dhcpd.init
cp /etc/rc.d/init.d/functions ./etc/functions
#
# Now to get the latest libraries, change as appropriate
#
cp /lib/ld-linux.ld-linux.so.2 ./lib/
cp /lib/libc.so.6 ./lib/
#
# And create the necessary symbolic links so that they behave
# Remember that named thinks /chroot-dns/ is /, so use relative links
#
# Done, now to manually edit some config files
#
```

where you see:
```
daemon named
```

replace with:
```
cd /chroot-dns/
chroot /chroot-dns/ ./usr/sbin/dhcpd -d -q 2>&1 | tee etc/dhcpd.log &
```

and also get rid of the:
```
# Check that networking is up.
[ ${NETWORKING} = "no" ] && exit 0
```

if your networking isn't up you ain't gonna be serving DHCP requests.

Some notes on file permissions. Since the server is running as 'root', it can do anything it wants. This is why you should put it on a separate partition, to throttle hardlinks/etc. If there is

a way to run dhcpd as non-root easily in RedHat Linux I would be glad to find out since root can escape from chrooted jails.

Once this is done simply remove /etc/rc.d/init.d/dhcpd and create a symlink from /etc/rc.d/init.d/named pointing to /chroot/dhcpd/etc/dhcpd.init, and dhcpd will behave 'normally' while in fact it is separated from your system. You may also wish to remove the 'original' DHCPD files laying about, however this is not necessary.
If you have done the above properly you should have a /chroot/dhcpd/ (or other dir if you specified something different) that contains everything required to run dhcpd.
And a ps -xau should show something like:

```
USER       PID %CPU %MEM  SIZE   RSS TTY STAT START   TIME COMMAND
root      6872  0.0  1.7   900   532  p0 S    02:32   0:00 ./usr/sbin/dhcpd
-d -q
root      6873  0.0  0.9   736   288  p0 S    02:32   0:00 tee
./etc/dhcpd.log
```

bootp

Not yet written.

Identd

The identd service is used to map users/processes to ports in use. For example most irc servers attempt to find out who is connecting to them by doing an identd lookup, which basically consists of asking the identd what information it has about a port number, and can range from nothing (if no-one is using that particular port) to a username, groupname, process id, and other interesting information. The default setting in RedHat is that identd is on, and will only hand out the username. The primary use of identd is to allow remote systems some means of tracking down users that are connecting to their servers, irc, telnet, or other, for authentication purposes (yikes) or more usually for logging purposes. The local university requires identd if you want to telnet into any of the main shell servers, primarily so they can track down cracked accounts. Identd is a useful tool, but generally only on machines with users you do not trust (ie shell account servers). It is also a two edged sword, while it gives out information useful for tracking down attackers (definitely people you want to boot off of your servers) it can also be used to gain information about users on your system, leading to their accounts being compromised. I would suggest only running identd on servers with shell accounts/etc.
Identd supports quite a few features, and can be easily set to run as a non root user. Depending on your security policies you may not want to give out very much information, or you might want to give out as much as possible. Simply tack the option on in inetd.conf, after in.identd (the defaults are -l -e -o).

```
-p port
-a address
```
Can be used to specify which port and address it binds to (in the case of a machine with ip's aliased, or multiple interfaces), this is generally only useful if you want internal machines to connect, since external machines will probably not be able to figure out what port you changed it to.

```
-u uid
-g gid
```
Are used to set the user and group that identd will drop it's privileges to after connecting to the port, this will result in it being far less susceptible to compromising system security. As for handling the amount of information it gives out:

```
-o
```
Specifies that identd will not return the operating system type, and simply say "UNKNOWN", a very good option.

```
-n
```
Will have identd return user numbers (ie UID) and not the username, which still gives them enough information to tell you and allow you to track the user down easily, without giving valuable hints to would be attackers.

```
-N
```
Allows users to make a ~/.noident file, which will force identd to return "HIDDEN-USER" instead of information.

```
-F format
```
Enables you to specify far more information then is standard, everything from user name and number to the actual PID, command name, and command name and arguments that were

given! This I would only recomend for internal use, as it is a lot of information attackers could find useful.

In general I would advise disabling identd, primarily due to the number of denial of service attacks it is susceptible to. You should only run it if you want to make the lives of other administrators easier, in tracking down which of your users are being bad.

Webmin

Webmin is one of the better remote administration tools for Linux, written primarily in Perl it is easy to use and easy to setup. You can assign different 'users' (usernames and passwords are held internally by webmin) varying levels of access, for example you could assign bob access to shutdown the server only, and give john access to create/delete and manipulate users only. In addition to this it works on most Linux platforms and a variety of other UNIX platforms. The main 'problem' with webmin is somewhat poor documentation in some areas of usage, and the fact that the username/password pair are sent in clear text over the network (this is minimized slightly by the ability to grant access to only certain hosts(s) and networks. Most importantly it makes the system more accessible to non technical people who must administer systems in such a way that you do not have to grant them actual accounts on the server. Webmin is available at: http://www.webmin.com/webmin/, and is currently free.

LPD

Not written yet.

xntpd

Not written yet.

cu-snmp

SNMP (Simple Network Management Protocol) was designed to let heterogeneous systems and equipment talk to each other, report data and allow modifications to there settings over a TCP-IP network. For example an SNMP enabled device (such as a Cisco router) can be monitored/configured from an SNMP client, and you can easily write scripts to say alert you if denied packets/second rises above 20. Unfortunately SNMP has no security built into it. SNMPv1, originally proposed in RFC 1157 (May 1990) and section 8 (Security Considerations) reads thusly: "Security issues are not discussed in this memo.". I think that about sums it up. In 1992/1993 SNMPv2 was released, and did contain security considerations however these security considerations were dropped later on when they were shown to be completely broken. Thus we end up today with SNMPv2 and no security. Currently the only way to protect your SNMP devices consists of setting the community name to something hard to guess (but it is very easy to sniff the wire and find the name), and firewall/filter SNMP so that only the hosts that need to talk to each other can (which leaves you open to spoofing). Brute force community name attacks are easy to do and usually effective, and there are several tools specifically for monitoring SNMP transmissions and cracking open an SNMP community, it is a pretty dangerous world out there. These risks are slightly mitigated by the usefulness of SNMP, if properly supported and implemented it can make network administration significantly easier. In almost every SNMP implementation the default community name is "public" (this goes for Linux, NT, etc), you must change this, to something obscure (your company name is a bad idea). Once a person has your community name they can conduct "snmpwalks" and take over your network. SNMP runs over UDP on ports 161 and 162, block this at all entrances to your network (the backbone, the dialup pool, etc). If a segment of network does not have SNMP enabled devices or an SNMP console you should block SNMP to and from that network. This is your only real line of defense with SNMP. Additionally the use of IPSec (or other VPN software) can greatly reduce the risk from sniffing. The RFC's for SNMPv3 however go extensively into security (especially RFC 2274, Jan 1998) so there is hope for the future. If you are purchasing new SNMP aware/enabled products make sure they support SNMPv3, as you then have a chance at real security.

There are no specific problems with cu-snmpd per se, apart from the general SNMP problems I have covered. The cu-snmp tools and utilities only support SNMPv1 and SNMPv2, so remember to be careful when using them on or across untrusted networks as your main line of security (the community name) will be out in the open for anyone to see.

CVS

CVS allows multiple developers to work together on large source code projects and maintain a large code base in a somewhat sane manner. CVS's internal security mechanisms are rather simple (and some would say weak) on their own, and I would have to agree. CVS's authentication is typically achieved over the network using pserver, usernames are sent in clear text, and passwords are trivially hashed (no security really). To get around this you have several good options. In a Unix environment probably the simplest method is to use SSH to tunnel connections between the client machines and the server, "Tim TimeWaster" has written an excellent page covering this that I won't bother to rehash, and it is available at: http://cuba.xs4all.nl/~tim/scvs/. A somewhat more complicated approach (but better in the long run for large installations) is to kerberize the CVS server and clients, typically large networks (especially in university environments) already have an established Kerberoes infrastructure. Details on kerberizing CVS are available at: http://www.cyclic.com/cyclic-pages/security.html. Apart from that I would strongly urge firewalling CVS unless you are using it for some public purpose (such as an open source project across the Internet).

There are other less obvious concerns you should be aware of, when dealing with source code you should be very to ensure no trojan horses or backdoors are emplaced. In an open source project this is relatively simple, review the code people submit, especially if it is a publicly accessible effort, such as the Mozilla project. Other concerns might be destruction of the source code, make sure you have back ups.

rsync

rsync is an extremely efficient method for mirroring files, be it source code files of a CVS tree, a web site, or even this document. rsync preserves file permissions, links, file times and more, in addition to this it supports an anonymous mode (which incidentally I use for the mirroring of this document) that makes life very easy for all concerned. The rsync program itself can act as the client (run from a command line or script) and as the server (typically run from inetd.conf). The program itself is quite secure, it does not require root privileges to run as a client nor as the server (although it can if you really want it to), and can chroot itself to the root directory of whatever is being mirrored (this however requires root privileges and can be more dangerous then it is worth). You can also map the user id and group id it will access the system as (the default is nobody for most precompiled rsync packages and is probably the best choice). In non anonymous mode rsync supports usernames and passwords, that are encrypted quite strongly using 128 bit MD4. The "`man rsyncd.conf`" page quite clearly covers setting up rsync as a server and making it relatively safe. The default configuration file is `/etc/rsyncd.conf`, and has a global section, and module sections (basically each shared out directory is a module).

rsyncd.conf example:

```
motd file = /etc/rsync.motd   # specifies a file to be displayed, legal
disclaimer, etc
max connections = 5           # maximum number of connections so you don't
get flooded
[pub-ftp]
       comment = public ftp area    # simple comment
       path = /home/ftp/pub         # path to the directory being exported
      read only = yes         # make it read only, great for exported
directories
      chroot = yes                 # chroot to /home/ftp/pub
      uid = nobody                 # explicitly set the UID
      gid = nobody                 # explicitly set the GID
[secret-stuff]
      comment = my secret stuff
      path = /home/user/secret           # path to my stuff
      list = no                    # hide this module when asked for a
list
      secrets file = /etc/rsync.users     # password file
      auth users = me, bob, santa         # list of users I trust to see my
secret stuff
      hosts allow = 1.1.1.1, 2.2.2.2      # list of hosts to allow
```

As you can see rsync is quite configurable, and generally quite secure, the exception being the actual file transfers which are not encrypted in any way. If you need security I suggest you use SSH to tunnel a connection, or some VPN solution like FREES/WAN.

Samba

SAMBA is one of the best things since sliced bread, that is if you want to share files and printers between Windows and *NIX. It is also somewhat misunderstood, and suffers heavily from interaction with various (sometimes broken) Windows clients. SAMBA has a great many kludges that attempt to make it somewhat sane, but can lead to what looks like broken behavior sometimes. SAMBA simply gives access to the filesystem VIA SMB (Server Message Block), the protocol Windows uses to share files and printers. It verifies the username and password given (if required) and then gives access to the files according to the file permissions and so forth that are set. I'm only going to cover Samba 2.0, Samba 1.X is pretty old and obsolete.

Samba 2.0 is controlled via `smb.conf`, typically in `/etc (man smb.conf)`. In `/etc/smb.conf` you have 4 main areas of configuration switches: `[globals]`, `[printers]`, `[homes]`, and each `[sharename]` has it's own configuration (be it a printer or drive share). There are a hundred or so switches, the smb.conf man page covers them exhaustively. Some of the important (for security) ones are:

`security = xxxx` where `xxxx` is share, server or domain, share security is per share, with a password that everyone uses to get at it, server means the samba server itself authenticates users, either via `/etc/password`, or `smbpassword`. If you set it to domain, samba authenticates the user via an NT domain controller, thus integrating nicely into your existing NT network (if you have one).

`guest account = xxxx` where `xxxx` is the username of the account you want the guest user to map to. If a share is defined as public then all requests to it are handled as this user.

`hosts allow = xxxx` where `xxxx` is a space separated list of hosts / ip blocks allowed to connect to the server.

`hosts deny = xxxx` where `xxxx` is a space separated list of hosts / ip blocks not allowed to connect to the server.

`interfaces = xxxx` where `xxxx` is a space separated list of ip blocks that samba will bind to

I would also highly recomend installing and using SWAT (samba Web Administration Tool) as it will cut down on the mistakes/etc that you are liable to make. Samba and SWAT are available at: http://www.samba.org/ and ship with almost every distribution.

SWAT

Not yet written.

Novell

Not yet written. (I'd like a copy of OpenLinux 2.2)

Network Based Authentication

NIS / NIS+

Not yet written.

Kerberoes

Not yet written.

Radius

Not yet written.

Firewalling

Firewalling is the practice of filtering TCP-IP traffic, typically at the point where your network connects to another (i.e. the Internet, a customers LAN or other) network, that may be untrusted (in the case of the Internet) or perhaps even trusted (another floor of your building). Like firewalls in a large building, a network firewall can prevent and even block the spread of an attack.

Linux has had firewalling capacity for quite a while now in the form of ipfwadm, which was a very simplistic packet level filter, but quite effective and good enough for most people. With the advent of kernel 2.1+ this has been replaced with ipchains which is quite a bit more sophisticated. Both are still basic packet filters however and do not allow for advanced features such as stateful inspection or some types of proxying connections, however Linux does support IPMASQ, an advanced form of NAT (Network Address Translation). IPMASQ allows you to hook up a network of computers to the Internet, but proxy their connections at the packet level, thus all traffic appears to be coming and going to one machine (the Linux IPMASQ box), which affords a high degree of protection to the internal network. As an added bonus the clients on the internal network require NO proxy configuration, as long as the Linux IPMASQ server is configured correctly things will work quite well.

Both ipchains and ipfwadm provide the following basic capabilities:

- blocking / allowing data to pass through based on IP/port/interface source / destination
- masquerading of connections, based on IP/port/interface source / destination

In addition to which ipchains supports:

- port forwarding
- creation of chains, for more intricate rules and conditions, easier to maintain
- quality of service (QOS) routing, useful on low speed connections or otherwise saturated connections
- specification of IP/port/interface as well as inverse specification (using the !)

The Firewall-HOWTO and "`man <command>`" (ipchains or ipfwadm) page both cover in great detail the mechanics for setting up rules, but don't really cover the strategy for firewalling safely. Your first choice to make is whether to go with default deny or default allow policies, followed by which services and hosts you wish to allow and block.

When deciding policy you should choose a policy that will default to denying everything unless specifically allowed through (that is if there is a failure it will hopefully be minimized via default policies) or a policy that allows everything and blocks certain services/hosts. I typically use a policy of default denial as it can accommodate mistakes and changes more safely then a policy that defaults to allowing data through. Case in point, you have a server secured via firewalling, currently running apache, you install Wu-FTPD on it for internal use (so people can upload files) at 3 am, you forget to change the firewall rules. If you have chosen a policy of default allowal anyone on the Internet can access the ftp server, and silly you, you installed an old version which allowed someone to compromise the machine. If on

the other hand you go with a policy of default denial they would not have access to the ftp server, and neither would your users, but you would find out quite quickly.

I have decided to not cover specific firewalling rules in this section, for each network service I will provide examples, as to properly firewall a protocol you need to understand how it behaves. There is a huge difference between firewalling www and ftp for inbound and outbound access for example. Some general concepts/rules:

IPFWADM

Ipfwadm is a solid packet filter for Linux, although it lacks a lot of features available in Ipchains. Ipfwadm only supports 3 targets for a packet, accept deny or reject, whereas ipchains rules can be targeted at 6 built in targets, or a user defined target. Ipfwadm is really only appropriate for a simple ip level firewall, ipmasquerading and if you plan to use FREES/WAN (which currently does not support kernel 2.2.x). The basic options are: specify a direction (in out or both, useful with the interface flag), input rules, output rules, forwarding rules (say you have multiple interfaces, also covers the masquerading rules) and masquerade rules which control the behavior of masquerading (timeouts, etc). You can insert, append and delete rules, set default policies, and list all the rules, unlike ipchains you only have the 3 targets (ACCEPT, DENY, REJECT). Other then that it is very similar to ipchains, with some minor variations. The following is a script appropriate for a server bridging 2 networks (10.0.0.x on eth0, 10.0.0.1 and 192.168.0.x on eth1, 192.168.0.1) with a mail server running.

```
#!/bin/bash
#
# Flush all the rule sets first
#
ipfwadm -f -I
ipfwadm -f -O
ipfwadm -f -F
#
# Allow forwarding between the two networks and otherwise deny it for
security
#
ipfwadm -F -a accept -P all -S 10.0.0.0/24 -i eth0 -D 192.168.0.0/24
ipfwadm -F -a accept -P all -S 192.168.0.0/24 -i eth1 -D 10.0.0.0/24
ipfwadm -F -p deny
#
# And of course we have to allow those packets in
#
ipfwadm -I -a accept -P tcp -S 10.0.0.0/24 -i eth0 -D 192.168.0.0/24
ipfwadm -I -a accept -P tcp -S 192.168.0.0/24 -i eth1 -D 10.0.0.0/24
#
# Let them access the mail server port on the server but nothing else
#
ipfwadm -I -a accept -P tcp -S 10.0.0.0/24 -i eth0 -D 10.0.0.1 25
ipfwadm -I -a accept -P tcp -S 192.168.0.0/24 -i eth0 -D 192.168.0.1 25
ipfwadm -I -p deny
```

The only time you should use ipfwadm instead of ipchains is if you plan to run FREES/WAN, and as soon as FREES/WAN supports kernel 2.2.x I'd advise moving. There are patches for kernel 2.0.x supposedly that implement ipchains, but the changes between 2.0.x and 2.2.x are quite drastic so I would recommend thorough testing first.

IPCHAINS

Several new things in IPCHAINS, you can create chains of rules (hence the name) and link them together, making administration of firewalls far easier. Ipchains supports more targets then ipfwadm, you can point a rule at: ACCEPT, DENY, REJECT, MASQ, REDIRECT, or RETURN or a user defined chain. As such it is very powerful, for example I could redirect all packets bound for port 80 (i.e. any www traffic) going through my gateway machine to be redirected to local port 3128, the squid proxy server. You can also use this in conjunction with quality of service routing, the example given in ipfwadm's documentation is that of prioritizing traffic going over a PPP link, you can give telnet traffic a much higher priority then say ftp, reducing latency problems caused by a saturated link. Typically I create an /etc/rc.d/init.d/ipchains-sh (or wherever appropriate) and call it immediately after the networking is brought up, this leaves a small time in which the server is vulnerable, but minimally so since no network daemons are running. The following script is appropriate for a gateway with 2 interfaces running, the reason I have used the DENY instead of REJECT target is so that the packet is dropped and not responded to in any way, this slows down network scans (as they wait for the packet to timeout instead of receiving a response) and gives away less information. I would also advise against logging data unless you have a significant amount of drive space available, for each packet I send (several bytes) many bytes of drive space is used up to create a log entry, making it easy to overwhelm syslog and/or your drive space on a fast connection.

```
#!/bin/bash
#
# This script sets up firewall rules appropriate for a server with 2
interfaces
# running as a gateway
# This script needs to be edited if you plan to use it.
# We assume the internal machines call all talk to the gateway, so no rules
block
# internal traffic
#
# A couple of variables
#
# ETH0 is the IP address on ETH0 (the external interface)
# ETH0NET is the network
# ETH0NETMASK is the network mask
# TRUSTEDHOST1 is a trusted host (for webmin/ssh)
# TRUSTEDHOST2 is a trusted host (for webmin/ssh)
# ETH1IP is the IP address on ETH1 (internal interface)
# ETH1NET is the network
# ETH1NETMASK is the network mask
#
ETH0IP=1.1.1.1
ETH0NET=1.1.1.0
ETH0NETMASK=24
TRUSTEDHOST1=1.5.1.1
TRUSTEDHOST2=1.5.1.2
ETH1IP=10.0.0.1
ETH1NET=10.0.0.0
ETH1NETMASK=24
#
PATH=/sbin
# FLUSH ALL RULES
ipchains -F input
ipchains -F output
```

```
ipchains -F forward
# ANTI-SPOOFING
ipchains -A input -p all -j DENY -s 10.0.0.0/8 -i eth0 -d 0.0.0.0/0
ipchains -A input -p all -j DENY -s 127.0.0.0/8 -i eth0 -d 0.0.0.0/0
ipchains -A input -p all -j DENY -s 192.168.0.0/16 -i eth0 -d 0.0.0.0/0
ipchains -A input -p all -j DENY -s 172.16.0.0/16 -i eth0 -d 0.0.0.0/0
ipchains -A input -p all -j DENY -s $ETH0IP -i eth0 -d 0.0.0.0/0
# ICMP FIRST
ipchains -A input -p icmp -j ACCEPT -s $ETH0NET/$ETH0NETMASK -i eth0 -d
0.0.0.0/0
ipchains -A input -p icmp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0
# SSH
ipchains -A input -p tcp -j ACCEPT -s $TRUSTEDHOST1 -i eth0 -d 0.0.0.0/0 22
ipchains -A input -p tcp -j ACCEPT -s $TRUSTEDHOST2 -i eth0 -d 0.0.0.0/0 22
# BLOCKING 1:1023
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 1:1023
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 1:1023
# BLOCKING OTHER THINGS
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 1109
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 1524
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 1600
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 2003
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 2049
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 2105
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 3001
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 3001
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0
3128:3130
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0
3128:3130
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 3306
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 3306
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 4444
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0
6000:6100
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0
6000:6100
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 6667
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 7000
# WEBMIN
ipchains -A input -p tcp -j ACCEPT -s $TRUSTEDHOST1 -i eth0 -d 0.0.0.0/0
10000
ipchains -A input -p tcp -j ACCEPT -s $TRUSTEDHOST2 -i eth0 -d 0.0.0.0/0
10000
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i eth0 -d 0.0.0.0/0 10000
# FORWARD RULES
ipchains -P forward DENY
ipchains -A forward -p all -j MASQ -s $ETH1NET/$ETH1NETMASK -d 0.0.0.0/0
```

Encrypting services / data

Encrypting network services

Virtually all network traffic is unencrypted and easily read by an attacker. If someone cracks a machine on your internet and installs a password sniffer (basically your common packet sniffer with a filter) your entire network can be compromised in a matter of hours. One ISP that shall remain nameless placed co-hosted customer machines on the same LAN, using a normal ethernet hub, meaning all machines could see each others traffic (users retrieving email via pop, telnet sessions, etc). This is one of the major reasons why encrypting data traffic is a good idea.

Various mechanisms exist and/or are being developed to encrypt network data traffic, at various levels of the network stack. Some schemes only encrypt the data sent (such as pgp encrypted email), some encrypt the session (SSL), and some encrypt the data payload of the packets (IPSec and other VPN's). Ultimately the best solution will be IPSec (my opinion), as it requires no modifications to the applications, and provides for a very high level of security between computers. Currently there are no widely used data encryption solutions, as Microsoft does not support many, which is a serious hindrance to any mass solution (to be fair Microsoft does have beta IPSec support, but it is not ready yet). The best scheme currently available is SSL, Secure Sockets Layer, originally proposed by Netscape. SSL encrypts the data at the session level, thus is your application supports SSL and the server supports SSL you are in luck. Most www browsers, some email/news readers, and a few ftp and telnet clients support SSL currently, for Linux most services can be SSL'ified. SSL does however require clients with SSL capabilities, something you won't be able to get most people to support, which means SSL'ified services are typically restricted to within an organization. The SSL libraries are available at http://www.openssl.org/.

SSL

HTTP - SSL

The most common www server, Apache, has very good SSL support, which can be downloaded free of charge outside the US (US patents on RSA/etc mean you have to pay royalties within the US, so the free software is illegal) from http://www.apache-ssl.org. There are many commercial www servers that support SSL, the majority also based off of Apache, such as RedHat Secure Server, Stronghold, and so forth.

Telnet - SSL

A drop in replacement for telnet, SSLtelnet and MZtelnet provide a much higher level of security then plain old telnet, although SSLtelnet and MZtelnet are not as flexible as SSH, they are perfectly free (ie. GNU licensed) which SSL is not. The server and client packages are available as tarballs at: ftp://ftp.uni-mainz.de/pub/internet/security/ssl/, and as RPM packages at ftp://ftp.replay.com/pub/replay/linux/redhat/.

FTP - SSL

Also a drop in replacement for your favourite ftpd (probably WuFTPD), also available as a set of patches for WuFTPD. This is highly appropriate as most servers have many users that

require ftp access. The tarball is available at: ftp://ftp.uni-mainz.de/pub/internet/security/ssl/, and as RPM packages at ftp://ftp.replay.com/pub/replay/linux/redhat/.

Virtual Private Network Solutions

IPSec

IPSec is a proposed (and now implemented in some operating systems) standard for IP level encryption. Linux IPSec support varies, with some distributions shipping it, but the majority do not (US export laws are one major reason). IPSec operates using public key cryptography, keys are exchanged and keyservers are also available (pluto, photuris, etc.). One major problem with IPSec is that the protocol has not been fully ratified as of yet, so implementations can vary. The main Linux IPSec effort (FREES/WAN) is hosted at: http://www2.kame.net/~freeswan/ and since it is headed up by Canadians (my home and native land) it is fully exportable to anywhere in the world that allows the use of strong crypto (exceptions being France, China, Libya, etc). They just released version 1.0, and it is quite excellent, the only 'problem' with it (in my opinion) is that it does not yet support 2.1 or 2.2 kernels, these will require some major changes, but the group has it as their next major goal.

PPTP (Point to Point Tunneling Protocol)

PPTP is a proprietary protocol created by Microsoft for VPN solutions. To date it has been shown to contain numerous and massive flaws. However if you need to integrate Linux into a PPTP environment all is not lost, http://www.moretonbay.com/vpn/pptp.html contains a Linux implementation of PPTP.

CIPE (Cyrpto IP Encapsulation)

CIPE is a free IP level encryption scheme, meant for use between routers. It is appropriate for 'bridging' networks securely together over insecure networks (like the Internet). The official cite for CIPE is at: http://sites.inka.de/~W1011/devel/cipe.html. I would however recommend FREES/WAN as a better long term solution.

ECLiPt Secure Tunnel (currently in beta)

Another GNU licensed solution for Linux VPN's. Currently in beta (and not recommended for mass use) but I thought I should mention it anyways since it seems to be a serious effort. The official page is at: http://eclipt.uni-klu.ac.at/projects/est/. Again I would have to recommend FREES/WAN as a better long term solution.

Encrypting Data

Several encryption programs are also available to encrypt your data, some at the file level (PGP, GnuPG, etc.) and some at the drive level (Cryptographic File System for example). These systems are very appropriate for the storage of secure data, and to some degree for the transmission of secure data (although both ends will require the correct software, compatible versions, and an exchange of public keys will somehow have to take place) which is unfortunately an onerous task for most people. In addition to this you have no easy way of trusting someone's public key unless you receive it directly from them (such as at a key signing party), or unless it is signed by someone else you trust (but how do you get the trusted

signer's key securely?). Systems for drive encryption such as CFS (CryptoGraphic Filesystem) are typically easy to implement, and only require the user to provide a password or key of some form to access their files.

PGP (Pretty Good Privacy)

The granddaddy of public encryption, this is by far one of the most popular programs as it is supported under Unix, Windows and Macintosh. Unfortunately it has now been commercialized, which has resulted in a loss of quality for users. I personally believe any software used to encrypt or otherwise secure data MUST be open source or how else can you be sure it is secure. PGP is now sold by Network Associates and I cannot in good faith recommend it as a security mechanism for the secure storage and transmission of files.

GnuPG (Gnu Privacy Guard)

The alternative to PGP, a direct replacement that is fully opensource and GNU licensed (as if the name didn't give it away). This tool is available at: http://www.gnupg.org, as source code or precompiled binaries for windows, and RPM's.

CFS (CryptoGraphic Filesystem)

CFS allows you to keep data on your harddrive in an encrypted format, and is significantly easier to use then a file encryption program (such as PGP) if you have many files and directories you want to keep away from curious people. The official distribution site is at: http://www.cryptography.org/, and RPM's are available at: ftp://ftp.replay.com/pub/replay/linux/redhat/, and Debian binaries are at: http://www.debian.org/Packages/stable/otherosfs/cfs.html.

Scanning and intrusion testing tools

Over the last few years the number of security tools for Windows and UNIX has risen dramatically, even more surprising is the fact that most of them are freely available on the Internet. I will only cover the free tools since most of the commercial tools are ridiculously expensive, are not open source, and in many cases have been shown to contain major security flaws (like storing passwords in clear text after installation). In any case any serious cracker/hacker will have these tools at their disposal, so why shouldn't you?

There are several main categories of tools, ones that scan hosts from within that host, ones that scan other hosts and report back variously what OS they are running (using a technique called TCP-IP fingerprinting), services that are available and so on, at the top of the food chain are the intrusion tools that actually attempt to execute exploits, and report back if they worked or not, lastly I include the exploits category, while not strictly an intrusion tool per se they do exist and you should be aware of them.

Network scanners

Strobe

Strobe is one of the older port scanning tools, quite simply it attempts to connect to various ports on a machine(s) and reports back the result (if any). It is simple to use and very fast, but doesn't have any of the features newer port scanners have. Strobe is available for almost all distributions as part of it, or as a contrib package, the source is available at: ftp://suburbia.net/pub/.

Nmap

Nmap is a newer and much more featured host scanning tool. It features advanced techniques such as TCP-IP fingerprinting, a method by which the returned TCP-IP packets are examined and the host OS is deduced based on various quirks present in all TCP-IP stacks. Nmap also supports a number of scanning methods, from normal TCP scans (simply trying to open a connection as normal) to stealth scanning and half open SYN scans (great for crashing unstable TCP-IP stacks). This is arguably one of the best port scanning programs available, commercial or otherwise. Nmap is available at: http://www.insecure.org/nmap/index.html.

Intrusion Scanners

Nessus

Nessus is relatively new, but is fast shaping up to be the best intrusion scanning tool. It has a client/server architecture, the server currently only runs on Linux, and clients are available for Linux, Windows and there is a Java client. Communication between the server and client is ciphered for added security, all in all a very slick piece of code. Nessus supports port scanning, and attacking, based on IP addresses or host name(s). It can also search through network DNS information and attack related hosts at your bequest. Nessus is relatively slow in attack mode, which is hardly surprising, but it currently has over 200 attacks, and a plug in language so you can write your own. Nessus is available from http://www.nessus.org/.

Saint

Saint is the sequel to Satan, a network security scanner made (in)famous by the media a few years ago (there were great worries that bad people would take over the Internet using it). Saint also uses a client/server architecture, but uses a www interface instead of a client program. Saint produces very easy to read and understand output, with security problems graded by priority (although not always correctly), and also supports add in scanning modules making it very flexible. Saint is available from: http://www.wwdsi.com/saint/.

Cheops

While not a scanner per se, it is useful for detecting a hosts OS and dealing with a large number of hosts quickly. Cheops is a "network neighborhood" on steroids, it builds a picture of a domain, or ip block, what hosts are running and so on. It is extremely useful for preparing an initial scan as you can locate interesting items (HP printers, Ascend routers, etc) quickly. Cheops is available at: http://www.marko.net/cheops/.

Exploits

I won't cover exploits specifically, since there are hundreds if not thousands of them floating around for Linux. I will simply cover the main archival sites.

http://www.rootshell.com/

One of the primary archive sites for exploits, it has almost anything and everything, convenient search engine and generally complete exploits.

http://www.geek-girl.com/bugtraq/

Not a site for exploits per se, it is the archive of the Bugtraq mailing list, which carries security information/bulletins and being a full disclosure list often has the exploits. I also suggest you sign up to this list, there are something like 30,000+ subscribers currently, not all of them nice people.

Scanning and intrusion detection tools, packet sniffing

If the last section has you worried you should be. There are however many defenses, active and passive against those types of attacks. The best ways to combat network scans are keep software up to date, only run what is needed, and heavily restrict the rest through the use of firewalls and other mechanisms. Luckily in Linux these tools are free and easily available, again I will only cover opensource tools, since the idea of a proprietary firewall/etc is rather worrying. The first line of defense should be a robust firewall, followed by packet filters on all Internet accessible machines, liberal use of TCP-WRAPPERS, logging and more importantly automated software to examine the logs for you (it is unfeasible for an administrator to read log files nowadays).

Host based attack detection

Firewalling

Most firewalls support logging of data, and ipfwadm/ipchains are no exception, using the -l switch you get a syslog entry for each packet, using automated filters (Perl is good for this) you can detect trends/hostile attempts and so on. Since most firewalls (UNIX based, and Cisco in any case) log via the syslog facility, you can easily centralize all your firewall packet logging on a single host (with a lot of harddrive space hopefully).

TCP-WRAPPERS

Wietse's TCP-WRAPPERS allow you to restrict connections to various services based on IP address and so forth, but even more importantly it allows you to configure a response, you can have it email you, finger the offending machine, and so on (use with caution however).

Klaxon

While mostly obsoleted by TCP-WRAPPERS and firewall logging, klaxon can still be useful for detecting port scans/etc if you don't want to totally lock down the machine. Klaxon is available at: ftp://ftp.eng.auburn.edu/pub/doug/.

Host Sentry (pre release software)

While this software is not yet ready for mass consumption I thought I would mention it anyways as it is part of a larger project (the Abacus project, http://www.psionic.com/abacus/). Basically Host Sentry builds a profile of user accesses and then compares that to current activity in order to flag any suspicious activity. Host Sentry is available at: http://www.psionic.com/abacus/hostsentry/.

Logcheck

This software is also part of the Abacus project, but is very mature and actually ships with some distributions. It scans log files, and based on criteria you choose emails you a report of anything strange/out of the ordinary. It is a very efficient tool as it cuts out the chaff and only shows you useful information from your log files, in addition to this it emails you reports, making it very portable and easy to use.

Port Sentry (beta)

The third component to the Abacus suite, it detects and logs port scans, including stealthy scans (basically anything nmap can do it should be able to detect). Port Sentry can be configured to block the offending machine (in my opinion a bad idea as it could be used for a denial of service attack on legitimate hosts), making completion of a port scan difficult. As this tool is in beta I would recommend against using it, however with some age it should mature into a solid and useful tool. Port Sentry is available at: http://www.psionic.com/abacus/portsentry/.

Network based attack detection

NFR

NFR (Network Flight Recorder) is much more then a packet sniffer, it actually logs data and in real time detects attacks, scans and so on. This is a very powerful tool and requires a significant investment of time, energy and machine power to run, but it is at the top of the food chain for detection. NFR is available at: http://www.nfr.com/.

Packet sniffers

Packet sniffing is the practice of capturing network data not destined for your machine, typically for the purpose of viewing confidential/sensitive traffic such as telnet sessions or people reading their email. Unfortunately there is no real way to detect a packet sniffer since it is a passive activity, however by utilizing network switches and fiber optic backbones (which are very difficult to tap) you can minimize the threat.

tcpdump

The granddaddy of packet sniffers for Linux, this tool has existed as long as I can remember, and is of primary use for debugging network problems. It is not very configurable and lacks advanced features of newer packet sniffers, but it can be useful. Most distributions ships with tcpdump.

sniffit

My favourite packet sniffer, sniffit is very robust, has nice filtering capabilities, will convert data payloads into ASCII text for easy reading (like telnet sessions), and even has a graphical mode (nice for monitoring overall activity/connections). Sniffit is available at: http://sniffit.rug.ac.be/sniffit/sniffit.html.

Other sniffers

There are a variety of packet sniffers for Linux, based on the libpcap library among others, here is a short list:

http://www.mtco.com/~whoop/ksniff/ksniff.html - KSniff
http://ksniffer.veracity.nu/ - Ksniffer
http://mojo.calyx.net/~btx/karpski.html - karpski
http://www.ozemail.com.au/~peterhawkins/gnusniff.html - Gnusniff
http://elektra.porto.ucp.pt/snmpsniff/ - SNMP Sniffer

Conducting baselines

One major oversight made by a lot of people when securing their machines is that they forget to create a baseline of the system, that is a profile of the system, it's usage of resources, and so on in normal operation. For example something as simple as a "`netstat -a -n > netstat-output`" can give you a reference to latter check against and see if any ports are open that should not be. Memory usage and disk usage are also good things to keep an eye on, a sudden surge in memory usage could result in the system being starved of resources, likewise for disk usage, it might be a user accident, a malicious user, or a worm program that has compromised your system and is now scanning other systems. Various tools exist to measure memory and disk usage: `vmstat`, `free`, `df`, `du`, all of which are covered by their respective man pages.

At the very minimum make a full system backup, and regularly backup config files and log files, this can also help you pinpoint when an intrusion occurred (user account "rewt" was added after the April 4th backup, but isn't in the March 20th backup). Once a system is compromised typically a "rootkit" is installed, these consist of trojaned binaries, and are near impossible to remove safely, you are better of formatting the disk and starting from scratch. There is of course a notable exception to this rule, if you were diligent and used file/directory integrity tools such as tripwire you will be able to pinpoint the affected files easily and deal with them (unfortunately tripwire is no longer free). There is an alternative to tripwire however, L5, available at: ftp://avian.org/src/hacks/L5.tgz, unfortunately tripwire was turned over to a commercial company and is ridiculously expensive now. Another simple tool to use is diff, you can directly compare binary files (this is rather slow however), so given a known good backup you can find what is bad relatively quickly.

Conducting audits

So you've secured your machines, and done all the things that needed to be done. So how do you make sure it's actually doing what it is supposed to do, or prove to someone that it is as secure as you say it is? Well you conduct an audit. This can be as simple as reviewing the installed software, configuration files and other settings, or as complex as putting together or hiring a tiger team (or ethical hackers, or whatever buzzword(s) you prefer) to actively try and penetrate your security. If they can't then you did your job well (or they suck), and if they do get in, you know what needs to be fixed (this is also a good method to get an increased security budget, show how vulnerable you are to the CIO).

There are also many free tools and techniques you can use to conduct a self audit and ensure that the systems react as you think they should (we all make errors, but catching them quickly and correcting them is part of what makes a great administrator). Tools such as nmap, nessus, crack, and so forth can be quickly employed to scan your network(s) and host(s), finding any obvious problems quickly. I also suggest you go over your config files every once in a while (for me I try to 'visit' each server once a month, sometimes I discover a small mistake, or something I forgot to set previously). Keeping systems in a relative state of synchronization (I just recently finished moving ALL my customers to Kernel 2.2.X, ipchains) which will save you a great deal of time and energy.

Backups

I don't know how many times I can tell people, but it never ceases to amaze me how often people are surprised by the fact that if they do not backup their data, if the drive craters out on them, or they hit 'delete' without thinking it will be gone. Always backup your system, even if it's just the config files, you'll save yourself time and money in the long run.
To backup your data under RedHat there exist many solutions, all with various pro's and con's. There are also several industrial strength backup programs, the better ones support network backups which are a definite plus in a large environment.

Tar and Gzip

My personal favorite, tar and gzip. Why? Because like vi you can darn near bet the farm on the fact that any UNIX system will have tar and gzip. They may be slow, klunky and starting to show their age, but it's a universal tool that will get the job done. I find with RedHat Linux installation of a typical system takes 15-30 minutes depending on the speed of the network/cdrom, configuration another 5-15 (assuming I have backups or it is very simple) and data restoration takes as long as it takes (definitely not something you should rush). Good example: I recently backed up a server and then proceeded to blow the filesystem away (and remove 2 physical HD's that I no longer needed), I then installed RedHat 5.2, and reconfigured all 3 network cards, apache (for about 10 virtual sites), Bind and several other services in about 15 minutes. If I had done it from scratch it would have taken me several hours. Simply:

```
tar -cvf archive-name.tar dir1 dir2 dir3....
```

to create the tarball of all your favorite files (typically /etc, /var/spool/mail/, /var/log/, /home, and any other user/system data), followed by a:

```
gzip -9 archive-name.tar
```

to compress it as much as possible (granted harddrive space is cheaper then a politicians promise but compressing it makes it easier to move around). You might want to use bzip, which is quite a bit better then gzip at compressing text, but it is quite a bit slower. I typically then make a copy of the archive on a remote server, either by ftping it or emailing it as an attachment if it's not to big (ie the backup of a typical firewall is around 100k or so of config files).

Commercial Backup Programs for Linux

BRU

RedHat ships with a nice backup program (well a demo version anyways) called BRU (**B**ackup and **R**estore **U**tility), this thing has been in the Linux world since as long as Linux Journal (they have had ads in there since the beginning). This program affords a relatively complete set of tools in a nice unified format, with command line and a graphical front end (easy to automate in other words). It supports full, incremental and differential backups, as well as catalogs, and can write to a file or tape drive, basically a solid, simple, easy to use backup program, and it ships with RedHat Linux (one license), so if you bought RedHat you should have a copy to play with. BRU is available at http://www.estinc.com/.

Quickstart

Quickstart is more aimed at making an image of the system so that when the hard drive fails/etc. you can quickly re-image a blank disk and have a working system. It can also be used to 'master' a system and then load other systems quickly (as an alternative to say RedHat's kickstart). It's reasonably priced as well and garnered a good revue in Linux Journal (Nov 1998, page 50). You can get it at: http://www.estinc.com/.

Backup Professional

Not written yet.

CTAR

Not written yet.

CTAR:NET

Not yet written.

Arkeia

Arkeia is a very powerful backup program with a client - server architecture that supports many platforms. This is an 'industrial' strength product and appropriate for heterogeneous environments, it was reviewed in Linux Journal (April 1999, page 38) and you can download a shareware version online and give it a try, the URL is: http://www.arkeia.com/.

Pro's and Con's of Backup Media

There are more things to back data up onto then you can drive a range rover over but here are some of the more popular/sane alternatives:

Name of Media	Pro's	Cons
Hard Drive	It's fast. It's cheap. It's pretty reliable. ($20-$30 USD per gig)	It might not be big enough, and they do fail, usually at the worst possible time. Harder to take offsite as well. RAID is a viable option though. 20 gig drives are $350 USD now.
CDROM	Not susceptible to EMP, and everyone in the developed world has a CDROM drive. Media is also pretty sturdy and cheap ($2 USD per 650 Megs or so)	CDROM's do have a finite shelf life of 5-15 years, and not all recordables are equal. Keep away from sunlight, and make sure you have a CDROM drive that will read them.
Tape	It's reliable, you can buy BIG tapes, tape carousels and tape robots, and they're getting cheap enough for almost everyone to own one.	Magnetic media, finite life span and some tapes can be easily damaged (you get what you pay for), also make sure the tapes can be read on other tape drives (in case the server burns down....).

Floppies	I'm not kidding, there are rumors some people still use these to backup data.	It's a floppy. Whaddya think?
Zip Drives	I have yet to damage one, nor have my cats. They hold 100 megs which is good enough for most single user machines.	Not everyone has a zip drive, and they are magnetic media.
Jazz Drives	1 or 2 gig removable harddrives, my SCSI one averages 5 meg/sec writes.	They die. I'm on my third drive. The platters also have a habit of going south if used heavily.
Syquest	1.6 gigs, sealed platter, same as above.	Sealed cartridges are more reliable. Company did recently declare bankruptcy though.
LS120	120 Megs, and cheap, gaining in popularity.	Slooooooooow. I'm not kidding. 120 megs over a floppy controller to something that is advertised as "up to 3-4 times faster then a floppy drive".
Printer	Very long shelf life. requires a standard Mark 1 human being as a reading device. Handy for showing consultants and as reference material. Cannot be easily altered.	You wanna retype a 4000 entry password file? OCR is another option as well.

X Window System

The X Window System provides a network transparent method for sharing graphical data, or more specifically for exporting the display of a program to a remote (or the local) host. Using it you can run a powerful 3d rendering package on your SGI origin 2000 and display it on a 486. Essentially it's the granddaddy to all this 'thin client' hype that is becoming very popular nowadays. It was created by MIT, and at the time security was not much of a concern, this of course has led to more then a few nasty bugs being found, as well the level of control X is given (it handles keystrokes, mouse movements, draws the screen, etc) means if it is compromised very bad things can happen. This data, if sent over the network (i.e. the X program being run is displaying on a remote host) can easily be logged, so sensitive information (like an xterm being used to login to another remote system) is vulnerable. In addition to these problems the authentication protocol that X uses is relatively weak (although it has been improved). Running a graphical xemacs session on a server 3 timezones away however can be a very handy thing.

X is very predictable in port usage, almost all implementations and installations of X use port 6000, thus making it quite easy to scan for. If you are not going to be using X to display program running on remote systems I suggest strongly you firewall port 6000. Control over who/what is allowed to connect to the X server can be accomplished several ways.

xhost

xhost simply allows you to specify which machines are, or aren't allowed to connect to the X server, this is a very simplistic security mechanism and is not really suitable in any modern environment, however used in conjunction with other mechanisms it can help. The command is quite simple: '`xhost +hostname.com`' adds hostname.com, '`xhost -hostname.com`' removes hostname.com from the list, you must also specify '`xhost -`' to turn on the access control list, or else everyone is let in by default.

mkxauth

mkxauth is definitely a step up, it helps create .Xauthority files, and merge them, which are used to specify hostnames and the related magic cookies (basically a token used to gain access). These cookies can then be used to gain access to a remote X host (you essentially have a copy of the cookie on each end) and are transfered either plain text (insecure) or DES encrypted (quite secure). Using this method you can be relatively safe and secure. Xauthority files can also be used in conjunction with Kerberoes, removing the necessity to copy Xauthority files around and keep them in synchronization. Hosts authenticate to each other through a central Kerberoes key server(s) in an encrypted fashion, this method is most appropriate for large installations/etc. mkxauth has an excellent man page '`man mkxauth`' and more generalized details are available in the Xsecurity man page (not sure how common this name page is) '`man Xsecurity`'.

SSH

SSH can be used to create a tunnel between hosts (or more specifically between two X servers), thus encrypting the channel, providing authentication, and generally making things

safer. The following web page explains it in detail:
http://csociety.ecn.purdue.edu/~sigos/projects/ssh/forwarding/.

Distribution specific tools

RedHat

Not yet written.

Debian

Not yet written.

Slackware

Not yet written.

Caldera

Not yet written.

SuSE

One of SuSE's employees (Marc Heuse) has written a few useful utilities for SuSE Linux, available at: http://www.suse.de/~marc/. The first one is called "Harden SuSE" and basically goes about removing sharp objects, tightening up file permissions, turning off daemons and so on. The second one "SuSE security check" is a set of shell scripts that check the password file for sanity, lists out all installed packages once a month and so on.

Distribution specific errata and security lists

RedHat

Errata
http://www.redhat.com/support/docs/errata.html
Security
http://www.redhat.com/support/docs/errata.html
Mailing lists
http://archive.redhat.com/

Debian

Errata
http://www.debian.org/distrib/packages/
Security
http://www.debian.org/security/
Mailing lists
http://www.debian.org/MailingLists/subscribe/

Slackware

Errata
ftp://ftp.cdrom.com/pub/linux/slackware-current/ChangeLog.txt
Security
ftp://ftp.cdrom.com/pub/linux/slackware-current/ChangeLog.txt
Mailing Lists
NO URL

Caldera

Errata
http://www.calderasystems.com/support/download.html
Security
http://www.calderasystems.com/news/security/index.html
Mailing Lists
http://www.calderasystems.com/support/forums.html

SuSE

Errata
http://www.suse.de/e/patches/
Security
http://www.suse.de/security/
Mailing Lists
http://www.suse.com/Mailinglists/index.html

TurboLinux

Errata
http://www.turbolinux.com/support/solutions.html
Security
http://www.turbolinux.com/support/solutions.html
Mailing Lists
NO URL

Stampede GNU/Linux

Errata
ftp://ftp.stampede.org/current/README.CHANGES
Security
ftp://ftp.stampede.org/current/README.CHANGES
Mailing Lists
http://www.stampede.org/maillists.php3

Mandrake

Errata
http://www.linux-mandrake.com/en/fupdates.html
Security
http://www.linux-mandrake.com/en/fupdates.html
Mailing Lists
http://www.linux-mandrake.com/en/flists.html

LinuxPPC

Errata
NO URL
Security
NO URL
Mailing Lists
http://lists.linuxppc.org/

Linux Pro

Errata
NO URL
Security
NO URL
Mailing Lists
NO URL

LinuxWare

Errata
NO URL
Security
NO URL
Mailing Lists
NO URL

MKLinux

Errata
NO URL
Security
NO URL
Mailing Lists
http://www.mklinux.org/mailinglists.html

Yggdrasil

Errata
NO URL
Security
NO URL
Mailing Lists
NO URL

Connectiva

Errata
NO URL
Security
NO URL
Mailing Lists
NO URL

DLD

Errata
NO URL
Security
NO URL
Mailing Lists
NO URL

Eagle Linux M68K

Errata
NO URL
Security
NO URL
Mailing Lists
NO URL

Eurielec

Errata
NO URL
Security
NO URL
Mailing Lists
NO URL

Kheops Linux

Errata
NO URL
Security
NO URL
Mailing Lists
NO URL

MNIS Linux

Errata
NO URL
Security
NO URL
Mailing Lists
NO URL

Appendix A: Books, Magazines and other

Sendmail - http://www.oreilly.com/catalog/sendmail2/
Linux Network Admin Guide (NAG) - http://www.oreilly.com/catalog/linag/
Running Linux - http://www.oreilly.com/catalog/runux2/noframes.html
DNS & BIND - http://www.oreilly.com/catalog/dns3/
Apache - http://www.oreilly.com/catalog/apache2/
Learning The Bash Shell - http://www.oreilly.com/catalog/bash2/
Building Internet Firewalls - http://www.oreilly.com/catalog/fire/
Computer Crime - http://www.oreilly.com/catalog/crime/
Computer Security Basics - http://www.oreilly.com/catalog/csb/
Cracking DES - http://www.oreilly.com/catalog/crackdes/
Essential System Administration - http://www.oreilly.com/catalog/esa2/
Linux in a nutshell - http://www.oreilly.com/catalog/linuxnut2/
Managing NFS and NIS - http://www.oreilly.com/catalog/nfs/
Managing Usenet - http://www.oreilly.com/catalog/musenet/
PGP - http://www.oreilly.com/catalog/pgp/
Practical Unix and Internet Security - http://www.oreilly.com/catalog/puis/
Running Linux - http://www.oreilly.com/catalog/runux2/
Using and Managing PPP - http://www.oreilly.com/catalog/umppp/
Virtual Private Networks - http://www.oreilly.com/catalog/vpn2/

RedHat/SAMS also publish several interesting books:
Maximum RPM (available as a postscript document on www.rpm.org)
RedHat User's Guide (available as HTML on ftp.redhat.com)

SNMP, SNMPv2 and RMON - W. Stallings (ISBN: 0-201-63479-1)

Magazines:
Linux Journal (of course, monthly)
Sys Admin (intelligent articles, monthly)
Perl Journal (quarterly)

Appendix B: ftp/www sites and online resources

Distributions

http://www.calderasystems.com/ – Caldera OpenLinux
http://www.redhat.com/ – RedHat
http://www.suse.com/ – SuSE
http://www.debian.org/ – Debian
http://www.slackware.org/ – Slackware
http://www.turbolinux.com/ - TurboLinux
http://www.stampede.org/ - Stampede GNU/Linux
http://www.linux-mandrake.com/ - Mandrake
http://www.linuxppc.org/ - LinuxPPC
http://www.wgs.com/ - Linux Pro
http://www.trans-am.com/ - LinuxWare
http://www.mklinux.apple.com/ - MKLinux
http://www.yggdrasil.com/ - Yggdrasil
http://www.conectiva.com.br/cl/index.html - Connectiva
http://www.delix.de/ - DLD
http://www.eagle-cp.com/www/m68k.html - Eagle Linux M68K
http://www.etsit.upm.es/~eurielec/linux/ - Eurielec
http://www.linux-kheops.com/ - Kheops Linux
http://www.mnis.fr/ - MNIS Linux

Network / Host Scanning / Intrusion

http://www.insecure.org/nmap/ - nmap
http://david.weekly.org/code/ - ftpcheck, relaycheck
http://www.marko.net/cheops/ - cheops
http://www.nessus.org – nessus
http://www.wwdsi.com/saint/ - saint
http://www.haqd.demon.co.uk/security.htm - SBScan
http://www.infowar.co.uk/mnemonix/ntinfoscan.htm – Samba scanner, runs on windows
http://www.cri.cz/kra/index.html - HUNT

Administration

http://www.rpm.org/ – rpm, rpm book
ftp://missinglink.darkorb.net/pub/rhlupdate/ - RedHat RPM upgrade scripts
http://www.psionic.com/abacus/logcheck/ - Logcheck
http://blue.dhs.org/bgcheck/ - bgcheck
http://www.coas.org/ - COAS
http://www.webmin.com/webmin/ - Webmin
http://www.solucorp.qc.ca/linuxconf/ - Linuxconf
ftp://ftp.ucolick.org/pub/users/will/ - super
http://www.suse.com/ - YaST

Firewalling

http://www.nerdherd.org/ipchains/ - ipchains setup made easier

Network Services

http://www.sendmail.org – sendmail
http://www.apache.org – apache
http://www.isc.org - BIND/DHCPD/DHCPCD
http://www.core-sdi.com/ssyslog/ - secure syslog
http://www.openssl.org - openssl
http://www.ncftp.com -ncftp, ncdftp
http://www.qmail.org – qmail
http://www.postfix.org – postfix
http://www.proftpd.org - ProFTPD
http://www.samba.org/rsync/ - rsync
http://www.interweft.com.au/other/ppp-howto/ppp-howto.html - PPP

Virtual Private Network Solutions
http://sites.inka.de/~W1011/devel/cipe.html - CIPE
http://eclipt.uni-klu.ac.at/projects/est/ - ECLiPt
http://www.redcreek.com/products/index.html - RedCreek VPN hardware products (IPSec)
http://pweb.netcom.com/~popov/index.html - RedCreek VPN card drivers for Linux
http://www.moretonbay.com/vpn/pptp.html - Linux PPTP implementation
http://www2.kame.net/~freeswan/ - FreeS/WAN, a Linux IPSec implementation

SSL
http://www.ssleay.org/ - SSLeay site
http://www.openssl.org/ - open SSL implementation site
ftp://ftp.uni-mainz.de/pub/internet/security/ssl/ - SSL tarballs and SSL'ified apps
ftp://ftp.psy.uq.oz.au/pub/Crypto/ - official SSLeay ftp site
http://www.apache-ssl.org/ - Apache SSL free implementation

Service / Data Encryption
http://www.net.lut.ac.uk/psst/ - free SSH implementation
http://www.gnupgp.org/ - free pgp replacement
http://www.cryptography.org/ - CFS

Security Tools
http://www.magma.ca/~tranter/linux.html – audit

Kernel
http://www.cse.ogi.edu/DISC/projects/immunix/StackGuard/ - stackguard
http://agn-www.informatik.uni-hamburg.de/people/1ott/rsbac/ - rule based access control

Intrusion Detection
http://www.psionic.com/abacus/portsentry/ - port sentry
http://www.psionic.com/abacus/hostsentry/ - host sentry
http://www.nfr.com/ - Network Flight Recorder

Online Resources / Other
http://www.geek-girl.com/bugtraq/ - Bugtraq, excellent mailing list

Security Response Groups
http://www.cert.org - USA - CERT
http://www.hert.org - INTERNATIONAL - HERT

http://www.auscert.org.au - AUSTRALIA - AUSCERT
http://www.singcert.org.sg SINGAPORE – SINGCERT

Security Websites
http://www.infowar.com - Winn Schwartau's
http://www.l0pht.com - L0pht
http://www.rootshell.com - Root Shell
http://www.infowar.co.uk - Infowar UK
http://www.cert.org/security-improvement/ - good documents

Commercial backup programs
http://www.estinc.com/ - BRU
http://www.estinc.com/ - Quickstart
http://www.arkeia.com/ - Arkeia
http://www.unitrends.com/ - CTAR
http://www.unitrends.com/ - CTAR:NET
http://www.unitrends.com/ - Backup Professional

Glossary / term definitions

ACL	Access Control List
ATM	Asynchronous Transfer Mode
CIDR	Classless InterDomain Routing
CRC	Cyclic Redundancy Check
DES	Data Encryption Standard
DNS	Domain Name System
FTP	File Transfer Protocol
GID	Group IDentifiers
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure (a.k.a. SHTTP)
ICMP	Internet Control Message Protocol
IDEA	International Data Encryption Algorithm
IGMP	Internet Group Message Protocol
IMAP	Internet Mail Application Protocol
IP	Internet Protocol
IPSec	IP Security - encrypted IP traffic
IPv6	IP version 6 - 128 bit addresses, various new security features
IRC	Internet Relay Chat
LAN	Local Area Network
MAN	Municipal Area Network
MCP+I	Microsoft Certified Professional with Internet
MCSE	Microsoft Certified Systems Engineer (or "must consult someone experienced" is my favourite alternative)
NDA	Non Disclosure Agreement
NFS	Network File System
NIS	Network Information Service
NIST	National Institute for Standards and Technology
NNTP	Network News Transfer Protocol
NTP	Network Time Protocol
POP	Post Office Protocol
RPC	Remote Procedure Call
SGID	Set Group ID
SGML	Standardized Generic Markup Language
SHTTP	Secure Hyper Text Transfer Protocol (a.k.a. HTTPS)
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SSL	Secure Sockets Layer
SUID	Set User ID
TCP	Transfer Control Protocol
TCP-IP	Transfer Control Protocol - Internet Protocol
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol
UID	User IDentifiers
UUCP	Unix to Unix CoPy
VPN	Virtual Private Network
WAN	Wide Area Network
WWW	World Wide Web

XML	eXtensible Markup Language

Version history

0.0.1	Initial draft, structure decided, basic information from older writings inserted. 04/01/1999.
0.0.2	Major spell checking, formatting changes. The book covers most major topics generally and with RedHat specific instructions. Added ipchains examples where ipfwadm examples are given. 04/03/1999.
0.0.3	Started Appendix B (www sites/etc.) and the glossary. Polished up some sections, added some new ones. Table of contents created. 04/07/1999.
0.0.4	General cleanup, added sections on Encryption, IPSec and the like. 04/11/1999
0.0.5	Intrusion scanning tools and detection, packet sniffing, added some new sections. Edited some older section in need of cleanup. Got rid of unnecessary graphics which removed about 200k (currently half the bulk). 04/12/1999
0.0.6	Added audits and baselines, inn, cvs, rsync, added some new tools in various sections (ssh, etc.). 04/16/1999
0.0.7	Added some new section titles (Network Based Authentication, X Window System, PAM, etc.), finished some other sections (PPP, Linuxconf, etc) and added some more commercial backup programs. Added a whole whack of Linux vendors. Tools like YaST, super, Linuxconf also added. 04/19/1999
0.0.8	Added some more sections (dpkg, tarballs/tgz's, tftp, etc.), added some more tools (ntop, etc). Actually installed Debian 2.1 and spent a few days playing with it. Now to experience Slackware for 0.0.9. Some minor reorganization (SAMBA is it's own subject now as well as Novell connectivity). 04/22/1999